Linguistic problem ranking for people with Dyslexia using crowdsourcing

BACHELOR’S THESIS

Author: Clara BAYARRI

Supervisor: Carles FARRÉ
Department of Service and Information System Engineering

Bachelor Degree in Informatics Engineering
Software Engineering Specialization
Barcelona School of Informatics - FIB

June 26th, 2013
Crowdsourcing is a technique by which you delegate the work of solving tasks to human users for problems that are computationally difficult to solve or that we do not have an algorithm for. In the past few years, the field of crowdsourcing has grown to be a very active area of research. However, authors do not seem to agree on what can and cannot be considered crowdsourcing, or what common characteristics crowdsourcing problems have. We intend to provide a first approach towards a practical analysis of crowdsourcing, by the means of building actual crowdsourcing systems and extracting their common features.

This project presents a first crowdsourcing system aimed to study the possible generalization of many systems into one common platform. The problem taken to solve involves ranking linguistic exercises for children with dyslexia by analyzing the data collected from exercise completion by users.

Dyslexia is a cognitive disability suffered by around 10% of children worldwide that affects their reading and writing abilities. Research in this area has been broad, but there are few tools directly designed for children with dyslexia. The author of this project collaborates with a researcher in this field, developing tools to help children with dyslexia overcome their difficulties at a young age.

We present a project that builds an online game capable of presenting linguistic exercises to users and collecting their data for later analysis, relying on a generic crowdsourcing platform capable of defining crowdsourcing tasks and storing large volumes of data for analysis. The project produces two functional systems that work together towards a common goal, collecting data on the performance of users on linguistic exercises to better tailor real exercises on a tool for children with dyslexia.

El crowdsourcing és una tècnica que delega la resolució de tasques difícils per una màquina a un conjunt d'usuaris. En els últims anys aquest camp de recerca ha guanyat importància, però sembla ser que els investigadors no es posen d'acord en què consideren crowdsourcing i què no, o en quines característiques comunes segueixen els sistemes de crowdsourcing. Ens proposem fer una primera aproximació pràctica a l’anàlisi de característiques dels sistemes de crowdsourcing a partir de la construcció de casos d'ús reals.

Aquest projecte presenta la construcció d’un primer sistema de crowdsourcing amb l’objectiu d’analitzar-ne la possible generalització cap a una plataforma genèrica de resolució de problemes de crowdsourcing. El problema que ens plantejem solucionar és la valoració de la utilitat d’exercicis lingüístics per nens amb dislèxia a partir de l’anàlisi de dades recollides sobre el comportament dels usuaris amb aquests exercicis.

La dislèxia és un transtorn cognitiu que pateixen vora el 10% dels nens del món i que afecta les seves habilitats de lectura i escriptura. S’ha fet molta investigació en el camp de la dislèxia, però existeixen molt poques eines dirigides directament a nens amb dislèxia. La autora d’aquest projecte collabora amb una investigadora en aquest camp, amb qui ha desenvolupat eines que ajuden als nens amb dislèxia a millorar les seves habilitats de lectura i escriptura des de ben petits.

El projecte aquí presentat descriu la construcció d’un joc en línia capaç de mostrar exercicis lingüístics de manera interactiva als usuaris i recollir-ne dades d’ús per un futur anàlisi. El joc depèn d’una plataforma genèrica de crowdsourcing capaç de definir tasques a resoldre i emmagatzemar grans volums de dades per ser analitzades. El projecte inclou dos sistemes funcionals que cooperen per arribar a un objectiu en comú, la recollida de dades sobre el comportament d’usuaris davant exercicis lingüístics per poder millorar els exercicis que apareixen en una eina existent per a nens amb dislèxia.
I would like to thank my supervisor Carles for his help all along the evolution of this project, from suggesting the initial ideas to helping me with the final details. It has been a fun journey, and I hope you feel proud of the result as much as I do.

In second place I must greatly thank Luz and Azuki, my teammates and co-creators of Dyseggxia. Working with you both makes work so much fun it is hard to believe, and your enthusiasm is contagious. Let this be just another step in our long path together.

I would also like to thank all friends and family who have helped me through the stressful days, and specially specially thank all those people who have volunteered to help with data collection by registering in the game and solving exercises. All this effort would be to waste if there weren’t people like you :)}
3.2 Individual use cases ........................................... 25
  3.2.1 Crowd platform ........................................... 25
  3.2.2 Game ....................................................... 41
3.3 Conceptual model .............................................. 50
  3.3.1 Crowd platform ........................................... 50
  3.3.2 Game ....................................................... 54
3.4 Behavior model ................................................ 58

4 Design Overview .............................................. 74
  4.1 Architecture .................................................. 74
    4.1.1 Spring Framework ....................................... 74
  4.2 Technology selection ......................................... 76
    4.2.1 Framework ............................................... 76
    4.2.2 Storage ................................................... 76
    4.2.3 Interface interaction ..................................... 76
    4.2.4 Graph representation ..................................... 77
    4.2.5 Data visualization ........................................ 77

5 Crowd Platform Design ....................................... 78
  5.1 Domain model .................................................. 78
  5.2 Interaction Design ........................................... 82
  5.3 Remote API ................................................... 95
  5.4 Interactions ................................................... 96
  5.5 Controllers ................................................... 106
  5.6 Views ........................................................ 136
  5.7 Services ....................................................... 139
  5.8 Utils .......................................................... 144
  5.9 Packages ....................................................... 148
  5.10 Data model .................................................... 149
    5.10.1 Map Reduce .............................................. 151

6 Game Platform Design ........................................ 153
  6.1 Domain model .................................................. 153
  6.2 Interaction Design ........................................... 158
10 Results

10.1 Solution ......................................................... 225
10.2 User feedback .................................................. 226
10.3 Data analysis .................................................... 227
10.4 Conclusions ..................................................... 232
10.5 Future work ..................................................... 234

11 References ...................................................... 235

12 Appendix .......................................................... 237

12.1 Database schema for the Game system ....................... 237
12.2 Gantt Diagrams ................................................ 241
  12.2.1 Initial planning ........................................... 241
  12.2.2 Control planning ......................................... 245
  12.2.3 Final planning ............................................ 249
12.3 Initial budget calculation ...................................... 253
12.4 Data analysis script ........................................... 255
1 Introduction

This document presents the elaboration of a crowdsourcing tool capable of collecting data on linguistic exercises to evaluate their effectiveness when given to children with dyslexia.

Crowdsourcing is a recently popular technique by which technological systems solve problems that are difficult to solve computationally by handing them out to a crowd of human users. There are some problems that we simply can’t ask a computer to solve, and by handing them out directly or indirectly to human users we can obtain fast results.

Dyslexia is a cognitive disability suffered by around 10% of children worldwide that affects their reading and writing abilities. Research in this area has been broad, but there are few tools directly designed for children with dyslexia. The author of this project collaborates with a researcher in this field, developing tools to help children with dyslexia overcome their difficulties at a young age.

The problem we are presented with is related to evaluating what linguistic exercises are best to include in a tool for children with dyslexia. We believe this problem is hard to solve from a theoretical point of view, but maybe by analyzing the performance of dyslexic and non-dyslexic users on exercises we can achieve some results as to what can help in reeducating dyslexia more effectively. Given that the best data we can obtain from users is not the data they give us consciously but the data they really generate in front of exercises, we here present the construction of a tool that presents linguistic exercises to users and collects data using the crowdsourcing technique.

Furthermore, we are also presented with the problem of building a generic crowdsourcing platform capable of supporting many kinds of crowdsourcing tasks, all with the goal of analyzing common characteristics that can be generalized from crowdsourcing problems. We believe building several concrete crowdsourcing projects and having them share a common platform is the best way to achieve this generalization, and therefore also present the construction of a platform that will support our previously mentioned tool and that will generalize the common characteristics found when comparing the tool with other crowdsourcing problems.

The sections that follow present the complete elaboration of a project that solves both problems. The first sections are centered in the software engineering process. We will first present an extensive analysis, where we have documented the influences on the project, including the problems to be solved, the requirements set by the stakeholders, the existing systems and the goals to be met, all in section 2. Once this is settled, section 3 presents the specification of the functionality that the project will cover, separating the work to be done in two systems. A general architecture overview will be covered in section 4, followed by two design sections stating all the design details of each of the two systems produced, sections 5 and 6. To finish the technical side of the document, we will cover deployment and testing in sections 7 and 8.

This document also includes extensive information on the project management, included in section 9. This includes the complete planning for the project and its budget, as well as other topics such as laws and regulations.

Finally, we present in section 10 the results of the project, reflecting to see if the goals are met and briefly analyzing the feedback and data gathered, as well as reaching the final conclusions and stating possible future work.
2 Analysis

The following analysis has been done following the Volere stencil.

2.1 Goals

This project will produce two interrelated products: a generic crowdsourcing platform and an online game that relies on this platform. The concrete goals for the project are as follows:

- Provide an engaging game that attracts users and encourages them to solve linguistic exercises by May 2013.
- Provide a first approach to a generalized Crowdsourcing platform that allows the creation of other platforms from its structure by May 2013.
- Provide a tool that allows the Dyseggxia team to upload exercises and obtain statistical data about their performance with users by May 2013.
- Develop a functional system that can be presented as a thesis project by June 2013.

2.2 Stakeholders

There are several actors interested in the development and outcome of this project. Specifically, we have detected four main actors that are directly involved with the project and will effect its evolution.

Firstly, we must consider the author as the main actor in this project. The author of this project is responsible for its execution, and aims to complete a thesis project and present it before the specified deadline, fulfilling all the goals that have been stated for the project.

Secondly, we must take into account the Crowdsourcing Research Group at UPC. This is the research group where the supervisor of the project is involved, and its members are interested in obtaining a first approach to a generalized crowdsourcing platform for their research. Therefore, we must take into account their needs and their suggestions on the project, as they will be a direct beneficiary of the final platform.

Thirdly, we must consider the team behind Dyseggxia. They are the client who is interested in the final product, in order to obtain data about the use of their exercises and then analyze it. They have a strong opinion on the game side of the project, and will be introducing restrictions in this part in order to fully fulfill their needs. Furthermore, they will be the direct users of the crowdsourcing platform and will be interested in having a friendly interface that allows them to easily upload tasks and retrieve results.

Finally, we have the game players. These will be the main users of the game in the project, and indirectly of the crowdsourcing platform. If we want to recruit them properly and keep them solving exercises, they will want an engaging game that encourages them to keep playing.

These four actors are the ones directly involved with the project, and will have a direct influence and impact on its development. Nevertheless, there are other smaller actors that may indirectly be affected by our project and must be taken into account, such as the legal authorities interested in the data we collect from users or the university where this project is being developed.
2.3 Scope of the work

2.3.1 The current situation

The supervisor for this project is a member of a research group at UPC that is studying the nature of crowdsourcing platforms. His group is interested in building a generic crowdsourcing platform that could serve as a basis for the construction of other crowdsourcing platforms. This task includes analyzing the common characteristics between different types of platforms in order to generalize, and they therefore decided to start by building several concrete cases and extracting the common behaviour from them. This project will be the first of these concrete cases, and will make the first step towards the development of a generic crowdsourcing platform.

Dyseggxia [1] is a mobile game developed to encourage children with dyslexia to work on their linguistic difficulties by playing with word exercises. The team behind Dyseggxia has carefully designed the process needed to generate exercises and make sure that the content included in the application targets the specific needs of dyslexic children [2].

Nevertheless, the exercise creation pipeline has one flaw: words used in exercises are hand-picked from the list of most common words from the language used, and then matched with specific difficulties observed in children with dyslexia at the team’s choice in order to generate an exercise. Even though both elements, word and difficulty, are obtained carefully from certain studies and parameters, the matching is done with no guarantee that it is the best possible combination. Therefore, the team behind Dyseggxia is searching for a means to find out what combinations of words and difficulties are best as an exercise.

The idea for this project is to combine these two needs in order to produce an online crowdsourcing application that will help the Dyseggxia team obtain analytical data on exercise performance and, at the same time, will provide the research group at UPC a first version of a generic crowdsourcing platform.

This project will have a big impact in the process done by the team behind Dyseggxia when generating exercises. I believe this is a first approach to a scientifically based process that will allow the team to further justify the rigor of their pipeline and will potentially help to better address the needs of children with dyslexia.

Furthermore, the project will be of impact to the crowdsourcing research group at UPC, as it will give them a first concrete case of a crowdsourcing system as well as a first approach to a generic crowdsourcing platform.

2.3.2 State of the art

In order to picture the context of this project, we must analyze the state of the art, discovering previous work done in the field as well as existing products that can be related or of use to the project. Our project has two clearly differentiated domains of knowledge, one related to crowdsourcing systems and the other related to the game Dyseggxia and its research, and we must analyze both in order to obtain a complete image of the context of the project.

Crowdsourcing taxonomies

Crowdsourcing is a broad research area that is currently the focus of many studies. Even though there is no consensus on an exact definition of what it is, we can consider that a Crowdsourcing
System is a system that recruits a crowd of humans to solve a problem and addresses the challenges involving how to recruit the users, what contributions it allows them to make, how to combine user contributions and how to evaluate users and their contributions [3].

The goal of the Crowdsourcing Research Group is to further study this subject, and we must therefore consider the previous work that has been done in order to define crowdsourcing platforms and classify them.

Several authors have attempted to specify a taxonomy for crowdsourcing platforms, analyzing their main characteristics. We take the definition given by Geiger et al.[4] as a base to describe what a crowdsourcing system is in essence, as described in figure 1.

Once this definition is settled, we can move on to find common characteristics and options for crowdsourcing platforms.

Doan et al.[3] consider nine dimensions that are important in order to classify crowdsourcing systems:

- Nature of collaboration
- Type of target problem
- How to recruit and retain users
- What users can do
- How to combine user input
- How to evaluate user input
- Degree of manual effort
- Role of human users
- Standalone or piggyback

As we can see, this leads to a high number of possibilities for crowdsourcing platforms. From a different perspective, Geiger et al.[4] consider only 4 dimensions, as stated in figure 2. These are:

- Preselection of contributors
- Accessibility of peer contributions

Figure 1: Simplified definition of the process executed by any crowdsourcing system, according to Geiger et al.
As we can see, this is a much more simplified approach to the classification, and may be more accessible from the point of view of designing a generic platform than the previous classification. If we consider the concrete case we will develop in this project according to the classification by Geiger et al., we will state the following characteristics:

- Pre-selection of contributors: context-specific, as we will separate dyslexic users from common users.
- Accessibility of peer contributions: not visible.
- Aggregation of contributions: integrative.
- Remuneration for contributions: none.

The research by Geiger et al. further analyzed this process of classification by comparing existing crowdsourcing systems and analyzing the most common combinations of characteristics. The concrete case addressed in this project falls into one of these combinations, named by the authors *Integrative sourcing without remuneration* [4].

There have been some attempts to approach the classification of crowdsourcing systems from a practical point of view as we intend to do. A clear example is the work done by Bozzon et al.[5], who developed a brand logo image detection game for Facebook that served as a basis for further logo detection in videos. This work was done with an abstract generalization of a crowdsourcing platform in mind, with a first approach that can be very useful for our work.
**Figure 3:** The CrowdSearch Framework, an approach to crowdsourcing system generalization by Bozzon et al.

**Amazon Mechanical Turk**

Amazon Mechanical Turk [6] is currently the most important crowdsourcing platform in the market. As a platform, it serves as an intermediary between users willing to perform crowdsourcing tasks and users with the need for human crowd workforce.

Users who are in need of human crowd workforce can create batches of tasks on Amazon Mechanical Turk, define how much they are willing to pay for each completed task and, after some time, obtain the results from users. Furthermore, the system allows remote connection through their APIs in order to adapt the use of Mechanical Turk to other systems. Users who complete tasks are taken from a broad user base registered on the system, and are classified by skills in order to assign them to the appropriate tasks.

Amazon Mechanical Turk is currently being used in many crowdsourcing systems, as it provides easy access to a big community of workers. There is a great variety of systems that use Mechanical Turk in order to gather data, from labels for images to answers on general knowledge.

An interesting example can be found in the database management system designed by Franklin et al.[7]. When a query is sent to the system that cannot be answered with the existing data and would produce an empty response, the system creates a task on Amazon Mechanical Turk and resorts to the crowd in order to find a response.

Nevertheless, we can see that Amazon Mechanical Turk is only useful for some types of crowdsourcing tasks, as it forces users to define the tasks to be completed beforehand, its workers are aware of the work they are producing and its workers always expect remuneration for their work.
Games with a purpose

Even though some crowdsourcing definitions do not consider games to be eligible crowdsourcing systems [3], there has been a long attempt to develop games with a purpose, a means of disguising crowdsourcing task executions behind a game interface. As stated by Von Ahn and Dabbish, these games enable human players to perform tasks computers are unable to perform as a side effect of playing [8].

The most referenced example in games with a purpose is the ESP game [9], also known as the Google Image Labeler. The goal of this crowdsourcing system is to label images found in Google Images. This is a task that cannot be easily completed by computers, and that would be too expensive if we consider paying users to manually label images. Therefore, the authors conceived a game where players, as a side effect of playing, produced meaningful labels for images taken from Google Images. In this game, players were challenged to find a common label in order to describe an image presented, and therefore the labels produced were meaningful.

Von Ahn and Dabbish classify the existing games with a purpose into 3 types of games, named Output-agreement, Inversion-problem and Input-agreement [8]. Nevertheless, these 3 types of games always involve the work of players in pairs and in competition, and do not fit the format of our intended game.

Dyslexia

Finally, we may talk about the existing systems addressed to people with dyslexia. Even though this is a broad field, we must focus on the existing systems in reeducation of dyslexic children. According to its authors, Dyseggxia is the first reeducation tool that has a scientific basis in the generation of its exercises [2].

Dyseggxia follows a mainly manual pipeline in the generation of the exercises included in the application. First, a collection of texts handwritten by children with dyslexia is analyzed in order to obtain errors in their spelling as well as repeated patterns formed by these errors. Secondly, the authors select words taken from the list of most common words in a language according to certain criteria. Finally, an exercise is formed by manually combining the designed exercise types, the observed errors and the selected words.

As far as we know, there are no existing tools that could help automatize this process while keeping the scientific basis of the exercise generation pipeline, or that could improve it by providing empirical data on what combinations of types, errors and words are most effective as exercises. In order to obtain real empirical data on the effectiveness of possible exercises, we would need to recreate the context in which final users will find the exercises, by providing a similar interface and game-play. Even though we could consider Amazon Mechanical Turk as an option, the fact that users would be aware of their monitoring and remuneration would change the perception and interaction of users with exercises, and would therefore invalidate the data collected.

2.3.3 The new situation

The system developed in this project will provide the Dyseggxia team with a tool to analyze potential exercises for their product. The team will be able to introduce potential exercises into the system and collect data from their execution for later analysis.

We will also develop a new generic crowdsourcing platform that will collect, store and analyze data gathered from crowdsourcing tasks and will be generic enough to serve other kinds of crowdsourcing
problems that are not related to Dyseggxia.

The generic crowdsourcing platform will allow by the end of this project to upload tasks, provide them to an external system in charge of executing them, register users to relate with task executions, collect execution data, provide progress information, provide data visually and download raw data.

2.3.4 The context of the work

![Diagram]

**Figure 4:** The context of the work.

The work to be done consists on several phases, although not all of them will be covered in this project. First, the Dyseggxia team must define the structure that their data will take, both for input (tasks) and output (executions). Then, they must create a set of exercises to be tested. These are shaped as specified, grouped into batches and uploaded as task batches to the platform. Then, the platform distributes these tasks to the players, who solve them and data is collected. Once all tasks have been executed as desired, the team can download the data for each batch, analyze it, and extract the results they need such as what problems are best fit for a certain age range or what letters have more effect.
2.3.5 Work partitioning

**BUC 1**  Administrator adds batch

*Input:* New batch of tasks  
*Output:*  
*Description:* Introduce the new batch of tasks to the system.

**BUC 2**  Administrator starts/stops batch

*Input:* A batch to start/stop  
*Output:*  
*Description:* Start/stop providing the tasks contained in the selected batch.

**BUC 3**  Administrator downloads data

*Input:* A batch to download  
*Output:* The data collected for executions of that batch  
*Description:* Download all the execution data collected related to the selected batch of tasks.

**BUC 4**  Administrator views data

*Input:* A batch to view  
*Output:* A visual representation of the data collected  
*Description:* Show all the execution data collected related to the selected batch of tasks in a visual manner without downloading.

**BUC 5**  Player solves exercise

*Input:*  
*Output:* The data collected from the execution of a task  
*Description:* The system provides a task and the player solves it, while the system collects data.
2.4 Glossary

**Dyslexia**: a developmental disorder which can cause learning difficulties in one or more of the areas of reading, writing, and numeracy.

**Disability**: the condition of being unable to perform a task or function because of a physical or mental impairment.

**Cognitive disability**: disability that affects the way things are processed in the brain. More info: [http://www.muhlenberg.edu/careercenter/emplguide/cognitive.html](http://www.muhlenberg.edu/careercenter/emplguide/cognitive.html)

**Crowdsourcing**: technique used to solve tasks that a computer finds difficult by the means of distributing them to a crowd of human users who can easily solve them.

**Task**: a specific unit to be tested/delivered to users. In our concrete case, a linguistic exercise.

**Linguistic problem/exercise**: task to be solved consisting on a wrong word and possible answers. Linguistic problems in Dyseggxia can take 7 forms:

- Simple insertion: the user has a blank space to fill in with several possible letters in order to form a correct word.
- Hard insertion: the user is presented with a word missing one letter and several options, having to select the correct spot for the correct letter to insert.
- Omission: the user is presented with a word with one or more additional letters that must be removed.
- Substitution: the user is presented with a word containing a wrong letter that must be replaced by another from a list of options.
- Derivation: the user must select the correct termination for a word from a list of possibilities.
- Transposition: the user must reorder letters or syllables to form a word.
- Separation: the user is presented with a set of words that must be separated.

**Batch**: a set of related tasks.

**Execution**: the data collected as a result of giving a specific task to a user once.

**Crowd platform**: interface visible to the administrator of the system and responsible for batch insertion and control.

**Game interface**: interface visible to users who complete tasks.

2.5 Domain properties and hypothesis

Given the nature of our project, we believe it is best to avoid properties and hypothesis we cannot control, as they will affect the results collected. Nevertheless, some factors should be taken into account:

- **Truthfulness**: we must believe that our users will be truthful with the data they provide us. This means that they will not lie about whether they are diagnosed with dyslexia or not or about their age and level in language. Nevertheless, we will want to check the final data to detect anomalies and, if needed, remove suspicious users from the data used for result extraction.
• **User engagement**: we must believe that users will enter our game on their own will and solve exercises. We can try and improve this aspect by providing an engaging experience and promoting our game.

### 2.6 Constraints

#### 2.6.1 Solution constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Web application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The solution must take the form of a web application.</td>
</tr>
<tr>
<td>Rationale:</td>
<td>The client is interested in an easily accessible application, and is therefore demanding a web-based solution.</td>
</tr>
<tr>
<td>Originator:</td>
<td>Crowdsourcing research group, Dyseggxia team</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Crowdsourcing generalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The solution must be as generalized as possible regarding crowdsourcing.</td>
</tr>
<tr>
<td>Rationale:</td>
<td>The produced crowdsourcing platform should be later applicable to other crowdsourcing problems without any modifications.</td>
</tr>
<tr>
<td>Originator:</td>
<td>Crowdsourcing research group</td>
</tr>
</tbody>
</table>

#### 2.6.2 Partner or collaborative applications

The product developed in this project is tightly related to an existing product called *Dyseggxia* [1]. Dyseggxia is a mobile application developed by the author of this project along with a team specialized in developing tools for children with learning disabilities [2]. Dyseggxia presents exercises for dyslexia reeducation in the format of a mobile interactive game aimed to children between 6 and 12 years old. Our project will be a design tool for Dyseggxia, providing analytic data on the use of exercises that can be potentially included in the final game.

At this date, the authors of Dyseggxia, which include the author of this project, have published two papers related to their research, and presented their application widely. Their work has been praised by the community of parents and teachers of dyslexic children, as well as by the research community. Since June 2012, their application has been downloaded more than 6,000 times by users across the world, it has been improved graphically with feedback from users [10] and they plan to further improve their game by making their exercises as effective as possible.
2.6.3 Schedule constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The system should be completed by June 2013.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The system will be presented as a thesis project, and must be complete for its presentation in June 2013.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Clara</td>
</tr>
</tbody>
</table>

2.6.4 Budget constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The system should be completed with the work of one engineer during 4 months.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>There is no real budget for the project, and it must therefore be completed by the author on her own in the schedule defined for the project.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Clara</td>
</tr>
</tbody>
</table>

2.7 Scope of the product

2.7.1 Product context

The system will consist of two domains. On one hand, we will have a generic crowdsourcing platform that will address the needs for creating projects with crowdsourcing tasks. On the other hand, we will develop an online version of the game *Dyseggxia* that allows users to solve the exercises we want to try out.

The crowdsourcing platform will be as generic as possible, allowing further creation of other projects with crowdsourcing needs from its structure. As a complete platform, it should allow users to create projects, upload tasks and obtain the data of executions of those tasks.

The generic crowdsourcing platform developed will include those features that can be extracted from the development of the concrete case addressed in this project. That is, we will only cover the functionality necessary for our concrete case, leaving the rest for future work due to time restrictions. The scope of the project includes the following functionalities:

- Create tasks in batches from data provided by the administrator of the project
- Serve individual tasks to the remote game application on demand
• Create users related to the project as requested remotely by the game application
• Save the results from task execution as returned by the game application
• Show execution results online to the administrator of the project
• Allow the administrator to manipulate results shown by the means of aggregating data
• Allow the administrator to download all the data
• Provide statistical data about the executions in a batch
• Delete batches

Other functionalities are therefore left as out of scope, such as project creation and definition or output generation. A complete crowdsourcing platform would also have to include functionalities related to user remuneration, data visibility to users or user selection according to their context as specified by Geiger et al.[4], but due to time restrictions these will be left as out of scope of this project.

The game will be a simplified online version of the game Dyseggxia, allowing users to complete exercises of the same types as the ones in the existing application. The game application should include all the functionality necessary to provide users with linguistic exercises:

• Register new users when they first access the application
• Allow users to solve exercises by providing them with a visual representation of a task obtained from the platform, allowing user interaction and recording the results of the execution
• Allow users to see their own personal data
• Allow users to delete their personal data

The scope of the project limits some of the ideal functionality for the complete exercise selection pipeline, that can be addressed in future work. This includes generating exercises from lists of words and difficulties, as well as obtaining a final ranked list of exercises. These tasks are the ones that are less related to crowdsourcing, and can mainly be done offline. Therefore, we have selected them to remain out of scope instead of other functionalities.
2.7.2 Product boundary

We identify nine major use cases related to the business use cases stated above that cover the basic functionality required for this project.

![Diagram of product boundary](image)

**Figure 6:** Product boundary

2.8 Functional requirements

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Introducing tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The system must allow the administrators to introduce tasks to be solved.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The administrators want to be able to introduce new tasks regularly into the system and have them solved.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team, Crowdsourcing research group</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Obtaining results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The system must allow the administrators to retrieve the data collected from task execution.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The administrators want to be able to retrieve the data collected in order to analyze it.</td>
</tr>
</tbody>
</table>
Functional requirement Progress visualization

Description: The system must allow the administrators to visualize the progress done on the batches defined.

Rationale: The Dyseggxia team wants to be able to see the progress done in existing batches.

Originator: Dyseggxia team

Functional requirement Execution control

Description: The system must allow the administrators to start and stop providing tasks in a batch.

Rationale: The Dyseggxia team wants to be able to start and stop batches in order to control what contents reaches the game.

Originator: Dyseggxia team

Functional requirement Data visualization

Description: The system must allow its administrators to visualize and manipulate the data collected online.

Rationale: The administrators want to be able to consult and extract simple calculations from data without having to download it all.

Originator: Crowdsourcing research group

Functional requirement Task providing

Description: The system must provide players with tasks to be solved.

Rationale: The users will want to solve new tasks when they enter the game.

Originator: Players, Dyseggxia team
**Functional requirement**  
**Task solving**

*Description:* The system must provide players with a means to solve tasks.

*Rationale:* The final goal is to collect execution data, and we need tasks solved to achieve it.

*Originator:* Players, Dyseggxia team

**Functional requirement**  
**Batch deletion**

*Description:* The system must allow the administrators to delete batches.

*Rationale:* The administrator might upload erroneous data accidentally.

*Originator:* Dyseggxia team

**Functional requirement**  
**User data visualization**

*Description:* The system must provide players with a means to view the personal data they have provided to the system.

*Rationale:* The users will want to see the personal data they have provided to the system.

*Originator:* Legal requirement

**Functional requirement**  
**User deletion**

*Description:* The system must provide players with the ability to delete their user data.

*Rationale:* The users may want to delete their personal data from the system.

*Originator:* Legal requirement
## 2.9 Quality requirements

### 2.9.1 Look and feel requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Game look and feel</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The final developed game interface must be clearly different from the Dyseggxia interface.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The exercises presented in the game developed in this project are not necessarily scientifically designed and verified, and we do not want users to believe that the exercises presented have the same properties as those presented in Dyseggxia.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team</td>
</tr>
</tbody>
</table>

### 2.9.2 Usability and humanity requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Children use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The final developed game interface must be easily usable by children.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The data collected from exercise execution will be most valuable when it is provided by children, as they are the target audience of Dyseggxia. Therefore, the developed game interface must be easily accessible to them.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Simplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The final developed game interface must be simple.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The developed game will receive many visits from people with different kinds of cognitive disabilities. The interface must be as simple as possible in order to make their use easy and fluid.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team</td>
</tr>
<tr>
<td>Requirement</td>
<td>Avoid text</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>The final developed game interface must contain as little text as possible.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The developed game will receive many visits from people with different kinds of cognitive disabilities, specially people with dyslexic disabilities. Text is hard for them to read and comprehend, and therefore should be avoided as much as possible.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The developed game must be in Spanish.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>Given that the problems that will be tested initially will be in Spanish, the developed game should be expressed in this language.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Localization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The developed game must be extensible to other languages.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>In the future, we might want to try exercises in languages other than Spanish, and therefore the game must be ready to be easily translated into other languages and allow changing between them.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Language correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The developed interface must contain correct messages and have a polite tone.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The developed game contains language exercises, and therefore it is not admissible to display grammatical or orthographical errors. Furthermore, the language used must be polite and must not offend users in any way.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team</td>
</tr>
</tbody>
</table>
**Requirement** Accessibility

*Description:* The developed interface must be accessible to people with cognitive disabilities.

*Rationale:* The exercises displayed in the game are targeted to people with cognitive disabilities, and therefore the interface must contribute to make the game accessible to them as much as possible.

*Originator:* Dyseggxia team

### Performance requirements

#### Requirement Speed and latency

*Description:* The developed systems should respond within the standard response time for a web application.

*Rationale:* The systems must be used through the Internet, and should not slow down the user's work.

*Originator:* Dyseggxia team, Crowdsourcing research group

#### Requirement Data recovery

*Description:* Backups of the data in the systems should be done regularly.

*Rationale:* The systems will store personal user data as well as valuable execution data that cannot be lost. Therefore, we must insure that backups are done regularly.

*Originator:* Dyseggxia team, Crowdsourcing research group

#### Requirement Data precision

*Description:* Data collected must be stored in the highest precision possible.

*Rationale:* The data collected through the developed systems will be used in further analysis, and a high precision will be very important. For fields where several precisions are available, we should ensure a reasonable precision, such as milliseconds for dates and times.

*Originator:* Dyseggxia team, Crowdsourcing research group
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Data presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Data collected must be shown with as much precision as possible without interfering with clearness in presentation.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The data shown to the user must be meaningful and precise, but be kept in a precision that does not interfere with an understanding of the data displayed.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team, Crowdsourcing research group</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Reliability and availability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The systems must be reliable and must be available at least 99% of the time.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The developed systems should be at the disposal of users whenever they feel like accessing, and therefore cannot be unavailable.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team, Crowdsourcing research group</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The system must be able to take at least 20 concurrent users.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>By the end of this project, the systems will be in beta testing with real users. Given that there is no budget, we acknowledge that a high concurrency is not achievable, but at least a minimum capacity should be ensured.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team, Crowdsourcing research group</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>The systems should be easily scalable to higher demands.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The systems should be easily scalable, as we foresee that the amount of users accessing them will increase as well as the data collected and stored.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team, Crowdsourcing research group</td>
</tr>
</tbody>
</table>
2.9.4 Operational and environmental requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Beta release</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>A beta version of the game must be released in order to collect feedback from users before public release.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>Given that the whole system will be developed by one person, having feedback from real users will be very valuable to detect problems and anomalies in the game and data collection system before public release.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Clara</td>
</tr>
</tbody>
</table>

2.9.5 Security requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Restricted access</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Access to each system and section must be restricted to the allowed users.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The platform will access, collect and store sensible information, and must therefore control carefully how access is granted to each section.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Dyseggxia team, Crowdsourcing platform, legal restrictions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Laws regarding privacy that affect the systems should be investigated and taken into account.</td>
</tr>
<tr>
<td><strong>Rationale:</strong></td>
<td>The systems to be developed will be under the influence of a legal system that should be considered and respected.</td>
</tr>
<tr>
<td><strong>Originator:</strong></td>
<td>Legal restrictions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Username privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
<td>Game player usernames should not be visible to others.</td>
</tr>
</tbody>
</table>
**Rationale:** In order to analyze the collected data, usernames are not necessary, and should be occluded and hidden from the data provided to the administrator in order to protect the players’ privacy.

**Originator:** Legal restrictions

**Requirement** User data privacy

**Description:** Game player data should only be visible to himself.

**Rationale:** The personal data related to one specific user should never be accessible to other users in the system in order to protect each user’s privacy.

**Originator:** Legal restrictions

**Requirement** Audition

**Description:** Data collected should be stored even if already downloaded by the end user.

**Rationale:** Legal audits may be carried, and data cannot be destroyed.

**Originator:** Legal restrictions

**Requirement** Immunity

**Description:** The systems should not be vulnerable to attacks.

**Rationale:** Within the system’s possibilities, all known popular attacks should be planned and avoided.

**Originator:** Legal restrictions

**Requirement** Spanish LOPD

**Description:** The Spanish law known as LOPD must be complied.

**Rationale:** The systems will work under Spanish legislation and must carry out all the requirements stated by the Spanish law regarding personal data collection.

**Originator:** Legal restrictions
2.10 Risks

During the development of this project we must take into account the risks involved in the process in order to detect their appearance at the earliest stage possible. Some of the most apparent risks are as follows:

- **Time management.** We could not be able to deliver a first version of the platform and game to users before the end of the project. In order to avoid this, we will follow a strict schedule and control its evolution in a weekly manner.

- **Malicious users.** We could find that the users that register for our game are not well-intentioned and try to trick the system. In order to avoid this, we will try to foresee the possible weaknesses of our system and find solutions for them.

- **Data collection.** We may not collect enough data before the end of the project to determine if the platform is of use for the *Dyseggxia* team. In order to avoid this, we will finish the implementation with enough time and will recruit users extensively for testing.

- **Data overflow.** We may receive too many users during the testing phase. In order to avoid this, we will restrict the testing of our platform during its beta phases and will take into account scalability during the development of the project.

- **Incorrect generalization.** We may come up with a generalization of the platform that works properly for our concrete case but does not generalize well to other cases. In order to avoid this, we will take into account other examples of existing platforms during our design, and will think of ways to accommodate their needs as well.
3 Specification

3.1 Actors

We have three types of actors in our system: unknown users, players and administrators. Additionally, since we are creating two platforms that will connect with each other, we must consider the Game platform to be an actor in the Crowd platform context.

Unknown user

An unknown user is anyone who approaches one of our systems without a registered account. Unregistered users will be able to register in our two systems, and therefore become players or administrators.

Player

A player is the user of our game. As a registered user, he has access to the gaming interface and can complete exercises, as well as view his personal data.

Administrator

An administrator is a user of the crowd platform. As such, he has access to the interface and to the data related to his projects. He can upload batches of tasks, view their progress and view and download results.

Additionally, we must consider the role of an administrator within the game, as it is the administrator who sets up the game for data collection. As such, the administrator will be in the context of the game a subactor of Player, as he will be registered in the game and able to perform the same tasks as a Player plus some additional administration features.

Game platform

Given that the game will have to communicate with the crowd platform, we must consider the game platform system as an actor in the crowd platform domain. The game will be able to request the creation of remote project users, request tasks to be executed and send in the results of executions.
3.2 Individual use cases

3.2.1 Crowd platform

Figure 7: Use cases for the crowd system
Use Case  |  Crowd register
--- | ---
Scope: | Crowd platform
Level: | Subfunction
Primary Actor: | Unknown User
Stakeholders and Interests: | Unknown User: obtain an account to access the crowd platform
Preconditions: | The unknown user does not have an existing account.
Postconditions: | The unknown user is registered in the system and becomes an administrator.
Main Success Scenario:
1. The user enters the crowd platform main page.
2. The user selects the register option.
3. The user enters his desired username, password and additional data requested in the fields provided.
4. The user presses the OK button.
5. The system registers the user and redirects him to the user’s main page.
Extensions:
3.a Invalid register data:
   1. System shows failure message
   2. User returns to step 3
Frequency of Occurrence: rare
**Use Case**

**Crowd login**

**Scope:**
Crowd platform

**Level:**
Subfunction

**Primary Actor:**
Administrator

**Stakeholders and Interests:**
- Administrator: access the crowd platform to control projects

**Preconditions:**
The administrator exists as a user in the system.

**Postconditions:**
The administrator is logged in to the crowd system.

**Main Success Scenario:**
1. The user accesses the web application home page.
2. The user navigates to the login page.
3. The user enters his username and password in the fields provided.
4. The user presses the login button.
5. The system authenticates the user and redirects him to the user's main page.

**Extensions:**
4.a Invalid login data:
1. System shows failure message
2. User returns to step 3

**Dependencies:**
Crowd register

**Frequency of Occurrence:**
frequent
Use Case Crowd logout

Scope: Crowd platform
Level: Subfunction
Primary Actor: Administrator

Stakeholders and Interests:
- Administrator: exit the crowd platform

Preconditions: The administrator has successfully logged in to the crowd platform.
Postconditions: The administrator is no longer logged in to the system

Main Success Scenario:
1. The user is within the crowd platform.
2. The user selects the option to logout.
3. The system logs out the user and redirects him to the main login page.

Dependencies: Crowd platform login

Frequency of Occurrence: frequent
Use Case | Crowd reset password
---|---
**Scope:** | Crowd platform
**Level:** | User-goal
**Primary Actor:** | Administrator

**Stakeholders and Interests:**
- Administrator: reset the forgotten password for his account

**Preconditions:**
- The administrator exists as a user in the system.
- The administrator is not logged in.

**Postconditions:**
The administrator’s password is changed.

**Main Success Scenario:**
1. The user accesses the web application home page.
2. The user navigates to the login page.
3. The user selects the forgot option and enters his username or email.
4. The system sends the user an email with a recovery link.
5. The user navigates to the recovery link.
6. The user enters a new password.
7. The system saves the new password and redirects the user to the login page.

**Extensions:**
3.a Invalid username or email:
   1. System shows failure message
   2. User returns to step 3

6.a Invalid password:
   1. System shows failure message
   2. User returns to step 6

**Dependencies:**
Crowd register, Crowd login

**Frequency of Occurrence:**
rare
**Use Case**

**Upload batch**

**Scope:** Crowd platform

**Level:** User-goal

**Primary Actor:** Administrator

**Stakeholders and Interests:**
- Administrator: provide the system with a batch of tasks to be distributed to players.

**Preconditions:** The administrator has successfully logged in to the crowd platform.

**Postconditions:** The provided batch of tasks is added to the system.

**Main Success Scenario:**
1. The user enters the crowd platform main page.
2. The user selects one of his projects.
3. The user enters the create batch page.
4. The user provides a file with the data to be entered, correctly formatted.
5. The system processes the file and adds the data as tasks within a new batch.
6. The system informs the user about the correct outcome of the task.

**Extensions:**
5.a Invalid data:
   1. The system shows failure message
   2. The user returns to step 4

**Dependencies:** Crowd login

**Frequency of Occurrence:** rare
<table>
<thead>
<tr>
<th><strong>Use Case</strong></th>
<th><strong>View batch progress</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope:</strong></td>
<td>Crowd platform</td>
</tr>
<tr>
<td><strong>Level:</strong></td>
<td>User-goal</td>
</tr>
<tr>
<td><strong>Primary Actor:</strong></td>
<td>Administrator</td>
</tr>
<tr>
<td><strong>Stakeholders and Interests:</strong></td>
<td>• Administrator: view the progress of a batch he has introduced in the system</td>
</tr>
</tbody>
</table>
| **Preconditions:** | • The administrator has successfully logged in to the Control and Administration panel.  
  • The administrator has previously entered a batch into the system.  |
| **Postconditions:** |                           |
| **Main Success Scenario:** |                         |
| 1. The user enters the crowd platform main page. | |
| 2. The user selects one of his projects. | |
| 3. The user selects one of the batches of tasks in the selected project. | |
| 4. The system shows the user information on the progress of that batch. | |
| **Dependencies:** | Crowd login             |
| **Frequency of Occurrence:** | frequent                |
Use Case: Start/stop batch

Scope: Crowd platform

Level: User-goal

Primary Actor: Administrator

Stakeholders and Interests:
- Administrator: start or stop providing tasks in a batch to the users.

Preconditions:
- The administrator has successfully logged in to the crowd system.
- The administrator has previously entered a batch of tasks into the system.

Postconditions:

Main Success Scenario:
1. The user enters the crowd platform main page.
2. The user selects one of his projects.
3. The user selects one of the batches of tasks in the project.
4. The system shows the user information on the progress of that batch.
5. The user selects the option to start/stop that batch.
6. The system informs the user that the action has been completed.

Dependencies: Crowd login

Frequency of Occurrence: rare
Use Case | View data graphs
--- | ---
**Scope:** | Crowd platform
**Level:** | User-goal
**Primary Actor:** | Administrator

**Stakeholders and Interests:**
- Administrator: view the results of executions of tasks he has introduced in the system as batches in the form of graphs.

**Preconditions:**
- The administrator has successfully logged in to the crowd platform.
- The administrator has previously entered one or more batches into the system.

**Postconditions:**

**Main Success Scenario:**
1. The user enters the crowd platform main page.
2. The user selects one of his projects.
3. The user selects the option to view graphs.
4. The system shows the user all the data collected with task executions as graphs, including a graph for daily executions and one graph for each custom defined output field in the project.

**Dependencies:** | Crowd login
**Frequency of Occurrence:** | rare
### Use Case View batch data graphs

**Scope:** Crowd platform  

**Level:** User-goal  

**Primary Actor:** Administrator  

**Stakeholders and Interests:**  
- Administrator: view the results of executions of tasks he has introduced in the system for a specific batch in the form of graphs.

**Preconditions:**  
- The administrator has successfully logged in to the crowd platform.  
- The administrator has previously entered a batch into the system.

**Postconditions:**

**Main Success Scenario:**
1. The user enters the crowd platform main page.
2. The user selects one of his projects.
3. The user selects one of the batches in the project.
4. The user selects the option to view graphs.
5. The system shows the user all the data collected with task executions for that batch as graphs, including a graph for daily executions and one graph for each custom defined output field in the project.

**Dependencies:** Crowd login  

**Frequency of Occurrence:** rare
<table>
<thead>
<tr>
<th><strong>Use Case</strong></th>
<th><strong>View batch data</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope:</strong></td>
<td>Crowd platform</td>
</tr>
<tr>
<td><strong>Level:</strong></td>
<td>User-goal</td>
</tr>
<tr>
<td><strong>Primary Actor:</strong></td>
<td>Administrator</td>
</tr>
<tr>
<td><strong>Stakeholders and Interests:</strong></td>
<td>• Administrator: view the results of executions of tasks he has introduced in the system as batches.</td>
</tr>
</tbody>
</table>
| **Preconditions:** | • The administrator has successfully logged in to the crowd platform.  
  • The administrator has previously entered a batch into the system. |
| **Postconditions:** |  |
| **Main Success Scenario:** | 1. The user enters the crowd platform main page.  
  2. The user selects one of his projects.  
  3. The user selects one of the batches of tasks in the project.  
  4. The system shows the user information on the progress of that batch.  
  5. The user selects the option to view results online.  
  6. The system shows the user all the data collected with task executions and allows him to manipulate it online. |
| **Dependencies:** | Crowd login |
| **Frequency of Occurrence:** | frequent |
## Use Case

### Download batch data

**Scope:** Crowd platform  
**Level:** User-goal  
**Primary Actor:** Administrator

**Stakeholders and Interests:**  
- Administrator: download the results of the executions of tasks he has introduced in the system as batches.

**Preconditions:**  
- The administrator has successfully logged in to the crowd system.  
- The administrator has previously entered a batch of tasks into the system.

**Postconditions:**

### Main Success Scenario:

1. The user enters the crowd platform main page.  
2. The user selects one of his projects.  
3. The user selects one of the batches of tasks in the project.  
4. The system shows the user information on the progress of that batch.  
5. The user selects the option to download results.  
6. The system provides the user with a file containing all task execution data for download.

**Dependencies:** Crowd login  
**Frequency of Occurrence:** frequent
Use Case Delete batch

Scope: Crowd platform
Level: User-goal
Primary Actor: Administrator

Stakeholders and Interests:
• Administrator: delete a batch of tasks.

Preconditions:
• The administrator has successfully logged in to the crowd platform.
• The administrator has previously entered a batch into the system.

Postconditions: The selected batch no longer exists in the system.

Main Success Scenario:
1. The user enters the crowd platform main page.
2. The user selects one of his projects.
3. The user selects one of the batches of tasks in the project.
4. The system shows the user information on the progress of that batch.
5. The user selects the option to delete the batch.
6. The system deletes the batch from the system and informs the user as well as redirects him to the project page.

Dependencies: Crowd login

Frequency of Occurrence: rare
**Use Case**  | **Create platform user**
--- | ---
**Scope:** | Crowd platform
**Level:** | subfunction
**Primary Actor:** | Game platform
**Stakeholders and Interests:** | • Game platform: create an instance of ProjectUser in the system related to a remote Player
**Preconditions:** | The Game platform has the correct credentials to access the Crowd platform remotely
**Postconditions:** | A new ProjectUser is created with the data given
**Main Success Scenario:**
1. The game platform receives a new Player registration.
2. The game platform communicates this new player’s data to the crowd platform
3. The system records the data received and sends back an id to identify the created user in the future
**Dependencies:** | Game register
**Frequency of Occurrence:** | rare
**Use Case**

**Obtain task**

**Scope:** Crowd platform

**Level:** subfunction

**Primary Actor:** Game platform

**Stakeholders and Interests:**
- Game platform: obtain a task to be executed remotely

**Preconditions:**
The Game platform has the correct credentials to access the Crowd platform remotely

**Postconditions:**

**Main Success Scenario:**
1. The game platform requires a task to be executed
2. The game platform requests a task from the crowd platform
3. The system selects a task for the given project and returns it

**Dependencies:** Solve exercise

**Frequency of Occurrence:** frequent
<table>
<thead>
<tr>
<th><strong>Use Case</strong></th>
<th><strong>Save execution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope:</strong></td>
<td>Crowd platform</td>
</tr>
<tr>
<td><strong>Level:</strong></td>
<td>subfunction</td>
</tr>
<tr>
<td><strong>Primary Actor:</strong></td>
<td>Game platform</td>
</tr>
<tr>
<td><strong>Stakeholders and Interests:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Game platform: save an execution recorded from a task</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
<td>The Game platform has the correct credentials to access the Crowd platform remotely</td>
</tr>
<tr>
<td><strong>Postconditions:</strong></td>
<td>The execution data is stored in the system</td>
</tr>
<tr>
<td><strong>Main Success Scenario:</strong></td>
<td></td>
</tr>
<tr>
<td>1. The game platform collects execution data for a task</td>
<td></td>
</tr>
<tr>
<td>2. The game platform sends the execution data to the crowd platform</td>
<td></td>
</tr>
<tr>
<td>3. The system collects the data and stores it</td>
<td></td>
</tr>
<tr>
<td><strong>Dependencies:</strong></td>
<td>Solve exercise</td>
</tr>
<tr>
<td><strong>Frequency of Occurrence:</strong></td>
<td>frequent</td>
</tr>
</tbody>
</table>
3.2.2 Game

Figure 8: Use cases for the game system
**Use Case**  | **Game register**  
--- | ---  
**Scope:**  | Game  
**Level:**  | Subfunction  
**Primary Actor:**  | Unknown User  
**Stakeholders and Interests:**  | • Unknown User: obtain an account to access the game  
**Preconditions:**  | The unknown user does not have an existing account.  
**Postconditions:**  | The unknown user is registered in the system and becomes a Player.  
**Main Success Scenario:**  
1. The user accesses the web application home page.  
2. The user selects the register button.  
3. The user enters his desired username, password and any other required personal data in the fields provided.  
4. The user presses the Register button.  
5. The system registers the user and redirects him to the task execution main page.  
**Extensions:**  
3.a Invalid register data:  
1. System shows failure message  
2. User returns to step 3  
**Frequency of Occurrence:**  rare
Use Case | Game login
--- | ---
Scope: | Game
Level: | Subfunction
Primary Actor: | Player
Stakeholders and Interests:
• Player: access the game
Preconditions: | none
Postconditions: | The Player is logged in to the system.
Main Success Scenario:
1. The user accesses the web application home page.
2. The user navigates to the login page.
3. The user enters his username and password in the fields provided.
4. The user presses the login button.
5. The system authenticates the user and redirects him to the game main page.
Extensions:
4.a Invalid login data:
1. System shows failure message
2. User returns to step 3
Dependencies: | Game register
Frequency of Occurrence: | frequent
<table>
<thead>
<tr>
<th><strong>Use Case</strong></th>
<th><strong>Game logout</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope:</strong></td>
<td>Game</td>
</tr>
<tr>
<td><strong>Level:</strong></td>
<td>Subfunction</td>
</tr>
<tr>
<td><strong>Primary Actor:</strong></td>
<td>Player</td>
</tr>
<tr>
<td><strong>Stakeholders and Interests:</strong></td>
<td>• Player: exit the game</td>
</tr>
<tr>
<td><strong>Preconditions:</strong></td>
<td>The Player has successfully logged in to the game.</td>
</tr>
<tr>
<td><strong>Postconditions:</strong></td>
<td>The Player is no longer logged in to the system</td>
</tr>
</tbody>
</table>

**Main Success Scenario:**

1. The user is within the game.
2. The user selects the option to logout.
3. The system logs out the user and redirects him to the main page.

**Dependencies:**

Game login

**Frequency of Occurrence:**

frequent
**Use Case**  
**Game reset password**

**Scope:** Game  
**Level:** User-goal  
**Primary Actor:** Player

**Stakeholders and Interests:**  
- Player: reset the forgotten password for his account

**Preconditions:**  
- The player exists as a user in the system.  
- The player is not logged in.

**Postconditions:**  
The player’s password is changed.

**Main Success Scenario:**

1. The user accesses the web application home page.
2. The user navigates to the login page.
3. The user selects the forgot option and enters his username or email.
4. The system sends the user an email with a recovery link.
5. The user navigates to the recovery link.
6. The user enters a new password.
7. The system saves the new password and redirects the user to the login page.

**Extensions:**

3.a Invalid username or email:
   1. System shows failure message
   2. User returns to step 3

6.a Invalid password:
   1. System shows failure message
   2. User returns to step 6

**Dependencies:**  
Game register, Game login

**Frequency of Occurrence:** rare
Use Case Solve exercise

Scope: Game

Level: User-goal

Primary Actor: Player

Stakeholders and Interests: • Player: complete an exercise execution

Preconditions: The Player has successfully logged in to the game.

Postconditions: A new task execution is created and saved with the produced output.

Main Success Scenario:
1. The user enters game main page
2. The user selects the play option
3. The System obtains and displays a linguistic exercise to the user
4. The user completes the exercise
5. The System retrieves and stores execution results

Dependencies: Game login

Frequency of Occurrence: frequent
Use Case: View personal data

Scope: Game

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:
- Player: view his personal data stored in the system

Preconditions: The Player has successfully logged in to the game.

Postconditions:

Main Success Scenario:
1. The user enters game main page
2. The user selects the option to view his personal data
3. The System displays the personal data it contains for the user

Dependencies: Game login

Frequency of Occurrence: medium
Use Case Delete user

Scope: Game

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:
- Player: delete his user and personal data

Preconditions: The Player has successfully logged in to the game.

Postconditions: The Player does no longer exist on the system

Main Success Scenario:
1. The user is within the game.
2. The user selects the option to view his personal data
3. The System displays the personal data it contains for the user
4. The user selects the option to delete his user from the system
5. The System deletes the user and his data, logs him out and redirects him to the main page

Dependencies: Game login

Frequency of Occurrence: rare
Use Case Enter platform security data

**Scope:** Game

**Level:** User-goal

**Primary Actor:** Administrator

**Stakeholders and Interests:**
- Administrator: modify the platform data so the game can connect successfully to the crowd platform

**Preconditions:** The Administrator has successfully logged in to the game.

**Postconditions:** The PlatformData is modified

**Main Success Scenario:**
1. The user is within the game.
2. The user selects the option to view his personal data
3. The user selects the option to view the administration panel
4. The System displays the existing platform data in a form
5. The user makes the changes he desires and confirms
6. The System modifies the existing platform data and takes the user back to the administration panel

**Dependencies:** Game login

**Frequency of Occurrence:** rare
3.3 Conceptual model

This project contains two domains to analyze, one related to the crowdsourcing platform and one related to the game. We hereby present two related conceptual models, one for each domain.

3.3.1 Crowd platform

![Diagram of conceptual model for the crowd domain](image)

**Figure 9:** Conceptual model for the crowd domain.

**Constraints**

- The Project User responsible for an execution must belong to the same project the task resolved belongs to.
- Each Project id must be unique.
- Each other id must be unique within it's parent's context.
- Fields within a Project must have a unique name.
- Administrator usernames must be unique within the entire system.
Project
A unit of study for the platform. It represents one context for a crowdsourcing project, and contains all the information related to it. In our case, we will have one sole instance to represent the Game project.

- **id**: An identifier for the project.
- **name**: A name given by the administrator to identify the project.
- **creationDate**: The date in which the project was created, for future reference.

Batch
A set of tasks. It groups related tasks into a unit, allowing the administrator to manage the whole set of tasks more effectively. The administrator should group tasks that will be managed and analyzed together into a batch. A project contains several batches, who then contain the tasks to be completed.

- **id**: An identifier for the batch within the project.
- **executionsPerTask**: The number of executions the administrator desires for each individual task within the batch.
- **creationDate**: The date in which the batch was created, for future reference.
- **percentageComplete**: A derived calculation of the progress of the batch, it contains the computation of the total number of executions for each task in the batch weighed over the total number that should be obtained in the end.
- **state**: The state in which the batch is at this moment. It can be any of the following: **RUNNING**, **PAUSED**, **COMPLETE**. A batch’s tasks will only be distributed to users if it is in a **RUNNING** state or, if lack of thereof, in a **COMPLETE** state. Executions for tasks in the batch will only be stored if, at the time of storage, the batch is **RUNNING**. A visual representation of state transitions can be found in figure 10.

![State diagram representing the states of a Batch and their transitions.](image-url)

**Figure 10**: State diagram representing the states of a Batch and their transitions.
Task
A unit of work to be completed by users. Tasks are the smallest unit of work introduced by the administrator in the system, and are the information served to the game on demand.

- **id**: An identifier for the task within the batch.
- **contents**: A map containing all the variable fields defined by the administrator as custom project input fields.

Execution
An execution is the result of a project user completing a task once. It contains the information gathered from the completion of the task, as well as the date in which is was completed. A project user may solve the same task several times, and will produce an execution for each of these times. Executions will be analyzed at a batch level, and therefore refer to the batch their task belongs to.

- **id**: An identifier for the execution within the batch.
- **date**: The date in which the execution was created, for future reference.
- **contents**: A map containing all the variable fields defined by the administrator as custom project output fields.

Project User
A user from the project who completes tasks. A project user belongs to a certain project, and is used to identify the author of executions. We do not require personal identification of users, and therefore refer to them with a simple id. The extra data collected by the remote project about the user is stored in the contents map, following the fields defined by the administrator.

- **id**: An identifier for the user within the project.
- **contents**: A map containing all the variable fields defined by the administrator as custom project user fields.

Password Reset Request
A request created by the user to reset his password. It contains a reference id and the date it was created to ensure its veracity.

- **requestId**: An identifier for the request.
- **dateCreated**: The date in which the request was created.
Administrator
A user of the crowd platform. In order to grant him restricted access, we collect a username and password, as well as an email to be able to retrieve forgotten passwords. Each administrator owns a set of projects.

- **username**: A unique username chosen by the user.
- **password**: A confidential code used to verify the user’s identity.
- **email**: A valid email address provided by the user.

Field
Fields are a unit of contents defined by the administrator on a project. These define three customized sets of additional information the administrator can collect for a project: input fields, related to the format of the tasks, output fields, related to the format of the data collected during executions, and user fields, related to the additional personal data collected from the users completing tasks.

- **name**: A key that identifies the field within the project. This is the key used to store information in the contents maps in `Task`, `Execution` and `ProjectUser`.
- **columnNames**: A list of names used for multivaluate fields, defining the names of the columns that should be read or written to form a multivaluate field from a plain format.
- **type**: The type of input the field accepts. In can be one of the following: `STRING`, `INTEGER`, `FLOAT`, `BOOLEAN`, `MULTIVALUATE STRING`. Each of these categories represents a primitive type except for `MULTIVALUATE STRING`, that represents a list of strings. This information is used to read and write the contents maps in `Task`, `Execution` and `ProjectUser`.
3.3.2 Game

The game domain takes most of its conceptual model from the existing product *Dyseggxia*, as it adapts its structure to a web application. *Dyseggxia* defines several types of problems that are generated and behave in different ways, and we will have to incorporate this in our model.

![Conceptual model for the game domain.](image)

**Figure 11:** Conceptual model for the game domain.

**Constraints**

- Each Problem id must be unique.
- Player usernames must be unique within the entire system.
Player
A user of the game. Players are registered in the system, who stores a set of personal data about them. Players are related to Project Users in the crowd platform domain, and each player has a corresponding project user.

- **username**: A unique username chosen by the user.
- **password**: A confidential code used to verify the user’s identity.
- **score**: A counter that increases with problem completion to engage the user into playing.
- **age**: The age of the user, useful for later data analysis. This will be included in the crowd platforms’ platform user contents map.
- **isDyslexic**: A boolean stating if the user has been diagnosed with dyslexia. This data will also be useful for later data analysis. This will be included in the crowd platforms’ platform user contents map.
- **isSpanishSpeaker**: A boolean stating if the user is a native Spanish speaker. This data will also be useful for later data analysis. This will be included in the crowd platforms’ platform user contents map.
- **email**: A valid email address provided by the user.

Password Reset Request
A request created by the user to reset his password. It contains a reference id and the date it was created to ensure its veracity.

- **requestId**: An identifier for the request.
- **dateCreated**: The date in which the request was created.

Execution
An execution is the result of a player completing a problem once. It contains the information gathered from the completion of the problem, and is connected to the Execution concept in the crowd platform. A player may solve the same problem several times, and will produce an execution for each of these times.

- **timeSpent**: The time the player has taken to complete the problem successfully since it was first presented to him. This will be included in the crowd platforms’ execution contents map.
- **failedAttempts**: The number of wrong answers the user has given before finding the correct answer. This will be included in the crowd platforms’ execution contents map.
- **wrongAnswers**: A list with all the wrong answers constructed by the user during the execution. This will be included in the crowd platforms’ execution contents map.
Platform Data
The concept Platform Data englobes the necessary data to correctly communicate with the Crowd platform.

- **projectId**: The id of the project that represents this system in the Crowd platform.
- **securityCode**: A security code generated by the Crowd platform for each Project to improve security.

Problem
A unit to provide to the users for completion. Each problem is a linguistic exercise that must be solved, and is related to a task in the crowd platform.

- **id**: A local identifier for the problem.
- **word**: The correct answer to be achieved when completing the problem.
- **display**: The wrong word to be displayed at the beginning of the execution. This string is encoded to contain information on how the display should be split into units before presenting them to the user if the separation does not correspond to a unit for each character.
- **answers**: A list with all the answers to display to the users, including the correct answer if necessary.
- **displayText**: A list calculated from the display string, containing one string for each unit to be displayed.

Word Problem
A problem that targets exercises at a word level. It contains only one word, and transformations are made on it to produce the exercise.

Sentence Problem
A problem that targets exercises at a level higher than a words. It contains more than one word, and produces exercises with words.

Insertion
A problem that consists on inserting a missing letter into a word with a blank space. Users are presented with a set of units and a blank space, and a set of possible answers that fit in the blank space. The solution is formed by inserting the correct answer in the blank space.
**Figure 12:** Exercises displayed on Dyseggxia. From left to right and top to bottom: Hard Insertion, Omission, Substitution, Transposition, Derivation and Separation.

**Hard Insertion**
Similar to insertion, the user is presented with a word that is missing a letter, but there is no blank space. The user must select an answer from a set of possible answers and insert it between any two units or at the beginning or end of the word in order to generate the solution.

**Omission**
A problem that consists on removing letters from a wrong word to produce a correct word. The user is presented with a word that contains additional erroneous letters and must detect them and remove them.

**Substitution**
A problem that consists on exchanging a unit from the displayed word with one of the possible answers in order to produce the solution. The user is presented with a word split into units and a set of possible answers, and must detect the wrong unit in the word and replace it with one of the possible answers.

**Derivation**
A problem that consists on selecting the correct termination for a word from a set of possible answers. By appending the termination to the word presented, the user produces the correct solution.
Transposition

A problem that consists on reordering the units presented to form a correct word. The user is presented with a word split into unordered units, and must reorder them to form the correct word.

Separation

A problem that consists on separating words from each other. The user is presented with a set of units that contain more than one word and must select the correct place to cut in order to separate the words apart.

3.4 Behavior model

context register(username: String, password: String, email: String): Administrator

Pre: A user with the username username does not already exist
     The username is not empty
     The password is not empty
     The email is a valid email address

Post: Creates a user of type Administrator in the system with the given username, password and email
context login(username: String, password: String)

**Pre:**
- A user with the username `username` exists
- The username is not empty
- The password is not empty

**Post:**
If the username and password match, the user is logged in to the system

context logout()

**Pre:**
A user is logged in

**Post:**
The user is no longer logged in to the system
context forgotPassword(username: String)

Pre: A user with the username or email \textit{username} exists
The username is not empty

Post: A PasswordResetRequest is created for the user
An email is sent to the user's email address

context resetPassword(username: String, newPassword: String)

Pre: A user with the username \textit{username} exists
The username is not empty
The user with username \textit{username} has requested a password reset

Post: The password for the user with username \textit{username} is changed to \textit{newPassword}
context `listProjects(): List<Project>`

*Pre:* A user is logged in

*Post:* The result is the list of projects created by the user

context `getProject(projectId: String): Project`

*Pre:* A user is logged in
A project identified by `projectId` exists
The logged in user owns the Project identified by `projectId`

*Post:* The result is the project identified by `projectId`

context `createBatch(project: Project, batchName: String, numExecutionsPerTask: Integer, file: File)`

*Pre:* The logged in user owns the Project `project`
The project `project` exists in the system
A batch with `batchName` does not already exist in the project
`batchName` is not empty
`numExecutionsPerTask` is greater than 0
`file` is a non-empty file containing task definitions

*Post:* A Batch is created with the `batchName`, `numExecutionsPerTask` and tasks defined in `file` and added to the Project `project`
context getBatch(project: Project, batchId: String): Batch

Pre: A user is logged in
The logged in user owns the Project project
The Project project exists within the system
A batch identified by batchId exists within the project

Post: The result is the batch identified by batchId within project
context startBatch(batch: Batch)

Pre: A user is logged in
The logged in user owns the Project to which batch belongs
The Batch batch exists within the system
The Batch batch is in a 'PAUSED' state

Post: The Batch batch is in a 'RUNNING' state

d context pauseBatch(batch: Batch)

Pre: A user is logged in
The logged in user owns the Project to which batch belongs
The Batch batch exists within the system
The Batch batch is in a 'RUNNING' state

Post: The Batch batch is in a 'PAUSED' state
context viewGraphs(project: Project)

Pre: A user is logged in
The Project project exists within the system
The logged in user owns the Project project

Post: The user is shown graphs that illustrate the execution data contained in project
context viewBatchGraphs(batch: Batch)

Pre: A user is logged in
The Batch batch exists within the system
The logged in user owns the Project to which batch belongs

Post: The user is shown graphs that illustrate the execution data contained in batch

d View batch data

<table>
<thead>
<tr>
<th>: Administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>projects = listProjects()</td>
</tr>
<tr>
<td>project = getProject(projectId)</td>
</tr>
<tr>
<td>batch = getBatch(project, batchId)</td>
</tr>
<tr>
<td>viewBatchData(batch)</td>
</tr>
</tbody>
</table>

context viewBatchData(batch: Batch)

Pre: A user is logged in
The Batch batch exists within the system
The logged in user owns the Project to which batch belongs

Post: The user is shown the execution data contained in batch
context `downloadBatchData(batch: Batch): File`

**Pre:**
A user is logged in
The Batch `batch` exists within the system
The logged in user owns the Project to which `batch` belongs

**Post:**
The result is a file containing all the executions for tasks in `batch`
context deleteBatch(batch: Batch)

Pre:  A user is logged in
      The Batch batch exists within the system
      The logged in user owns the Project to which batch belongs

Post: The Batch batch is deleted from the system
context createUser(projectId: String, user: ProjectUser): Integer

Pre: There is a Project identified by projectId in the system

Post: The ProjectUser user is added to the Project identified by projectId
The result is the id given to the user within the project

context getTask(projectId: String): Task

Pre: There is a Project identified by projectId in the system
The Project identified by projectId has at least one batch in 'RUNNING' state

Post: The result is a task belonging to a running batch from the Project identified by projectId
context saveExecution(projectId: String, execution: Execution): Integer

Pre: There is a Project identified by projectId in the system

Post: The Execution execution is added to the Project identified by projectId, within the Batch it belongs to

context register(username: String, password: String, email: String, isDyslexic: Boolean, isSpanishSpeaker: Boolean): Player

Pre: A user with the username username does not already exist
The username is not empty
The password is not empty
The email is a valid email address

Post: Creates a user of type Player in the system with the given username, password, email, isDyslexic and isSpanishSpeaker values
context login(username: String, password: String)

Pre: A user with the username $username$ exists
The username is not empty
The password is not empty

Post: If the username and password match, the user is logged in to the system

context logout()

Pre: A user is logged in

Post: The user is no longer logged in to the system
context forgotPassword(username: String)

Pre:  A user with the username or email *username* exists
      The username is not empty

Post: A PasswordResetRequest is created for the user
      An email is sent to the user’s email address

context resetPassword(username: String, newPassword: String)

Pre:  A user with the username *username* exists
      The username is not empty
      The user with username *username* has requested a password reset

Post: The password for the user with username *username* is changed to *newPassword*
**context** getProblem(): Problem

_Pre:_ A user is logged in

_Post:_ The result is a Problem from the system

**context** saveExecution(execution: Execution)

_Pre:_ A user is logged in
The Execution execution is not empty

_Post:_ The Execution execution is stored in the system related to the currently logged in user
**context**  
\texttt{showUserData()}: Player

\textit{Pre:} A user is logged in

\textit{Post:} The result is the logged in user's Player instance, with all its data

\begin{center}
\includegraphics[width=0.5\textwidth]{context-showUserData.png}
\end{center}

**context**  
\texttt{deleteUser(user: Player)}

\textit{Pre:} The Player \textit{user} exists in the system

\textit{Post:} The Player \textit{user} no longer exists in the system

\begin{center}
\includegraphics[width=0.5\textwidth]{context-deleteUser.png}
\end{center}

**context**  
\texttt{saveSecurityData(data: PlatformData)}

\textit{Pre:} A user with Administrator role is logged in

\textit{Post:} The PlatformData \textit{data} is stored in the system

\begin{center}
\includegraphics[width=0.5\textwidth]{context-saveSecurityData.png}
\end{center}
4 Design Overview

4.1 Architecture

The overall project architecture for the two developed systems has been highly influenced by the technological choices made, as the framework chosen for development has defined the overall architectural pattern to be used.

Both developed systems follow a layered architecture, defined by a Model View Controller base architectural pattern. The chosen framework Spring easily adapts to an MVC pattern, taking care of view handling and request forwarding [11].

Following the MVC structure, we have designed our systems to be split into several components:

- **Controllers** handle requests and manipulate the model
- **View templates** define how the model will be displayed to the user
- **Model** represents all data in the system
- **Services** allow controllers to interact with the database and external resources

Additionally, the Service layer in both systems is designed to follow a Domain Model pattern, simply storing the changes done to instances in the system.

4.1.1 Spring Framework

Spring is a popular application framework for Java. It is made out of several modules built on top of a main core container.

![Spring framework modules](Image taken from [12])

**Figure 13:** Spring framework modules. Image taken from [12]
Spring Core is the base module and takes care of Bean creation using the Inversion of Control pattern. This allows us to declare dependencies between beans and let the framework manage the binding between them. Spring configuration uses XML to define properties and beans, and therefore abstracts the actual code from deployment details like database access. Dependency injection allows us to abstract the behavior of a component from its dependencies, and is specially useful when the components must be tested.

The Data Access module abstracts the developer from the database, providing a JDBC module as well as support for mapping APIs like Hibernate. Additionally, it takes care of transactions.

The Spring Web MVC framework is an extension to Spring which adds a Dispatcher Servlet, an element that dispatches requests to controllers and resolves views. Using annotation, the developer can easily mark the role for each class, including controllers, services or validators. Each operation within a controller is annotated with the corresponding path, and the Dispatcher Servlet takes care of dispatching requests for that path to that controller operation. The framework also allows the developer to define return types by annotation, which can then be transformed to another format for delivery. For instance, when a request for a Task arrives to the crowd platform, it can simply return a Task instance and let the framework convert it to JSON and send it back to the request origin.

Views are defined through JSP and therefore allow for dynamic view building. Spring adds libraries to the JSP basic tag libraries and allows the developer to easily bind form fields to model attributes, after which the framework will take care of the binding and supply a filled-in instance of the desired class to the controller operation receiving the form contents.

![Figure 14: Spring framework front controller pattern. Image taken from [12]](image)

As can be seen, the Spring Dispatcher Servlet implements a Front Controller pattern that accepts all incoming requests and forwards them to the appropriate custom designed controller. If the request is for a view, the Front Controller obtains from the Controller the view name and model contents to display, and builds the view from View templates in order to finally return the view to the requester.

Spring differentiates roles between components in a software project. The ones we are interested in are controllers, which receive requests from the dispatcher servlet, services, which process data and interact with other components such as the database or external applications and configurations, which allow us to configure certain aspects from code instead of XML.

For application to application communication, we can take advantage of Spring REST support, a series of components that abstract the developer from common needs when using a RESTful API.
The base Spring framework also includes other components that will be of great use to us, such as utilities for Email sending or task management for asynchronous duties.

The Spring Security framework is an addition to Spring for access-control. With this extension, we can define access control properties in XML, and let the framework take care of the rest. By defining mappings from the database user objects to a pair of username and password, we can delegate log in and log out functionalities to Spring Security as well.

Finally, we have used the Spring Data project, which provides data access for non-relational databases in Spring. This project includes a module for MongoDB we have included in our crowd project, and which takes care of mapping from the database to Java objects.

For the main parts of development with Spring, we have followed [13] and [14].

4.2 Technology selection

In order to start implementing the project, we have had to make several selections concerning technology.

4.2.1 Framework

The systems developed in this project are being built in Java, using the Spring framework [15]. This selection was made by both the author and the supervisor of this project, as it involves a language we are both familiar with and allowed us to focus our efforts on the platforms rather than on overcoming the learning curve of a new language. Spring is a well known framework for Java that supports web application development, and the author was introduced to its basics during a course taught by the supervisor.

Given that the project is developed in Java, the chosen unit testing framework is JUnit, a popular unit testing framework for Java. We have also included Mockito [16], a mocking framework which works well with JUnit and allows us to mock dependencies in each class to isolate during testing. Test coverage is controlled using eCobertura [17], a plugin for Eclipse that analyzes the lines, blocks and branches covered by unit tests.

4.2.2 Storage

Data storage is an important part of this project. Given the ease of use when combined with the previously mentioned technologies, we incorporated Hibernate as an automatic persistence framework to deal with data storage and mapping. Additionally, Spring offers easy integration with Hibernate. As we will explain in section 5.10, one of the systems was migrated to MongoDB due to the large volumes of data it handles.

4.2.3 Interface interaction

The game to be developed is presented as a web application, and therefore we had the need to manipulate a game online. We decided to use Javascript and the HTML5 Canvas API in order to render and control the game execution. For Javascript unit testing, we used QUnit [18], a unit testing
framework specifically designed for Javascript. We decided to use this testing framework due to its popularity in front of others.

The use of HTML5 Canvas might be a bit of a risk, as older browsers do not support this feature. Nevertheless, we have considered this to be a good option, as we aim to create a modern game and are eager to learn about current technologies. Given that the developed game will be used in a controlled environment of users, we consider that this will not be a problem, and we include informative messages with instructions on how to overcome this problem in case we have affected users.

4.2.4 Graph representation

The crowd platform has a use case that implies representing graph data on a page. We searched extensively for graph rendering libraries and plugins, and were finally convinced by d3 [19], a javascript library that is easily customizable and can manage data easily as well as transform it if necessary.

4.2.5 Data visualization

Initially, we considered building a data visualization tool within our project. Nevertheless, given our short schedule and large amount of work, we decided to search for external options. We finally decided to use Google Fusion Tables [20], a platform that works on any Google user's Drive account and allows us to create tables and export data there for the user to see and manipulate.
5 Crowd Platform Design

5.1 Domain model

The original conceptual models have been adapted to produce the final domain model for each system. Here we present the final domain model for the crowd system.

![Diagram](image)

**Figure 15**: Crowd system domain model

The crowd platform domain model is practically identical to its conceptual model, with the derived attribute `percentageComplete` from the concept Batch materialized. A new class `PasswordResetRequest` appears, representing the request made by a user to reset his password and used in the Crowd reset password use case. We can also notice the `Administrator` concept has been renamed `PlatformUser`, to unify notation with `ProjectUser`, the entity related to users created within a project for execution data storage.

In order to improve the management of large quantities of data, executions are now grouped in a `BatchExecutionCollection`. We have one collection for each batch grouping all the executions of tasks in the batch instead of relating these executions directly to their task for efficiency, as will be described in the following section. Given that it is no longer possible to navigate from a task to its executions, we have included a materialized derived attribute in Task called `numExecutions` that contains the total count of executions stored for that task.
Constraints

- Each Project id must be unique.
- Each Batch Execution Collection id must be unique.
- Each other id must be unique within it's parent's context.
- Fields within a Project must have a unique name.
- Platform User usernames must be unique within the entire system.
- A batch must contain at least one task.
- All executions contained in a Batch Execution Collection must relate to a Task from the Batch to which the collection belongs to.
- The Project User responsible for an execution must belong to the same project the task resolved belongs to.

Apart from the basic domain model, the system contains an auxiliary model package which contains classes that are used to transform data for input and output purposes. These classes are related to classes from the model package, and either collect input data before transforming it to a model entity or transform the entity's data to transfer it externally.

Figure 16: Crowd system domain model auxiliary classes
Registration

A registration encapsulates the data introduced by the user during his register. This class is used as input, and its data is then transferred to a Platform User.

- **username**: A unique username chosen by the user.
- **password**: A confidential code used to verify the user’s identity.
- **confirmPassword**: A copy of the password provided by the user. This is used to check that the user committed no errors when introducing his password.
- **email**: A valid email address provided by the user.

PasswordResetData

A PasswordResetData encapsulates the data provided by the user during password reset.

- **requestId**: The requestId for the referred PasswordResetRequest. This information is included in the link mailed to the user and automatically introduced to the generated PasswordResetData.
- **password**: A confidential code used to verify the user’s identity.
- **confirmPassword**: A copy of the password provided by the user. This is used to check that the user committed no errors when introducing his password.

ProjectUserInfo

A ProjectUserInfo encapsulates the data received from the remote application about one of its users.

- **contents**: A map containing the custom defined fields for users.

ExecutionInfo

An ExecutionInfo contains the data of an execution received from the remote application.

- **batchId**: The id of the batch to which the task belongs.
- **taskId**: The task id for this execution’s task.
- **userId**: The id of the ProjectUser who executed this task.
- **contents**: A map containing the custom defined fields by the administrator for output.
TaskRequest
A TaskRequest contains the data received from the remote application when requesting tasks.

- **count**: The number of tasks to return for this request.

TaskInfo
A TaskInfo encapsulates the data to be sent to the remote application for one task, omitting unnecessary information such as relations.

- **batchId**: The id of the batch to which the task belongs to.
- **taskId**: The task id for this task.
- **contents**: A map containing all the custom fields defined by the project administrator for input.

MapReduceResult
A MapReduceResult is a special structure used for Map Reduce calculation results, and the only entity in the auxiliary package that is not devoted to input and output. It simply encapsulates a key and value of any type, to handle Map Reduce results.

- **id**: The key produced for one of the calculated items in a Map Reduce calculation.
- **value**: The value produced for one of the calculated items in a Map Reduce calculation.
5.2 Interaction Design

We now present the navigation between screens as presented in the crowd system for the described use cases.

Figure 17: Crowd system interaction design overview.
Crowd register

```
pkg crowd

<<screen>> Home page
<<screen>> Register page
<<form>> Register form
- username : String
- password : String
- email : String

Register error

Register successful

<<screen>> Projects page
```

<<screen>> Projects page

sd Crowd register

: Unknown user

navigate()

show register()

/ enter user information /

register()

navigate()
Crowd login

(pkg crowd)

Login error

Login successful

Projects page

Project

Home page

Login page

Login form
- username : String
- password : String

sd Crowd login

: Administrator

navigate()

show login()

/ enter login details /()

login()

navigate()
Crowd logout

(pkg crowd)

<<component>>
Logout button

<<screen>>
Home page

(sd Crowd logout)

: Administrator

log out()

navigate()
Crowd reset password

pkg crowd

<<screen>> Home page

<<screen>> Login page

<<form>> Reset form
  - requestId : long
  - username : String
  - password : String

<<screen>> Reset page

Reset successful

Reset error

sd Crowd reset password

Administrator

navigate()

show login()

navigate()

forgot password()

read email()

navigate()

/ enter reset data /

reset password()

navigate()
Upload batch

```
<screen> Projects page  <screen> Project page  <screen> Create batch page
```

```
Project  Batch
```

```
<screen> Batch page  <form> Create batch form
```

```
Create successful
Create error
```

```
sd Upload batch

: Administrator
```

```
navigate() select project() show create() / enter batch data / create() navigate()
```

```
: Project page  
: Project page  
: Create batch page  
: Batch page
```
View batch progress

pkg crowd

View batch progress

sd view batch progress

88
Start/stop batch

pkg crowd

<<screen>>
Projects page

<<screen>>
Project page

<<screen>>
Batch page

Start/stop

Project

Batch

sd Start/stop batch

: Administrator

navigate()

select project()

navigate()

select batch()

navigate()

start / stop()

navigate()
View data graphs

pkg crowd

<<screen>> Projects page
<<screen>> Project page
<<screen>> Graph page
- calculatedData : List<Map>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0..*</td>
<td></td>
</tr>
</tbody>
</table>

sd View data graphs

Administrator : Projects page
| navigate() |
| select project() |
| navigate() |
| show graphs() |
| navigate() |

navigate()
View batch data graphs
Download batch data

Diagram 1: Class diagram

pkg crowd

Projects page

Project page

Batch page

Project

Batch

0..*

1

0..*

Diagram 2: Sequence diagram

Administrator

navigate()

select project()

navigate()

select batch()

navigate()

download()
Delete batch

Note that there are three use cases that do not have an interaction design, which are the use cases in the crowd platform related to the Game platform as an actor. These use cases are fruit of a communication between both platforms, and therefore do not require any navigation between interfaces.
5.3 Remote API

The Crowd platform exposes a simple API that is then used by the Game to request and send data. This API has been designed as a REST API, and corresponds to the necessary operations for the use cases *Create project user*, *Obtain task* and *Save execution*.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Create project user</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI:</strong></td>
<td>/project/&lt;projectId&gt;/user</td>
</tr>
<tr>
<td><strong>Method:</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>Post contents:</strong></td>
<td></td>
</tr>
<tr>
<td>projectUid</td>
<td>Long</td>
</tr>
<tr>
<td>contents</td>
<td>Map&lt;String, Object&gt;</td>
</tr>
<tr>
<td><strong>Response:</strong></td>
<td>Integer: an Integer value with the id for the created user, 0 if an error occurred.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Obtain task</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>URI:</strong></td>
<td>/project/&lt;projectId&gt;/task</td>
</tr>
<tr>
<td><strong>Method:</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>Post contents:</strong></td>
<td></td>
</tr>
<tr>
<td>projectUid</td>
<td>Long</td>
</tr>
<tr>
<td>count</td>
<td>Integer</td>
</tr>
<tr>
<td><strong>Response:</strong></td>
<td>TaskInfo[]): the JSON encoding of a list of TaskInfo elements, each containing the following fields:</td>
</tr>
<tr>
<td><strong>Field name</strong></td>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>batchId</td>
<td>Integer</td>
</tr>
<tr>
<td>taskId</td>
<td>Integer</td>
</tr>
<tr>
<td>contents</td>
<td>Map&lt;String, Object&gt;</td>
</tr>
</tbody>
</table>
### Operation

**Save execution**

**URI:**

`/project/<projectId>/execution`

**Method:**

`POST`

<table>
<thead>
<tr>
<th>Field name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectUid</td>
<td>Long</td>
<td>The security code for the project</td>
</tr>
<tr>
<td>batchId</td>
<td>Integer</td>
<td>The batchId received with the obtained task</td>
</tr>
<tr>
<td>taskId</td>
<td>Integer</td>
<td>The taskId received with the obtained task</td>
</tr>
</tbody>
</table>

**Post contents:**

<table>
<thead>
<tr>
<th>contents</th>
<th>Map&lt;String, Object&gt;</th>
<th>The contents map with custom defined output fields for the project and their values for this execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>Integer</td>
<td>The platformId for the user who solved the task, as stated at user creation time.</td>
</tr>
</tbody>
</table>

**Response:**

The operation responds with HTTPSTATUS CREATED if it succeeds in saving the execution.

### 5.4 Interactions

Given the previous interaction design and the designed API, we can now discover the operations needed for the users to interact with the system, which will be split into several controllers. We have defined controllers at a conceptual level, joining requests that are related to one same conceptual entity in one controller. The defined controllers are just a part of the framework used, as the framework receives all requests from the users and forwards them to the appropriate controller we have defined.

**Crowd register**

The design for the Crowd register use case follows several steps, extending the original specification. First, the user navigates to the home page, provided by the HomeController. Then, he navigates to the register page, provided by the UserController, and registers himself through the `processRegistration` request operation. Finally, once registration is complete, the user is redirected to the projects page, provided by the ProjectController through its `listProjects` operation.
Crowd login

The design for the Crowd login use case is similar to its original specification. First, the user navigates to the home page, provided by the HomeController. Then, he navigates to the login page, provided by the UserController, and enters his credentials using the *login* operation. Finally, the user is redirected to the projects page, provided by the ProjectController.

We must note that the real logic for the *login* operation will not actually be provided by the UserController given that the framework takes care of this functionality.
Crowd logout

The design for the Crowd logout use case is very straightforward. From any other page in the system and while being logged in, the user may request a logout to the UserController, who will proceed with this petition and then redirect the user to the home page provided by the HomeController.

We must note that the real logic for the logout operation will not actually be provided by the User-Controller given that the framework takes care of this functionality and allows us to define a redirect page.
**Crowd reset password**

The design for the Crowd reset password use case implies several interaction steps. The user will navigate to the home page and then to the login page. Once there, he will select the `forgotPassword` option and will enter his username or email, after which he will receive an email from the system. By following the link in the email, the user will navigate to the password reset page provided by the UserController and will enter the data required. Once the user invokes the `resetPassword` operation, the system will change the password and redirect him to the login page.

![Diagram of Crowd reset password use case](image)

**Upload batch**

The upload batch use case is designed to interact mainly with the Batch Controller. First, the user must navigate to his project list and to the specific project he desires using the ProjectController. Then, he can invoke the `loadCreateBatch` operation from the BatchController, providing the id of the project to which the batch should be added. The BatchController will load a create batch page, where the user can introduce the data necessary as well as a file with the task definitions. Once sent to the system through the `createBatch` operation, the system will create the batch and redirect the user to the new batch's page.
**View batch progress**

The View batch progress use case is designed to contain the basic flow later included in other use cases such as Start/stop batch. The user will navigate to his project list and to a specific project using the ProjectController, and then load a specific batch's page using the `getBatch` operation in the BatchController.
Start/stop batch

The Start/stop batch use case is actually the union of two identical use cases, one to start and one to stop batches. Once the user has navigated to the batch page as stated in the View batch progress use case design, he may select the option to start the batch if it is paused or to stop it if it is running. Each of these actions will invoke its operation from the BatchController, who will change the batch status and return the user to the batch page.

View data graphs

The View data graphs use case is the first to use the GraphDataController, a controller designed specifically to respond to data requests for graphic plotting. When the user has navigated to a specific project's page as described earlier, he may select the show graphs option, which invokes its homonym operation in the GraphDataController.
View batch data graphs

The View batch data graphs use case design is very similar to the previous View data graphs use case. The user will this time have to navigate to the specific project page, select a batch to load the batch’s page and the select the show graphs option, which will invoke the `showBatchGraphs` operation in the GraphDataController.
**View batch data**

In the View batch data use case, the user will navigate down to the batch page as described in the View batch progress use case, and then select the view data option. The BatchController will take in this request, providing the batch’s executions data in a visible and interactive manner.

**Download batch data**

The Download batch data use case is also designed to extend the View batch progress use case. Once in the batch page, the user may select the option to download data, in which case the BatchController will accept the request and prepare a file with all the data for the user.
Delete batch

The Delete batch use case is designed to extend the View batch progress use case as well. In this case, the user may select the delete batch option, under which the BatchController will delete the displayed batch and redirect the user to the project page.
Create project user

Create project user is the first use case that involves the Game platform as a user, and therefore we have designed a special controller named `RemoteServiceController` that will be responsible for remote calls from the other platform. As such, we will expose the operations available as a REST API. In this case, we will have an available operation called `saveUser` which the other platform will be able to invoke on the `RemoteCommunicationController`.

Obtain task

As in the previous use case, the Obtain task use case is designed to include a remote call from the game platform to the system through the Remote Service Controller, in this case using the `provideTasks` operation.

Save execution

Finally, the Save execution use case has also been designed as a remote call from the game platform involving the operation `saveExecution` from the Remote Service Controller.
5.5 Controllers

We have divided the possible requests into controllers based on the concepts they interact with, and have obtained the set of controllers pictured below. Once we have distinguished the necessary operations for interaction with our users, we can proceed to describe their design in more detail.

Figure 18: Crowd system controllers
Home Controller

Operation loadHome

Inputs:
Output:

Preconditions:
Postconditions: Provides the home page by informing the framework of the home page view.

User Controller

Operation loadRegister

Inputs:
Output:

Preconditions: The user is not logged in
Postconditions: Provides the register page by informing the framework of the register page view and providing it with an empty instance of the Registration class

Operation processRegistration

Inputs: registration: an instance of the Registration class with its attributes filled with the data provided by the user in the register page form
Output:

Preconditions: The user is not logged in
Postconditions:
- The system validates the format of the data provided, and returns the user to the register page if there are any errors
- The system creates a new user with the data provided and stores him in the system
- The system logs the new user in and redirects him to the projects page
Operation loadLogin

Inputs:

Output:

Preconditions: The user is not logged in

Postconditions: Provides the login page by informing the framework of the login page view

Operation login

Inputs:

- username: a String containing the user's username
- password: a String containing the user's password

Output:

Preconditions: The user is not logged in

Postconditions:

- The system checks the user's credentials, and if they are not correct returns him to the login page with an error message
- The system logs in the user and redirects him to the projects page

Operation logout

Inputs:

Output:

Preconditions: The user is logged in

Postconditions:

- The user is logged out of the system
- The system redirects the user to the home page

Operation forgotPassword

Inputs:

- username: a String containing the username or email of the user

Output:
Preconditions:  
- The user is not logged in  
- The user has an account in the system  

Postconditions:  
- If there is no user in the system with the given username or email, the user is returned to the login page with an error message  
- A new PasswordResetRequest is created for the user and stored in the system  
- An email is sent to the user’s email address with a link to the password reset page
### loadPasswordReset

**Inputs:**
requestId: a String containing the requestId sent by email to the user, encoded in the link provided

**Output:**

**Preconditions:** The user is not logged in

**Postconditions:**
- Provides the password reset page by informing the framework of the password reset page view
- Provides the framework with an empty instance of the PasswordResetData class
- Adds the requestId to the PasswordResetData instance

### resetPassword

**Inputs:**
data: an instance of PasswordResetData with the attributes filled in with the data provided by the user through the password reset page form

**Output:**

**Preconditions:** The user is not logged in

**Postconditions:**
- The system validates the format of the data provided and returns the user to the password reset page if there are errors
- The system changes the password and deletes the existing PasswordResetRequest for the user
- The system redirects the user to the login page
**Project Controller**

**Operation**  
**listProjects**

**Inputs:**

**Output:**

**Preconditions:**  The user is logged in

**Postconditions:**

- Provides the projects page by informing the framework of the projects page view
- Provides a list of project model instances to the view

**Operation**  
**getProject**

**Inputs:**  
projectId: a String with the projectId to be retrieved

**Output:**

**Preconditions:**  The user is logged in

**Postconditions:**

- Provides the project page by informing the framework of the project page view
- Checks the credentials of the logged in user for the specific project
- Provides the project model instance to the view
Batch Controller

**Operation**  loadCreateBatch

**Inputs:**  
- projectId: a String containing the projectId for the project to contain the new batch

**Output:**

**Preconditions:**  The user is logged in

**Postconditions:**  
- Provides the create batch page by informing the framework of the create batch page view
- Provides the framework with an empty instance of the Batch class
- Adds the projectId to the form as a hidden attribute

**Operation**  createBatch

**Inputs:**  
- projectId: a String containing the projectId for the project to contain the new batch
- batch: an instance of the Batch class with its attributes filled in with the data provided by the user
- file: a File containing the definitions of tasks for the batch

**Output:**

**Preconditions:**  The user is logged in
\textbf{Postconditions:}

- The system verifies the user's credentials to modify the project
- The system transforms the file contents into a set of tasks and assigns them to the batch
- The system creates a new \texttt{BatchExecutionCollection} for the batch
- The system adds the batch to the project
**Operation** getBatch

**Inputs:**
- projectId: a String with the projectId to be retrieved
- batchId: the Integer that identifies a batch within the project

**Output:**

**Preconditions:**
The user is logged in

**Postconditions:**
- Provides the batch page by informing the framework of the batch page view
- Checks the credentials of the logged in user for the specific project
- Provides the batch model instance to the view

**Operation** startBatch

**Inputs:**
- projectId: a String with the projectId to be retrieved
- batchId: the Integer that identifies a batch within the project

**Output:**

**Preconditions:**
The user is logged in
**Postconditions:**

- Checks the credentials of the logged in user for the specific project.
- The system changes the identified Batch's status to 'RUNNING'.
- The system returns the user to the batch page.
**Operation** pauseBatch

**Inputs:**
- projectId: a String with the projectId to be retrieved
- batchId: the Integer that identifies a batch within the project

**Output:**

**Preconditions:** The user is logged in

**Postconditions:**
- Checks the credentials of the logged in user for the specific project
- The system changes the identified Batch’s status to ‘PAUSED’.
- The system returns the user to the batch page
Operation viewBatchData

Inputs:
- projectId: a String with the projectId to be retrieved
- batchId: the Integer that identifies a batch within the project

Output:

Preconditions: The user is logged in

Postconditions:
- Checks the credentials of the logged in user for the specific project
- Provides the view data page by informing the framework of the redirect view provided by the data viewer
- Provides the view to be shown with all the batch’s execution data
Operation  downloadBatch

Inputs:
- projectId: a String with the projectId to be retrieved
- batchId: the Integer that identifies a batch within the project

Output:

Preconditions: The user is logged in

Postconditions:
- Checks the credentials of the logged in user for the specific project
- The system creates a file containing all the data for the batch’s executions
- The system provides the file to the user
**Operation** deleteBatch

**Inputs:**
- projectId: a String with the projectId to be retrieved
- batchId: the Integer that identifies a batch within the project

**Output:**

**Preconditions:** The user is logged in

**Postconditions:**
- Checks the credentials of the logged in user for the specific project
- The system deletes the batch identified by the project and batchId
- The system deletes the BatchExecutionCollection related to the batch
Graph Data Controller

Operation **showGraphs**

**Inputs:**
- projectId: a String with the projectId

**Output:**

**Preconditions:**
- The user is logged in

**Postconditions:**
- Checks the credentials of the logged in user for the specific project
- For each output field defined in the project, the system calculates the aggregated data necessary for the graphs
- The system provides the graphs page by informing the framework of the graphs page view
- The system provides all the calculated data to the view

Operation **showBatchGraphs**

**Inputs:**
- projectId: a String with the projectId to be retrieved
- batchId: the Integer that identifies a batch within the project

**Output:**
**Preconditions:** The user is logged in

**Postconditions:**
- Checks the credentials of the logged in user for the specific project
- For each output field defined in the project, the system calculates the aggregated data necessary for the graphs from the specified batch
- The system provides the batch graphs page by informing the framework of the batch graphs page view
- The system provides all the calculated data to the view
Remote Service Controller

Operation saveUser

Inputs:
- projectId: a String with the projectId
- user: an instance of ProjectUserInfo with the data for a new ProjectUser

Output: id: The id given to the new user in the project

Preconditions:

Postconditions:
- Checks the credentials of the request for the specific project
- The system creates a new ProjectUser with the data provided and assigns him to the project
- The system returns the id given to the new user within the project for future reference
Operation: provideTasks

Inputs:
- projectId: a String with the projectId
- request: an instance of TaskRequest with the number of tasks to retrieve

Output: tasks: an array containing instances of TaskInfo with the available information for each task retrieved

Preconditions:

Postconditions:
- Checks the credentials of the request for the specific project
- The system returns the a set of retrieved tasks converted to the class TaskInfo for output
**Operation**  
saveExecution

**Inputs:**
- projectId: a String with the projectId
- execution: an instance of ExecutionInfo with the data collected from an execution

**Output:**

**Preconditions:**

**Postconditions:**
- Checks the credentials of the request for the specific project
- The system creates a new Execution with the data provided and stores it in the BatchExecutionCollection for the batch specified
5.6 Views

We have seen now the elements that form the M and C of MVC, and we must take a minute to analyze the Views.

Given that we are using the Spring framework, views are automatically managed by the framework. Each controller which desires to respond to a request with a view returns a view template name and the elements necessary to compose the final view, and the framework pulls up the appropriate template, gives it the data and constructs the final screen shown to the user. We now present the relation of controllers, view templates and views.

For the Crowd platform, we have five controllers that manage views:

**Home Controller**

![Home Controller Diagram]

**User Controller**

![User Controller Diagram]
Graph Data Controller

Notice that we do not provide information on the Remote Service Controller. This controller does not manage any view requests, as it only receives remote requests for data management through the previously described API.
5.7 Services

Up until now we have seen the interactions between the user and the system as well as the operations needed by the controllers, and have detailed the behavior of non-trivial operations. We can now move our focus to the Services level, where the rest of the logic can be found.

Services are a conceptual part in Spring MVC, which describes them as components that relate to data processing or tools. In our case, the Services layer includes the communication with the database and with external sources, as well as processing model data.

In order to abstract the controller logic from the data storage implementation details, all services have been defined as interfaces, and we have then created specific classes that implement these services for a specific storage or processing implementation. We will omit the details of those operations which simply interact with the database, and will describe those that have additional behavior.

The operations described until now require the use of three database services, one for users, projects and batch execution collections, and one for processing the tasks to be returned on demand.

![Diagram of Crowd system services.](image)

**Figure 19:** Crowd system services.
Platform User Service

Operation getCurrentUser

Inputs:

Output: An instance of PlatformUser with the attributes filled with the data for the currently logged in user

 Preconditions:

 Postconditions:

Operation getUserByUsernameOrEmail

Inputs: username: a String containing the username or email of a user

Output: An instance of PlatformUser matching the provided input by username. If none is found, an instance matching the provided input by email. Otherwise, a null instance.

 Preconditions:

 Postconditions:

Operation saveUser

Inputs: user: an instance of PlatformUser

Output:

 Preconditions:

 Postconditions: The system stores the user. If an existing user is found with the same username, it is replaced. Otherwise, it is stored as a new user.

Operation getUserWithPasswordResetRequestId

Inputs: requestId: a Long containing a password reset request id

Output: An instance of PlatformUser matching the provided requestid. If none is found, a null instance.

 Preconditions:

 Postconditions:
Project Service

Operation getProjectsForUser

Inputs: user: an existing user in the system

Output: A list containing all the projects owned by the user.

Preconditions:

Postconditions:

Operation getProject

Inputs: projectId: a String containing the id for a project

Output: An instance of Project matching the specified id if found, null if none match.

Preconditions:

Postconditions:

Operation saveProject

Inputs: project: an instance of Project

Output:

Preconditions:

Postconditions: If a Project with the same id is already stored in the system, it is over- ridden. Otherwise, the project is stored as a new Project in the system.
BatchExecutionService

Operation: getBatchExecutionCollection

Inputs: collectionId: a String containing the id for a collection

Output: An instance of BatchExecutionCollection matching the specified id if found, null if none match.

Preconditions:

Postconditions:

Operation: saveBatchExecutionCollection

Inputs: collection: an instance of BatchExecutionCollection

Output:

Preconditions: If a BatchExecutionCollection with the same id is already stored in the system, it is overridden. Otherwise, the collection is stored as a new BatchExecutionCollection in the system.

Postconditions:

Operation: removeBatchExecutionCollection

Inputs: collection: an instance of BatchExecutionCollection

Output:

Preconditions:

Postconditions: If the instance of BatchExecutionCollection is stored in the system, it is removed.
TaskRetrievalStrategy

Operation retrieveTasksForExecution

Inputs:

• project: an instance of Project from the system

• count: the number of tasks to retrieve

Output: tasks: a list of tasks

Preconditions:

Postconditions:

This functionality has been included in an interface even though it is not strictly a service to allow the definition of several task retrieval strategies, following a Strategy pattern. We have for now defined one concrete implementation, titled Random Batch Less Executed Task Retrieval Strategy. As the name implies, this strategy takes one batch from the project at random from the batches that are in 'RUNNING' state. If there are none, it takes a 'COMPLETED' batch. Once the batch is selected, it shuffles the tasks and then orders them by number of executions collected, and then provides the number of requested tasks from the end of the list with less executions.

We believe the current implementation covers our needs, as it gives no preference to any batch as well as tries to ensure that all tasks are executed an equal number of times, providing them shuffled and disperse.
5.8 Utils

Finally, we must take into account the classes in the *util* package, which have appeared along the way.

![Diagram of utility classes](image)

**Figure 20:** Crowd system utilities

**File Reader**

**Operation**  
readFile

**Inputs:**  
file: A file containing task data to decode

**Output:**  
a list where each item is a task from the file, and is encoded as a map of attributes

**Preconditions:**

**Postconditions:**

We have created a concrete implementation of FileReader for CSV files, reading each csv line as a task and each column as an attribute for the map. This interface allows the further extension to support other types of data files, though for now we have considered csv to be enough.
File Writer

**Operation**: writeBatchData

**Inputs**: 
- project: an instance of Project containing the fields to be written
- batch: an instance of Batch containing the tasks to be written
- collection: an instance of BatchExecutionCollection with all the execution data to be written

**Output**: a string that encodes the contents to be written into a file for data download

**Postconditions**:

File Writer specifies a generic interface to encode output data into several formats, just like File Reader. In our case, we have implemented File Writer to produce a CSV formatted output, de-normalizing the data of tasks so that each row in the produced csv is an execution with all its data plus all the data about the task it has executed and the user that has produced the data.

Data Miner

**Operation**: aggregateByField

**Inputs**: 
- project: an instance of Project
- fieldName: the name of the field to be aggregated

**Output**: a map containing the aggregation data in the format key-value for the specified field from all executions contained in the project

**Postconditions**:

**Operation**: aggregateByField

**Inputs**: 
- project: an instance of Project
- batch: an instance of Batch
- fieldName: the name of the field to be aggregated

**Output**: a map containing the aggregation data in the format key-value for the specified field from all executions contained in the batch

**Postconditions**:
The Data Miner is once again a generic interface to allow several implementations. We first produced a version that simply computed the results by obtaining all the data from the database and computing, but it became clear that this method would soon be scaled out by the magnitude of the data gathered. Therefore, we decided to change the implementation to a Map Reduce which calculates the data on the database server. With this, we have simply had to define some simple functions which process the data and have left the rest of the work to the MongoDB framework. Details will be explained further in the thesis.

**Data Viewer**

**Operation** getDataURL

**Inputs:**
- project: an instance of Project
- batch: an instance of Batch
- collection: an instance of BatchExecutionCollection

**Output:** the link to a view capable of showing the data to the viewer and manipulating it

**Postconditions:**

Data Viewer is an interface that allows us to use several data visualization tools to show the data collected to the user. In our case, we have implemented an adapter that integrates our project with Google Fusion Tables. We export all the data in the same format as the File Writer, de-normalizing the data and sending it to a Google Fusion Table. Then, we provide the link to the created table so the user can visualize and manipulate it.

**Task Creator**

**Operation** createTasks

**Inputs:**
- batch: an instance of Batch
- fields: an list containing the input fields for the Project
- contents: a list where each item contains a Map with the data for one task in a key-value format

**Output:**

**Postconditions:**
- The system transforms the input contents data into Tasks, filling their attributes with the data provided.
- The system assigns these tasks to the batch provided.
Mail Sender

Operation sendPasswordResetEmail

Inputs:
- user: The PlatformUser to who the email is addressed
- request: The request for which the email is sent

Output:

Preconditions:

Postconditions: An email is sent to the user's email address containing a link for the password reset page with an embedded reference to the request.

Password Reset Data Validator

Operation validate

Inputs: data: An instance of PasswordResetData with the information introduced by the user

Output:

Preconditions:

Postconditions:
- The system checks that the data is correctly formatted.
- The system checks that the password and confirmation password introduced are the same.
**Registration Validator**

**Operation** validate

**Inputs:** registration: An instance of Registration with the information introduced by the user

**Output:**

**Preconditions:**

**Postconditions:**

- The system checks that the data is correctly formatted.
- The system checks that the username is not already taken by another user
- The system checks that the password and confirmation password introduced are the same.

### 5.9 Packages

With this, we have seen a complete overview of the classes included in the project's packages as well as their operations. For a final wrap-up, we present the package diagram, summarizing the packages that have appeared.

![Package Diagram](image_url)

**Figure 21:** Crowd system packages
5.10 Data model

Both systems developed in this project have taken advantage of automatic persistence, but from different points of view. While the game system follows a relational structure, this system has been adapted to a NOSQL solution.

Implementing a Hibernate solution is easy to develop and maintain, and is well integrated into the Spring ecosystem. Nevertheless, Hibernate has been proven to have its limits when dealing with large volumes of records [21]. This will not represent a problem for our game system, as the only table that can grow is the one related to GameUsers, and we believe it will be hard to reach a problematic volume of users in the near future. Nevertheless, the problems detected on Hibernate could be of great impact to our crowd system, as it will gather a large quantity of executions. Therefore, we explored other possible solutions including NOSQL systems.

Initially, this system was adapted to Hibernate through its annotations package [22], which allows the developer to annotate Java model classes and delegate the transformation to Hibernate, just like the Game system. Nevertheless, halfway through the project we realized that the amount of executions to be collected would soon be unmanageable from a relational storage solution. Once we discovered this solution would not be enough, we analyzed our structure to find the best option from the currently available Big Data solutions. The two strongest candidates were Hadoop + HBase [23], a Big Table based solution, and MongoDB [24], a document store solution.

The Crowd system has a particular feature that was difficult to adapt to a relational model: the entities Task, Execution and ProjectUser have custom defined fields for each Project. Given that we can not create a database schema for each project, we have encoded these custom defined fields in a Map structure within each entity. Hibernate is not capable at this moment to translate Map structures into a database, and therefore our first solution relied on converting the custom defined attributes Map into a JSON stream we could store as a simple String. Given this particularity, we found that both NOSQL solutions considered, being schema-less, allowed us to undo this conversion and keep the attributes as a Map, which is easier to manage at a logic level.

Hadoop and HBase suggest a key-value NOSQL solution to data storage and processing. In essence, a map with several levels of depth is stored for each row, grouping attributes into column families and keeping several timestamped versions of each value. This solution is very powerful to store large volumes of data, but is schema-less and requires some work on the data manipulation layer to convert the data into instances.

MongoDB is a document store solution, which stores data in an extended version of JSON called BSON. As such, it can store embedded objects, including relationships and collections. Given that we were already encoding our data in a JSON format, MongoDB seamed to fit our project perfectly. As most popular NOSQL solutions, MongoDB allows the user to fragment and replicate data across servers, as well as to process data using Map Reduce. Additionally, an extension of the regular MongoDB driver exists for Spring, under the project Spring Data [25], which allows the developer to work with annotations and takes care of automatic persistence, making the use of MongoDB almost transparent to the project’s logic.

In the end, we decided to use MongoDB, given the ease of use it presented when combined with Spring Data as well as the ease to incorporate it into our deployment environment. We also believe that MongoDB can support our model better including entity relations. Given that MongoDB allows us to store embedded objects, we took the original data model and had to decide on what entities we would keep separated and what entities we would denormalize and store as one document.
As we can see, we could easily store all information in one document using the Project as the root. Nevertheless, we must take into account that every time we retrieve a Project from the database we will be retrieving all the information as it will be stored in one document. Given that our problem was the large amount of Executions we will store for each Project, we decided to separate Executions from the rest. With this separation, each time we retrieve a Project we will also be retrieving its owner, its users, its fields, its batches and their tasks. We notice at this point that we will be storing duplicate information, as the creator may have other projects in the system. Therefore, we separated PlatformUsers to another collection, leaving a reference attribute in the Project to fetch the creator when necessary.

Finally, we must consider Executions. They could be stored individually, but this would incur in a large overhead when we must retrieve all Executions for a given batch, as happens in some of our use cases. Given that we never retrieve Executions at a task level, the most convenient solution relies on grouping Executions by Batch. It is at this point that the created BatchExecutionCollection entity makes sense to our users, as it represents the root entity from which we are going to embed and store Executions. By storing a reference to a BatchExecutionCollection on each Batch, we will be able to easily retrieve all Executions for that Batch. The relationship between Executions and their Task must also be broken, as we do not wish to store repeated Task data once in each Execution, and we therefore also keep a reference to the appropriate Task within each Execution.

With all the previous decisions, we can now model the schema for our MongoDB database. Since we are using an annotation-based solution in our project, we will have to reflect this schema at a logic level within our Model package and manually retrieve those entities which have been stored as
a reference when needed.

![Database Schema Diagram](image)

**Figure 22:** Crowd system database schema

Finally, we must consider additional optimizations. We realized we query in one use case for a user’s projects, and these are stored in different documents. We therefore added an index on the creator attribute of the Project collection, to ensure that collecting all Project documents owned by a user would be easy. We also query users by their email or by their password reset request in the Reset password use case, so we added indexes on the email and passwordResetRequest.id attributes of the PlatformUser collection.

### 5.10.1 Map Reduce

Once we had migrated our project to a NOSQL solution, we were able to explore new data processing techniques. Since some of our use cases process large amounts of data, we thought it would be interesting to introduce Map Reduce into our system.

Map Reduce is a programming model that allows us to manipulate large amounts of data in a parallel manner [26]. By specifying a Map and a Reduce function, we can solve some of our data processing
problems, specifically those related to the *Data Miner*. Map Reduce is a query-shipping technique, as the processing of the data occurs in the same place where the data is stored. This offers us an additional advantage, as it takes computational stress off our own system.

The Spring Data package for MongoDB allows us to use the Map Reduce technique easily, as we simply have to define the two functions in Javascript and let the framework do all the rest. We therefore must define the two operations that will take care of the *aggregateByField* operation within the *Data Miner*.

The Map function defined will take an instance of BatchExecutionCollection as input, and will emit a tuple with the value for the specified field and a counter set to one for each execution. The Reduce function will then simply have to sum all the counters received for each value.

```javascript
// Map function
function() {
    for (var i = 0; i < this.executions.length; ++i) {
        var val = this.executions[i].contents[fieldName];
        emit(val, 1);
    }
}

// Reduce function
function (reducekey, reducevalues) {
    return Array.sum(reducevalues);
}
```

Given that the Map Reduce framework can easily deal with large amounts of documents, we could consider grouping executions by Task as a better option, as we would have Executions better divided to execute the Map Reduce functions. Nevertheless, we decided to keep the Executions grouped by Batch as other use cases will access to Executions, and it is preferable to retrieve one document with all the information than several documents, one for each Task.
6 Game Platform Design

6.1 Domain model

The original conceptual models have been adapted to produce the final domain model for each system. Here we present the final domain model for the crowd system.

![Game system domain model]

The game domain is strongly simplified from its conceptual model. Given that the different types of problems only differ in their behavior and generation, there is no need to represent them as subclasses. We have simplified the Problem hierarchy into one class, `Problem`, with a new attribute `type` that indicates what problem subclass each problem belongs to. The problem hierarchy will reappear with view controllers when we work on the game interface logic, representing the behavior of each subclass.

The `Execution` class disappears, as this information will not be stored in this system. However, we will see that in the auxiliary structures we will have two classes representing what we know as an Execution, who will manage the execution data through our system from the game interface until it is sent to the crowd platform.

As in the crowd domain, a new class `PasswordResetRequest` appears, representing the request made by a user to reset his password and used in the Game reset password use case, and the Player concept is renamed to `Game User` to unify notation.

We must notice the Game User also gains a new attribute, `platformId`, which relates his to his counterpart class instance in the crowd platform system. Similarly, Problem gains two new attributes, `batchId`
and `taskId`, which relate the Problem to its counterpart Task in the crowd platform system.

Finally, we see a new concept, the `ProblemCollection`. This class will be used to group problems, and mainly separate into different collections the problems obtained as tasks from the platform and the problems stored internally and used in case the communication with the crowd platform is not reliable.

**Constraints**

- Each `ProblemCollection` id must be unique.
- Each `Problem` id must be unique.
- Game User usernames must be unique within the entire system.

Apart from the basic domain model, the system contains an auxiliary model package which contains classes that are used to transform data for input and output purposes. These classes are related to classes from the model package, and either collect input data before transforming it to a model entity or transform the entity’s data to transfer it externally.

![Figure 24: Game system domain model auxiliary classes](image)
Registration
A registration encapsulates the data introduced by the user during his register. This class is used as input, and its data is then transferred to a Game User.

- **username**: A unique username chosen by the user.
- **password**: A confidential code used to verify the user's identity.
- **confirmPassword**: A copy of the password provided by the user. This is used to check that the user committed no errors when introducing his password.
- **age**: The age of the user.
- **isDyslexic**: A boolean stating if the user has been diagnosed with dyslexia.
- **isSpanishSpeaker**: A boolean stating if the user is a native Spanish speaker.
- **email**: A valid email address provided by the user.

GameUserInfo
A GameUserInfo encapsulates the data to be sent to the crowd platform for ProjectUser creation. Given that the crowd platform encodes the custom data in a contents map, we transform the data at this point before shipping.

- **contents**: A map containing the attributes from GameUser to be sent: age, isDyslexic and isSpanishSpeaker.

PasswordResetData
A PasswordResetData encapsulates the data provided by the user during password reset. In order to verify the identity of the user resetting the password, we obtain the requestId and the username of the user, and cross-check their relation.

- **requestId**: The requestId for the referred PasswordResetRequest. This information is included in the link mailed to the user and automatically introduced to the generated PasswordResetData.
- **username**: The username for the Game User whose password is being changed.
- **password**: A confidential code used to verify the user's identity.
- **confirmPassword**: A copy of the password provided by the user. This is used to check that the user committed no errors when introducing his password.
TaskInput

A TaskInput encapsulates the data obtained from the crowd platform for each task retrieved. This data is then transformed to a Problem, reading the custom fields from the contents map.

- **batchId**: The id of the batch to which the task belongs to in the crowd platform system.
- **taskId**: The task id for this task in the crowd platform system.
- **contents**: A map containing all the custom fields defined by the project administrator and provided at batch creation time. These should include: `word`, `display`, `type` and `answers`. The Problem attribute `displayText` is calculated from the information in `display`.

TaskRequest

A TaskRequest contains the data sent to the crowd platform to request more tasks.

- **count**: The number of tasks to retrieve with this request.

ProblemOutput

A ProblemOutput contains the data sent to the game interface to represent a Problem. As an output structure, it contains the data from a Problem omitting the extra information not necessary for the interface.

- **batchId**: The id of the batch to which the problem belongs to in the crowd platform system.
- **taskId**: The task id for this problem in the crowd platform system.
- **type**: The type of Problem this information belongs to. It can be one of the following: `insertion1`, `insertion`, `omission`, `substitution`, `transposition`, `derivation`, `separation`.
- **word**: The correct answer to be achieved when completing the problem.
- **answers**: A list with all the answers to display to the users, including the correct answer if necessary.
- **displayText**: A list containing one string for each unit to be displayed.
ExecutionResults
An instance of ExecutionResults contains the data gathered by the game interface during a problem’s execution. This data, instead of being transformed to a local model entity, is directly transformed to the output format represented by an ExecutionInfo and shipped to the crowd platform system.

- **batchId**: The id of the batch to which the problem belongs to in the crowd platform system.
- **taskId**: The task id for this problem in the crowd platform system.
- **timeSpent**: The time the user has taken to complete the problem successfully since it was first presented to him.
- **failedAttempts**: The number of wrong answers the user has given before finding the correct answer.
- **wrongAnswers**: A list with all the wrong answers constructed by the user during the execution.

ExecutionInfo
An ExecutionInfo encapsulates the data of an execution in the format necessary to send it to the crowd platform system.

- **batchId**: The id of the batch to which the problem belongs to in the crowd platform system.
- **taskId**: The task id for this problem in the crowd platform system.
- **userId**: The platformId of the currently logged in user.
- **contents**: A map containing the custom defined fields for executions. These should be: timeSpent, failedAttempts and wrongAnswers.
6.2 Interaction Design

We now present the navigation between screens as presented in the game system for the described use cases.

Figure 25: Game system interaction design overview.
Game register
Game login

```
<<screen>>
Home page

<<screen>>
Login page

<<form>>
Login form
- username : String
- password : String

<<screen>>
Game page
- score : Integer

Problem
GameUser

<<screen>>
Home page

<<screen>>
Login page

<<screen>>
Game page
```

Player

navigate()

show login()

navigate()

/ enter login details /()

login()

navigate()
Game logout

```
-- pkg game

<<component>>
Logout button

<<screen>>
Home page

-- sd Game logout

: Player

logout()

navigate()

<<component>>
: Logout button

<<screen>>
: Home page
```
Game reset password

pkg game

<<screen>> Home page

<<screen>> Login page

<<form>> Reset form
- requestId : Long
- username : String
- password : String

Reset successful

<<screen>> Reset page

Reset error

sd Game reset password

Player

navigate()

show login()
navigate()

forgot password()

read email()

navigate()

/ enter reset data /

reset password()

navigate()
Solve exercise
View personal data

```
pkg game

<<screen>>
Game page
- score : integer

<<screen>>
User page

Problem

GameUser

1
```

```
sd View personal data

<<screen>>
Game page

<<screen>>
User page

Player

navigate()

show user page()

navigate()
```
Delete user

pkg game

Game page
- score : Integer

Problem

User page

GameUser

1

Home page

sd Delete user

Player

navigate()

show user page()

navigate()

delete user()

navigate()
Enter platform security data

![Diagram showing game page, user page, admin page, and platform data relationships]

---

**pkg** game

- **Game page**
  - score: Integer

- **User page**

- **Admin page**
  - projectID: String
  - securityCode: Long

**PlatformData**

---

**sd** Enter platform security data

![Sequence diagram showing player navigating through game page, user page, and admin page]

- **Player**
  - navigate
  - show user page
  - navigate
  - show admin page
  - navigate

/ enter platform data /

- save
  - navigate
6.3 Interactions

We proceed now to the design of the game system internally. Parting from the interactions of the actors with the system we have previously analyzed, we discover the need for controllers capable of receiving requests from the actors. As with the crowd system, the controllers are just a part of the framework, and requests to the system are directly handled by the framework and then redirected to the appropriate controller.

Game register

The design for the Game register use case follows several steps, extending the original specification. First, the user navigates to the home page, provided by the HomeController. Then, he navigates to the register page, provided by the UserController, and registers himself through the processRegistration request operation. Optionally, the user may want to review the site policy, which is also provided by the UserController. Finally, once registration is complete, the user is redirected to the game page, provided by the GameController.

Game login

The design for the Game login use case is similar to its original specification. First, the user navigates to the home page, provided by the HomeController. Then, he navigates to the login page, provided by the UserController, and enters his credentials using the login operation. Finally, the user is redirected to the game page, provided by the GameController.
We must note that the real logic for the login operation will not actually be provided by the UserController given that the framework takes care of this functionality.

**Game logout**

The design for the Game logout use case is very straightforward. From any other page in the system and while being logged in, the user may request a logout to the UserController, who will proceed with this petition and then redirect the user to the home page provided by the HomeController.

We must note that the real logic for the logout operation will not actually be provided by the UserController given that the framework takes care of this functionality and allows us to define a redirect page.
**Game reset password**

The design for the Game reset password use case is nearly identical to its equivalent use case in the crowd system. The user will navigate to the home page and then to the login page. Once there, he will select the `forgotPassword` option and will enter his username or email, after which he will receive an email from the system. By following the link in the email, the user will navigate to the password reset page provided by the UserController and will enter the data required. Once the user invokes the `resetPassword` operation, the system will change the password and redirect him to the login page.

![Diagram of Game reset password use case](image.png)

**Solve exercise**

This is the main use case in this system, and has been designed to be a bit more complex than the original specification. When the user navigates to the game page, provided by the GameController, he will receive the game template, and will then iterate for each exercise he wants to solve first requesting the GameController for a task and then sending his results to the controller. This design allows the user to solve many exercises in one use case execution, without having to reload the game page.
View personal data

This use case is very straightforward, as it only requires that the user navigate to the User data page provided by the UserController from any other page while being logged in to the system.

Delete user

This use case has been designed to extend View personal data. Once in the User data page, the user can select to invoke the deleteCurrentUser request, by which the system will delete the logged in user, log him out and redirect him to the home page.
Enter platform security data

This use case allows the administrator to provide crowd platform details to the system by navigating to the User data page and then to the Administration panel page, provided by the PlatformAdminController only to the administrator.
6.4 Controllers

We have divided the possible requests into controllers based on the data they manage, and have obtained the set of controllers pictured on figure 26. Once we have distinguished the necessary operations for interaction with our users, we can proceed to describe their design in more detail.

**Figure 26: Game system controllers**

**Home Controller**

**Operation** loadHome

*Inputs:*

*Output:*

*Preconditions:*

*Postconditions:* Provides the home page by informing the framework of the home page view.

**User Controller**

**Operation** loadRegister

*Inputs:*

*Output:*

**Game Controller**

**Operation** loadGame

*Inputs:*

*Output:*

*Preconditions:*

*Postconditions:*

**PlatformAdmin Controller**

**Operation** loadAdministrationPanel

*Inputs:*

*Output:*

*Preconditions:*

*Postconditions:*
Preconditions: The user is not logged in

Postconditions: Provides the register page by informing the framework of the register page view and providing it with an empty instance of the Registration class

Operation processRegistration

Inputs: registration: an instance of the Registration class with its attributes filled with the data provided by the user in the register page form

Output:

Preconditions: The user is not logged in

Postconditions: • The system validates the format of the data provided, and returns the user to the register page if there are any errors
  
  • The system creates a new user with the data provided and stores him in the system
  
  • The system logs the new user in and redirects him to the game page
Operation loadLogin

Inputs:

Output:

Preconditions: The user is not logged in

Postconditions: Provides the login page by informing the framework of the login page view

Operation login

Inputs:

• username: a String containing the user’s username
  
• password: a String containing the user's password

Output:

Preconditions: The user is not logged in

Postconditions:

• The system checks the user’s credentials, and if they are not correct returns him to the login page with an error message

• The system logs in the user and redirects him to the game page

Operation logout

Inputs:

Output:

Preconditions: The user is logged in

Postconditions:

• The user is logged out of the system

• The system redirects the user to the home page
**Operation** forgotPassword

**Inputs:**
username: a String containing the username or email of the user

**Output:**

**Preconditions:**
- The user is not logged in
- The user has an account in the system

**Postconditions:**
- If there is no user in the system with the given username or email, the user is returned to the login page with an error message
- A new PasswordResetRequest is created for the user and stored in the system
- An email is sent to the user’s email address with a link to the password reset page
Operation loadPasswordReset

*Inputs:* requestId: a String containing the requestId sent by email to the user, encoded in the link provided

*Output:*

*Preconditions:* The user is not logged in

*Postconditions:* • Provides the password reset page by informing the framework of the password reset page view

• Provides the framework with an empty instance of the PasswordResetData class

• Adds the requestId to the PasswordResetData instance

Operation resetPassword

*Inputs:* data: an instance of PasswordResetData with the attributes filled in with the data provided by the user through the password reset page form

*Output:*

*Preconditions:* The user is not logged in

*Postconditions:* • The system validates the format of the data provided and returns the user to the password reset page if there are errors or the requestId and user do not match

• The system changes the password and deletes the existing PasswordResetRequest for the user

• The system redirects the user to the login page
Operation: loadPolicy

Inputs:

Output:

Preconditions:

Postconditions: Provides the policy page by informing the framework of the policy page view

Operation: loadUserData

Inputs:

Output:

Preconditions:

Postconditions:  
- Provides the user data page by informing the framework of the user data page view
- Provides the user model instance to the view
### Operation deleteCurrentUser

**Inputs:**

**Output:**

**Preconditions:**

**Postconditions:**
- Removes the currently logged in user from the system
- Logs out the user
- Redirects the user to the home page

---

### Game Controller

**Operation** loadGame

**Inputs:**

**Output:**

**Preconditions:** The user is logged in

**Postconditions:** Provides the game page by informing the framework of the game page view
**Operation**

**getTask**

**Inputs:**

**Output:**
an instance of the ProblemOutput class with all the fields necessary to describe a Problem to the interface

**Preconditions:**
The user is logged in

**Postconditions:**
Provides a task from the ones stored in the system to the interface

**Operation**

**saveExecution**

**Inputs:**

execution: an instance of the ExecutionResults class with the collected results from a Problem execution

**Output:**

**Preconditions:**
The user is logged in

**Postconditions:**
The system processes the execution for remote storage in the crowd system
Platform Admin Controller

**Operation**

**loadAdministrationPanel**

*Inputs:*

*Output:*

*Preconditions:* The user is logged in and is an administrator

*Postconditions:* Provides the admin page by informing the framework of the admin page view

**Operation**

**saveAdministrationData**

*Inputs:* data: an instance of PlatformData with the data collected in the admin page form

*Output:*

*Preconditions:* The user is logged in and is an administrator

*Postconditions:* The system stores the received data and returns the user to the admin page
6.5 Views

Similarly to the Crowd platform, we must analyze the relationship between controllers and view templates used to form the final views displayed to the users.

For the Game system, we have four controllers managing views:

**Home Controller**
User Controller

```
<template> login.jsp
<screen> Login page

<template> register.jsp
<screen> Register page

<controller> User Controller
<template> policy.jsp
<screen> Policy page

<template> password-reset.jsp
- requestid : long
<screen> Reset page

<template> user.jsp
- user : GameUser
<screen> User page
```

Game Controller

```
<controller> Game Controller
<template> game.jsp
- user : GameUser
- problem : Problem
<screen> Game page
```
Platform Admin Controller
6.6 Services

Up until now we have seen the interactions between the user and the system as well as the operations needed by the controllers, and have detailed the behavior of non-trivial operations. We can now move our focus to the Services level, where the rest of the logic can be found. In order to abstract the controller logic from the data storage implementation details, all services have been defined as interfaces, and we have then created specific classes that implement these services for a specific storage. We will omit the details of those operations which simply interact with the database, and will describe those that have additional behavior.

The operations described until now require the use of four services, one for users, problems, executions and platform data. We will see that due to the requirements of these four we will have two additional services that manage problem collections and remote communication. The final set of service interfaces and classes can be found in figure 27.

Figure 27: Set of Services and operations contained in the Game system.
Game User Service

Operation getCurrentUser

Inputs:

Output: An instance of GameUser with the attributes filled with the data for the currently logged in user

Preconditions:

Postconditions:

Operation saveUser

Inputs: user: an instance of GameUser

Output:

Preconditions: The system stores the GameUser. If an existing user is found with the same username, it is replaced. Otherwise, it is stored as a new user.

Postconditions:

Operation getUserByUsernameOrEmail

Inputs: username: a String containing the username or email of a user

Output: An instance of GameUser matching the provided input by username. If none is found, an instance matching the provided input by email. Otherwise, a null instance.

Preconditions:

Postconditions:

Operation getUser

Inputs: username: a String containing the username of a user

Output: An instance of GameUser matching the provided input by username. If none is found, a null instance.

Preconditions:

Postconditions:
Operation removeUser

Inputs: user: an instance of GameUser

Output:

Preconditions:

Postconditions: If the user exists and is stored in the system, it is removed from storage.

Problem Service

Operation getProblem

Inputs:

Output: An instance of the class Problem containing the data of a valid exercise in the system

Preconditions:

Postconditions:

This operation is worth detailing. We have implemented the behavior for the interface ProblemService in the service class ProblemServiceImpl, following the behavior stated below. When a request for a Problem arrives, the service first tries to return a problem from the regular collection, which is populated with problems from the crowd platform. If the number of problems reaches a low level, new problems are requested. If the collection is empty, we retrieve a backup collection with backup problems stored in this system. This will allow us to keep on running even if the communication with the crowd platform is broken. We will retrieve a problem at random from the appropriate collection, and if it was from the normal collection we will erase it to free up space for new problems.
We must now detail the implementation of the private method `retrieveMoreProblems`, which will simply delegate the responsibility to the `RemoteCommunicationService`, who is in charge of communicating with the crowd platform.
Execution Service

Operation: saveExecution

Inputs: execution: an instance of ExecutionResults with the data gathered during an exercise execution

Output:

Preconditions:

Postconditions: The execution data is stored

We have previously mentioned that we decided to store executions in the crowd platform, and therefore have no internal storage in this system that takes care of executions. The implementation of Execution service, named ExecutionServiceImpl, takes care of delegating the task of storage to the Remote communication service, who is in charge of communicating with the crowd platform.

Platform Data Service

Operation: getPlatformData

Inputs:

Output: The single existing instance of PlatformData stored in the system

Preconditions:

Postconditions:
**Operation**  
**savePlatformData**

*Inputs:*  
data: an instance of PlatformData

*Output:*  

*Preconditions:*  

*Postconditions:*  
The data is stored in the system, overriding any other existing Platform-Data instances.

---

**Problem Collection Service**

**Operation**  
**getCollection**

*Inputs:*  

*Output:*  
The instance of ProblemCollection in the system that contains the main Problems, obtained from the remote crowd platform

*Preconditions:*  

*Postconditions:*  

**Operation**  
**getBackupCollection**

*Inputs:*  

*Output:*  
The instance of ProblemCollection in the system that contains the backup Problems, created locally for backup purposes

*Preconditions:*  

*Postconditions:*  

**Operation**  
**removeProblemFromCollection**

*Inputs:*  
- collection: an existing instance of ProblemCollection
- problem: an instance of Problem

*Output:*  

*Preconditions:*  

*Postconditions:*  
If the problem is contained in the collection provided, it is removed from the collection and this change is stored.
Remote Communication Service

Operation addTasksToProblemCollection

Inputs: 

Output: 

Preconditions: 

Postconditions: The service connects asynchronously to the remote crowd platform system and obtains new tasks, converts them to local Problems and stores them in the main ProblemCollection.

Operation postExecutionResults

Inputs: execution: An instance of ExecutionResults with the results of an exercise execution

Output: 

Preconditions: 

Postconditions: 

• The service creates a new instance with the output format for an execution and assigns it the information contained in the provided execution as well as the current user platformId.

• The service sends the data remotely to the crowd platform system.
6.7 Utils

Finally, we must take into account the classes in the *util* package, which have appeared along the way.

![Figure 28: Game system utilities](image)

**MailSender**

**Operation** sendPasswordResetEmail

*Inputs:* 
- user: The GameUser to who the email is addressed
- request: The request for which the email is sent

*Output:*

*Preconditions:*

*Postconditions:* An email is sent to the user's email address containing a link for the password reset page with an embedded reference to the request.

**PasswordResetDataValidator**

**Operation** validate

*Inputs:* data: An instance of PasswordResetData with the information introduced by the user

*Output:*

*Preconditions:*
Postconditions:

- The system checks that the data is correctly formatted.
- The system checks that the password and confirmation password introduced are the same.

RegistrationValidator

Operation validate

Inputs: registration: An instance of Registration with the information introduced by the user

Output:

Preconditions:

Postconditions:

- The system checks that the data is correctly formatted.
- The system checks that the username is not already taken by another user
- The system checks that the password and confirmation password introduced are the same.
6.8 Packages

With this, we have seen a complete overview of the classes included in the project's packages as well as their operations. For a final wrap-up, we present the package diagram, summarizing the packages in figure 29.

Figure 29: Game system packages
6.9 Game interface

Another important part in our project we must consider is the Game interface. Even though it technically does not appear in our analysis, specification and design up to this point, it is a key part of the Game system and we have devoted great efforts to it.

The Game Interface is a sub-system that works to present the exercises to our Players in the Game page within the Game system. As such, its logic is capable of understanding all the attributes of a Problem as well as all the possible types of problems and presenting them to the Player in an interactive manner.

6.9.1 Game objects

The design taken for the Game Interface is mainly derived from the Dyseggxia application design, with several additions. First, we show the Game Object hierarchy, which depicts the objects that exist in our interface.

Figure 30: Game system interface objects
**Game Object**

Game Object is the root object that represents a drawable element. As such, it contains its bounds, represented by an origin point and a certain width and height. This will allow the object to know where on the canvas it must draw itself. All elements descending from a Game Object will override the `draw` operation which will include the concrete implementation for that element's rendering on screen.

**Bounds**

Bounds is an auxiliary class that structures the data needed to define the bounds of an object, mainly its origin point and a width and height.

**Point**

Point is an auxiliary class that encapsulates two coordinates, an $x$ and $y$, which represent a point on the canvas.

**Dialog**

Dialog is a generic class that defines the needed logic to present a dialog on screen. As such, it represents the main dialog background shared by all dialogs.

![Figure 31: Visual representation of Dialogs. On the left, the Start Dialog. On the right, the Continue Dialog.](image)

**Start Dialog**

This class represents the dialog shown at the start of a game session, which includes a Start button.

**Continue Dialog**

This class represents the dialog that is shown each time the user completes and exercise. It includes the completed problem’s answer as well as a continue button.
Figure 32: Visual representation of the Loading Signal.

Figure 33: Visual representation of the Visual Arrow.

Figure 34: Visual representation of the Visual Button.

Loading Signal
This class represents the Loading message that appears in an Empty Game Controller, when the interface has no Problem to present.

Visual Arrow
This class represents a red arrow, as seen in the Skip exercise button.

Visual Button
This class represents a green label, that can be used as a button.

Tile Layout
This generic class represents a layout of tiles, such as the problem displayed or the answers displayed. It has three subclasses for each particular type of layout that will appear in the problems. As a superclass to these layouts, it holds the tiles to be rendered and helps with interaction operations such as determining the child under a click action.

Word Layout
A Word Layout is the main problem layout, used to place the displayed problem in the center of the screen. It arranges its tiles horizontally, and takes care of animations.

Answer Layout
An Answer Layout is similar to a Word Layout, as it displays its children horizontally, but it is placed in the bottom half of the screen and is used to hold the answers to a problem if it has any. Furthermore, it renders itself as a tile support.

Vertical Answer Layout
This layout displays its children vertically arranged, and is used in the Derivation exercises to display the possible terminations.
Word Cube
This class represents a tile inside a layout, with a text contents of one or more letters.

Render Utils
This file is not actually a class, but contains utility rendering methods such as the ribbon shape and general color definitions.

6.9.2 Game Controllers
We can now proceed to take a look at the controllers, which handle objects for interaction and rendering. At the base of the execution we find the Application Manager, who then owns a Game Controller as seen in figure 35.

Figure 35: Game system interface controllers
**Application Controller**

This main class takes care of network requests and general setup. At the beginning, it creates a temporal controller while it waits for the first problem to load, and then takes care of the cycles of receiving problems, preparing the correct controller in each case and sending results to the Game platform.

**Game Controller**

This is the base class for a controller. As such, it defines all the generic behavior and attributes necessary to run a controller, which can then be overridden by subclasses.

As we can see, a Game Controller contains a Problem to display, as well as several other items. The instructions, places on the top left side of the screen, as presented as a Visual Button, while the skip button in the lower right side of the screen is a Visual Arrow. It holds reference to two layouts, one central Word Layout and one answerLayout which will be defined by subclasses if necessary. Additionally, it may hold a reference to a dialog to be displayed. The Game Controller collects all items that should be rendered in an ordered collection which he iterates during the render function.

**Problem**

This class does not explicitly exist as a file, but a structure with all the fields sent as a ProblemOutput from the Game system is created from the received input for a Problem and assigned to a Game Controller.
Figure 36: Visual representation of a Game Controller with all its elements.

Empty Game Controller
This controller is used when the user is not viewing a dialog and there is no problem to present. It displays an animated Loading Signal to inform the user.

Paused Game Controller
This controller is used when the Start Dialog must be presented to the user, and controls the interaction with the central button it contains.

Insertion Game Controller
This controller is used to display an Insertion exercise. It displays the central Word Layout as well as an Answer Layout at the bottom, and allows answer tiles to be selected or dragged to the blank.

Hard Insertion Game Controller
This controller is used to display a Hard Insertion exercise. It displays the central Word Layout as well as an Answer Layout at the bottom like the Insertion Game Controller, but there is no blank space in the Word Layout.

Omission Game Controller
This controller is used to display an Omission exercise. It displays the central Word Layout and allows tiles to be removed from it.
Figure 37: Visual representation of all the exercise Game Controllers. From left to right and top to bottom: Insertion, Hard Insertion, Omission, Substitution, Transposition, Derivation and Separation.

**Substitution Game Controller**

This controller is used to display a *Substitution* exercise. It displays the central Word Layout as well as an Answer Layout at the bottom, and allows tiles to be swiped.

**Derivation Game Controller**

This controller is used to display a *Derivation* exercise. It displays the central Word Layout as well as a Vertical Answer Layout on the right, and allows answers to be selected or dragged on top of the Word Layout.

**Transposition Game Controller**

This controller is used to display a *Transposition* exercise. It displays the central Word Layout and allows the tiles to be reordered, animating the background tiles as one is dragged.

**Separation Game Controller**

This controller is used to display a *Separation* exercise. It displays the central Word Layout and allows the user to select a space between tiles or cut to separate them.
6.10 Data model

The Game system follows a traditional relational model. The data model containing all entities to be persisted in the system is as follows:

![Game system data model diagram](image)

**Figure 38**: Game system data model

Additionally, we have been able to define constraints on the entities and their attributes with annotations. These include the following constraints:

- The ProblemCollection `internalId` is the entity identifier and should be automatically generated by the system.
- The Problem `id` is the entity identifier and should be automatically generated by the system.
- The PlatformData `id` is the entity identifier.
- The GameUser `username` is the entity identifier.
- The PasswordResetRequest `requestId` is the entity identifier.

Using Hibernate Annotations, we obtain an automatically generated schema for the database. As we can see, the multivalued attributes such as `answers` within the entity `Problem` become new tables automatically. A complete description of the generated schema can be found in the appendix, section 12.1.
Figure 39: Game system database schema
7 Deployment

Figure 40: Deployment model

Our two projects have been deployed to a platform named Heroku [27]. Heroku is a cloud application platform that automatically takes care of infrastructure for deployed apps. Therefore, it is responsible for server management as well as for increasing the amount of infrastructure assigned to an app as needed. It provides elastic scalability, by managing demand automatically and charging its users only for the resources used. We selected this platform for deployment in front of other options such as deploying to a proprietary server due to its simplicity. By using Heroku, we have isolated our work from infrastructure problems and have been able to center our efforts on building our systems. There are other similar products in the market, such as Amazon’s Web Services [28]. Nevertheless, these options only include the server instance and leave actual deployment to the user.

At a basic development level, Heroku is free of charge. We have maintained our apps at this level, as we did not have a real budget for the project. This allows us to have each project deployed and run on a single server instance, but the instance is reused when the project is not accessed for a short time and therefore carries a small start-up time the first time any of the two systems are accessed.

Heroku allows the user to select “add ons” for his projects, which also have different levels and are usually free at the basic level. We added an instance of PostgreSQL to the Game system and an instance of MongoHQ to our Crowd system using this capability. The PostgreSQL add on provides us with relational storage up to certain limits we were sure would not be exceeded during testing, and the MongoHQ add on provides us with access to a simple instance of MongoDB deployed on a MongoHQ server and capable of storing 512MB of data. Access to this instance is automatically provided by Heroku through environment configuration and appears to us as any MongoDB instance. Additionally, Heroku includes an automatic backup addon that makes regular copies of the data stored in both systems.

Heroku works on a git-based deployment system. By properly configuring a git repository on the project we want to deploy and setting a remote to heroku’s servers, we can deploy by simply pushing our changes to Heroku. The system receives these changes and automatically compiles and deploys...
our project to a tomcat server. Furthermore, Heroku offers prepared project templates, including a template prepared with Spring and Hibernate integration which we used as a base for our two systems.

Additionally, we have synchronized our projects with GitHub. This allowed us to control the flow of work done as well as to visualize changes online and to have a copy of our work on the cloud. Given that the Heroku projects were already initialized with Git, we easily set up the remote to GitHub necessary to upload our code changes there.

The two systems developed in this project are publicly available and accessible from a web browser.

- Crowd platform: http://crowd.clarabayarri.com
- Game: http://mapuche.clarabayarri.com

Similarly, the two GitHub repositories with the code are publicly accessible through the GitHub website.

- Crowd repository: https://github.com/clarabayarri/crowd
- Game repository: https://github.com/clarabayarri/crowd-game
8 Testing

Testing has been done exhaustively in this project at two levels: unit testing and functional testing. Unit testing has been done for Java and Javascript code, while functional testing has been done to verify the functional requirements were met.

8.1 Unit tests

Unit testing has been very important in this project. Unit tests have been developed in parallel with the code, ensuring that the code was majorly covered by tests at all times. In Java, all classes with behavior have been covered by tests. This only excludes Java model classes, as we have not tested simple getter and setter methods. Detailed coverage can be found in figures 41 and reftest-coverage-game.

Java testing has been done using JUnit, a Java testing framework, along with Mockito, a mocking framework that allows us to isolate classes from their relations when testing.

![Figure 41: Test coverage summary for the Crowd system, provided by eCobertura.](image1)

![Figure 42: Test coverage summary for the Game system, provided by eCobertura.](image2)

For the game interface, we have used QUnit, a javascript testing framework. We have not covered all behavior as most of it is interleaved with interface events, but have covered with tests as much as possible.
### 8.2 Functional tests

We now define a set of tests that will check if the functional requirements have been met in our project.

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Introducing tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actions:</strong></td>
<td>• The user enters the create batch page</td>
</tr>
<tr>
<td></td>
<td>• The user enters data for a new batch and a file containing the task definitions</td>
</tr>
<tr>
<td><strong>Satisfaction require-ment:</strong></td>
<td>• The system transforms the provided file into a set of tasks</td>
</tr>
<tr>
<td></td>
<td>• The system creates a new batch with the data provided</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Obtaining results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actions:</strong></td>
<td>• The user enters a batch page</td>
</tr>
<tr>
<td></td>
<td>• The user clicks the download button</td>
</tr>
<tr>
<td><strong>Satisfaction require-ment:</strong></td>
<td>• The system produces a file with the data gathered and allows the user to download it</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Progress visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actions:</strong></td>
<td>• The user enters a batch page</td>
</tr>
<tr>
<td><strong>Satisfaction require-ment:</strong></td>
<td>• The system shows all the available data for that batch, including progress</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Execution control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actions:</strong></td>
<td>• The user enters a batch page</td>
</tr>
<tr>
<td></td>
<td>• The user clicks the start/stop button</td>
</tr>
<tr>
<td><strong>Satisfaction require-ment:</strong></td>
<td>• The system changes the batch state and shows this change to the user</td>
</tr>
<tr>
<td>Functional requirement</td>
<td>Data visualization</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| **Actions:**           | • The user enters a batch page  
                        | • The user clicks the view data button |
| **Satisfaction require-ment:** | • The system exports the gathered data to Fusion Tables and redirects the user |

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Batch deletion</th>
</tr>
</thead>
</table>
| **Actions:**           | • The user enters a batch page  
                        | • The user clicks the delete button |
| **Satisfaction require-ment:** | • The system deletes the batch  
                        | • The system shows the existing batches to the user, and the previous batch is not there anymore |

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>Task providing</th>
</tr>
</thead>
</table>
| **Actions:**           | • The user enters the game page  
                        | • The user clicks the start button |
| **Satisfaction require-ment:** | • The system provides the interface with a problem to solve  
                        | • The interface displays the problem to the user |

<table>
<thead>
<tr>
<th>Functional requirement</th>
<th>User data visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actions:</strong></td>
<td>• The Game system user enters the user page</td>
</tr>
<tr>
<td><strong>Satisfaction require-ment:</strong></td>
<td>• The system shows the user all the personal data he provided during registration</td>
</tr>
</tbody>
</table>
Functional requirement  User deletion

**Actions:**
- The user enters the user page
- The user clicks the delete user button

**Satisfaction requirement:**
- The system deletes the user and logs him out of the system
- The user is no longer able to log in to the system

8.3 Performance tests

This project contains no performance tests. This has been done on purpose, as performance tests mainly affect the deployment environment where the project is deployed and we have kept our project in a development environment. Given the current deployment to Heroku, we can easily increase automatically the number of instances dedicated to receive requests to our project but this comes with a financial cost we do not believe is necessary at this point.
9 Management

In this section we will tackle with management for the project as a whole. We will introduce the final planning, comparing it to the initial planning as well as the budget, and we will briefly talk about the methodology followed, laws that affect the project and sustainability and social compromise.

9.1 Planning

In order to talk about planning we must first introduce the resources consumed in order to complete our goals and the monitoring we have carried out to control the schedule and make sure it was met. Then, we can proceed to introduce the planning itself, from its initial version to its final version.

9.1.1 Resources

The main resource for this project is the author, who will act as project manager, analyst, designer, developer and tester. This resource is common to all phases of the project, and should not be used further than the hours allocated for this project. We can consider other resources such as web hosting or development tools, but we will use systems that are always freely available and therefore do not need previous planning. Furthermore, we should consider the referenced bibliography as a resource needed throughout the project.

9.1.2 Schedule monitoring

In order to monitor the progress of the project related to this schedule, we have established a series of milestones\(^1\). These will be located at the end of each iteration and will represent a control point in the project. Once each milestone is reached, we will check that the functionality added to the system is complete and fully functional and, if necessary, we will make corrections and adjustments. This will also be the point where we will be able to compare the progress with the schedule and, if necessary, correct the schedule to any delays appreciated.

The schedule has been elaborated including a period of time for refactoring and debugging at the end of each phase that will hopefully provide the necessary time to correct deficiencies without affecting the overall schedule. Furthermore, the schedule has been planned to ensure an early completion of development and a long user testing phase. Nevertheless, this user testing phase can be shortened without serious consequences in the overall project deadline.

9.1.3 Initial planning

Initially, we considered the a set of phases that included all the tasks we then foresaw would be required to complete our project. This division into phases included the features to be developed as well as system deployment and user testing.

Each phase represents a big iteration. The first six phases are focused on development, and include the analysis, design, implementation and testing of the features they contain. Therefore, we have six points along the project where we will be able to establish milestones.

\(^1\)Milestones are represented in the Gantt diagram by the appearance of dates next to task names.
**System core**

Priority: very high

Requirements: none

This phase will include the minimum functionality necessary to have a first functioning system. By the end of this phase, we aim to achieve a system that obtains basic tasks from the database, interacts with the user to solve these tasks and saves the execution results. This will be the basis for all other iterations, containing the core functionality necessary.

Tasks included:

1. **Game simple task execution.** Given a task of the most simple type "insertion", the game must show the task to the user in a simplified interface and allow for interaction. The game must control the expected results for the execution: number of failed intents, time taken to solve the task and wrong solutions given. *Estimated difficulty: medium*

2. **Serve task from database to game.** When requested through the API, the platform must send a task from the database to the requester. *Estimated difficulty: low*

3. **Save results from execution to platform.** When requested through the API, the platform must save the results given for an execution in the database. The game must send a request when the task execution is complete, sending the data previously controlled. *Estimated difficulty: low*

4. **Show batches on platform with progress.** Show the existing batches for a project in the database in the administration interface, as well as their calculated progress. *Estimated difficulty: medium*

**System core improvements**

Priority: medium

Requirements: The system core must be complete.

This phase will include improvements to the initial system core. As such, it has less priority and will be mainly focused in improving the game interface and interaction and extending the types of tasks supported to the 5 types necessary for our project.

Tasks included:

1. **Improve task interface.** Improve the simple interface previously implemented to create an interaction with the user closer to the original Dyseggxia game. This includes formatting the problem by cubes and allowing the user to drag answers around the screen as in the original application. *Estimated difficulty: high*

2. **Extend to 5 types of tasks.** Incorporate the necessary changes to allow execution of the 5 types of exercises in Dyseggxia. *Estimated difficulty: high*

3. **Rework task interface to final version.** Work on the game interface to give it a polished appearance and smooth interaction. *Estimated difficulty: medium*
Tasks
Priority: high
Requirements: The system core must be complete.
This phase will include the functionality necessary to create and manage tasks within the platform.
Tasks included:

1. **Allow to start/stop batches.** The administrator must be able to control batch execution. Provide functionality to start and stop the execution of a batch. *Estimated difficulty: medium*

2. **Create batches by uploading tasks.** The administrator must be able to create batches of tasks by providing the necessary data and a file containing the task data. This includes reading the provided file and saving all data to the database. *Estimated difficulty: high*

Results
Priority: high
Requirements: The system core must be complete.
This phase will include the functionality necessary to visualize and download task execution results. This includes data visualization, download and online manipulation.
Tasks included:

1. **Download raw results.** Provide the administrator with a means of downloading a file that contains all the task execution results. This includes generating the file. *Estimated difficulty: medium*

2. **Show results online.** The administrator must be able to visualize all the results online, including all fields from tasks as well as results. *Estimated difficulty: low*

3. **Manipulate results online.** Provide the administration with a means to interact with the data by sorting columns or aggregating fields. *Estimated difficulty: high*

User management
Priority: medium
Requirements: none
This phase will focus on user registration and access to both systems.
Tasks included:

1. **Game login / logout.** Develop the user login management for the game. This also includes restricting access to users that are not logged in. *Estimated difficulty: medium*

2. **Game player registration.** Provide registration functionality in the game, as well as the functionality necessary for these users to be registered in the platform and associated to the current project through the API. *Estimated difficulty: low*
3. **Platform login / logout.** Develop the user login management for the platform. *Estimated difficulty: low*

**User data**

Priority: low

Requirements: none

This phase will focus on calculating and showing personal data to the game users.

Tasks included:

1. **Allow the definition of data to be calculated for each user.** Provide the necessary functionality to define how user data must be calculated from task execution results. *Estimated difficulty: medium*

2. **Update data on each task execution.** Develop the necessary functionality to update user data on task execution. *Estimated difficulty: low*

3. **Show user data on game for the logged in user.** From the game, request and show user data stored and calculated on the platform. *Estimated difficulty: low*

4. **Link a player account with a Facebook account.** Integrate game registration with Facebook in order to link accounts. *Estimated difficulty: low*

5. **Share personal score on Facebook.** Allow the user to share on Facebook his user data. *Estimated difficulty: low*

**System deployment**

Requirements: All previous phases must be complete.

Once the system is complete, it can be deployed to allow public access to the game and platform.

**User recruitment and testing**

Requirements: The system must be deployed.

This phase will focus on finding players for our game and engaging them in resolving tasks to obtain data on the platform. This will be done by social media. Ideally, we will be able to gather 2-3 weeks of user data before the completion of this project, which will allow us to validate the system with users, find undetected deficiencies and determine the validity of the data gathered.

We have seen there are certain dependencies between some of the phases described. These can be resumed in the following diagram:

Therefore, the order of execution of the project was designed taking these dependencies into account and following the order in which they were presented. Given that the project duration is quite long and that the work has been divided into several phases and tasks, the resulting Gantt diagram is extensive and can be found in the appendix section 12.2.1.

Given the initial planning, project implementation should finish by the end of May providing between 2 and 3 weeks for user testing and ensuring the project is complete around the beginning of June.
9.1.4 Control planning

The evolution of the project at the control point had naturally lead to certain adjustments in the initial planning. The planning produced at the beginning of the project stated the development iterations in an order that respected dependencies between functionalities, but ordered non-dependant iterations in a sequential non-related manner. These iterations were reordered during the development of the project as functionality became more clear.

Up until the control point, we had focused on producing a functioning system early on, and maintaining it as we introduced new functionality. Therefore the iterations related to functionality that influenced the overall project were prioritized.

Some of the initial iterations were split into two in order to separate critical from secondary functionalities. Each new set of iterations contains all the tasks from the initial iteration, as well as an analysis and design phase and a refactor and debug phase. The weight of these two auxiliary tasks was split between the new iterations, in order to maintain the total hour count of the project.

Given that the new iterations generated by splitting initial iterations separated functionality by its urgency, iterations were dynamically reordered to fit the best evolution. Once an iteration was finished and revised, the following iteration was chosen from the available list. Figure 45 shows the order up to the control point, as well as the remaining iterations.

Given the pace at this point, the project was complying with its planning, and would be complete as expected.
A second version of the Gantt diagram can be found in the appendix, under section 12.2.2. This version includes the changes in planning mentioned, such as iteration splitting and reordering. It also displays the work pending at this point, which fits into the planned schedule.

During the development of the project up to the control point, we found some tasks were slightly faster than expected. We took advantage of these little free slots to include small functionality that was initially left out of scope of the project and can improve usability with little effort, such as batch deletion.

Furthermore, some small features were added with the only purpose to learn more about the technology used in this project. This is the case for example of user password resetting, a small functionality that allowed us to learn how to send emails and protect petitions from the platform.

9.1.5 Final planning

The planning has suffered several changes since the control point, until it has reached its final state. This includes phase additions as well as date rearrangements, which we will now see.

First of all, we notice several phase redefinitions and additions. The first pending phase from the control point was completed as scheduled, but the second phase *Results manipulation* was redefined. This was due to design decisions, as during the phase analysis we decided to export and manipulate data through Google Fusion Tables. This was a functional simplification from the original planning, and therefore the phase was shortened from 9 days to 4.

The User Data phase was completely redefined during a meeting with the project supervisor, and the tasks included changed. The User Data phase had been described as a phase where we would define user attributes from the Crowd platform that would be sent to the Game platform. Nevertheless, we reached a conclusion that this data visibility for the user could impact the process of collecting data, as users would be able to observe their data and try to manipulate it. Therefore, we changed the required tasks in this phase to include personal data visualization within the Game platform for each user and user deletion, a required functionality that had appeared with the analysis of legal requirements. At this point, we also notices the changes that would need to be done to the underlying database in the
Crowd system, which were shaped as the following phase *MongoDB migration*. Due to this addition, we decided to remove the tasks related with Social network inclusion, and left them for future work.

Finally, once we had defined the migration to a new database system, we also realized we were able to calculate data aggregations easily and felt curiosity towards implementing a Map Reduce functionality. Therefore, and given the extra time reduced from Social network functionality, we were able to add another new phase related to Graph data visualization, which included data processing using Map Reduce and graph visualizations.

<table>
<thead>
<tr>
<th>Control point iteration</th>
<th>New iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. System Core</td>
<td>1. System Core</td>
</tr>
<tr>
<td>2.1 Task Type Definition</td>
<td>2.1 Task Type Definition</td>
</tr>
<tr>
<td>3. Tasks</td>
<td>3. Tasks</td>
</tr>
<tr>
<td>4.1 Results Download</td>
<td>4.1 Results Download</td>
</tr>
<tr>
<td>5. User Management</td>
<td>5. User Management</td>
</tr>
<tr>
<td>2.2 Interface improvement</td>
<td>2.2 Interface improvement</td>
</tr>
<tr>
<td>4.2 Results Manipulation</td>
<td>4.2 Results Manipulation</td>
</tr>
<tr>
<td>6. User Data</td>
<td>6. User Data</td>
</tr>
<tr>
<td></td>
<td>7. MongoDB migration</td>
</tr>
<tr>
<td></td>
<td>8. Graph data</td>
</tr>
</tbody>
</table>

Figure 46: Final iteration evolution. On the left, the control point planned iterations. On the right, the iterations revised for the final planning.

All these changes, nevertheless, maintained our deployment date intact. Furthermore, with all the refactoring done during the evolution of the project, we found the *Main refactoring and debugging* phase was complete before the expected data and we were able to deploy and start releasing the Game platform to users a few days earlier than expected.

At this point we also learnt the exact date of defense of the project, and the corresponding deadline for project submission. Therefore, we were able to readjust the final testing and document writing tasks to span more days.

The final version of the Gantt diagram can be found in the appendix, section 12.2.3.
9.2 Budget

The budget is another important management topic we should cover. As we will see, the budget is divided in human and material costs, and then added to generate the total cost. We also present a control plan for the budget, and a small viability analysis.

9.2.1 Human Resources

The main cost in this project is given by the human resources necessary to carry it out. We can distinguish between 5 types of roles: project manager, analyst, designer, developer and tester. Nevertheless, given the project planning presented earlier, the tasks of analysis and design are joined together, as well as development and testing. In order to calculate the human resources total costs, we need to estimate the salary of these roles. Figure 47 presents a summary of the estimated salaries, taken approximately by information from the Catalan College of Informatics Engineers\(^2\) [29].

\[
\begin{array}{|l|c|}
\hline
\text{Resource} & \text{Salary} \\
\hline
\text{Project manager} & 40 \, \text{€/h} \\
\text{Analyst & Designer} & 40 \, \text{€/h} \\
\text{Developer & Tester} & 20 \, \text{€/h} \\
\hline
\end{array}
\]

Figure 47: Estimated cost for each type of human resource needed during project development.

We consider our working days to be 4 hours long, since the author of this project is also taking other courses and will not be able to work full time on the project. With this data, we can now estimate the cost of the tasks previously stated in the project planning. The final budget differs slightly from the original budget, which can be found in the appendix section 12.3.

\[
\begin{array}{|l|c|c|c|}
\hline
\text{Task name} & \text{Duration} & \text{Resource type} & \text{Cost (€)} \\
\hline
\text{Requirements definition} & 5d & \text{Manager} & 800 \\
\text{Project scope definition} & 3d & \text{Manager} & 480 \\
\text{Use case definition} & 3d & \text{Analyst} & 480 \\
\text{Project modelling} & 4d & \text{Designer} & 640 \\
\text{Project management course} & 14d & \text{Manager} & 2240 \\
\text{Project setup} & 2d & \text{Developer} & 160 \\
\text{System core} & & & \\
\text{Analysis & design} & 2d & \text{Analyst/Designer} & 320 \\
\text{Game simple task execution} & 4d & \text{Developer} & 320 \\
\text{Serve task to game} & 1d & \text{Developer} & 80 \\
\text{Save results to platform} & 1d & \text{Developer} & 80 \\
\hline
\end{array}
\]

\(^2\)Col·legi Oficial d’Enginyeria Informàtica de Catalunya
<table>
<thead>
<tr>
<th>Task name</th>
<th>Duration</th>
<th>Resource type</th>
<th>Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show batches on platform</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>1040</strong></td>
</tr>
<tr>
<td><strong>Task type definition</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Extend to 5 types of tasks</td>
<td>3d</td>
<td>Developer</td>
<td>240</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>720</strong></td>
</tr>
<tr>
<td><strong>Tasks</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Start / stop batches</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Create batches</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>800</strong></td>
</tr>
<tr>
<td><strong>Results download</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>1d</td>
<td>Analyst/Designer</td>
<td>160</td>
</tr>
<tr>
<td>Download raw results</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>320</strong></td>
</tr>
<tr>
<td><strong>User management</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Game login / logout</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Game registration</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Platform login / logout</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>800</strong></td>
</tr>
<tr>
<td><strong>Interface improvement</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>1d</td>
<td>Analyst/Designer</td>
<td>160</td>
</tr>
<tr>
<td>Improve task interface</td>
<td>4d</td>
<td>Developer</td>
<td>320</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>560</strong></td>
</tr>
<tr>
<td><strong>Results manipulation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>1d</td>
<td>Analyst/Designer</td>
<td>160</td>
</tr>
<tr>
<td>Task name</td>
<td>Duration</td>
<td>Resource type</td>
<td>Cost (€)</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>----------</td>
<td>-----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Export to Google Fusion Tables</td>
<td>3d</td>
<td>Developer</td>
<td>240</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td><strong>User data</strong></td>
<td></td>
<td></td>
<td><strong>480</strong></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>0.5d</td>
<td>Analyst/Designer</td>
<td>80</td>
</tr>
<tr>
<td>User personal data visualization</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td><strong>MongoDB migration</strong></td>
<td></td>
<td></td>
<td><strong>160</strong></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>1d</td>
<td>Analyst/Designer</td>
<td>160</td>
</tr>
<tr>
<td>Migration to Mongo</td>
<td>4d</td>
<td>Developer</td>
<td>320</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td><strong>Graph data</strong></td>
<td></td>
<td></td>
<td><strong>560</strong></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>1d</td>
<td>Analyst/Designer</td>
<td>160</td>
</tr>
<tr>
<td>Data calculation</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Graph visualization</td>
<td>3d</td>
<td>Developer</td>
<td>240</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td><strong>Main refactor and debugging</strong></td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>System deployment</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Write project document</td>
<td>17d</td>
<td>Manager</td>
<td>2720</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>113.5d</strong></td>
<td></td>
<td><strong>14,520</strong></td>
</tr>
</tbody>
</table>

Figure 48: Total cost estimation for human resources related to each task.

As we can see, there has been a slight increase in the project budget, which was initially estimated at 12,720 €, with 108 days of work. This is due to the extra days added to the final task Write project document, which have increased the total amount of working hours dedicated to the project.

9.2.2 Material Resources

The development of this project also entails a series of material resources detailed below. Given that there is no real budget for the project, we have selected development and deployment tools that are free of charge and will be using personal equipment to develop the project.

If we are to compute the equipment amortization, we must take into account the official amortization coefficient stated by the Spanish government, currently fixed to 26% per year [30][31]. We must take into account the fraction of a whole year equivalent to the length of this project, with a standard day’s
work of 8 hours. Since our working days consist of 4 hours, we must compute the amortization for 56.75 days in a year:

\[
1000 \text{ €} \times 0.26 \times 56.75 / 365 = 40.42 \text{ €}
\]

<table>
<thead>
<tr>
<th>Resource</th>
<th>Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work station</td>
<td>40.42</td>
</tr>
<tr>
<td>Development tools</td>
<td>0</td>
</tr>
<tr>
<td>Deployment tools</td>
<td>0</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>40.42</strong></td>
</tr>
</tbody>
</table>

**Figure 49:** Estimated cost for each type of material resource needed during project development.

### 9.2.3 Total Cost

If we take into account the previously stated calculations, we will see that the total project cost will be

\[
14,520 + 40.42 = 14,560.42 \text{ €}
\]

As can be expected, the slight increase in human resources costs is reflected in the total cost, which has also slightly increased from the initially estimated 12,758.5 €. The increase however has been dedicated to writing a better project document and therefore a better thesis, which the author considered to be appropriate.

### 9.2.4 Control Plan

This project's costs are directly related to the number of hours spent on project development. Therefore, we must control the length of the project in order to control the budget. As stated during the project planning, several milestones were fixed along the development process of this project. Each milestone represented a control point, where we compared the progress done with the initial schedule and made the necessary corrections to continue and ensure the project was completed on time. Therefore, we were also able to control the budget in each control point and ensure it did not increase. Nevertheless, since there is no real budget for the project, we made a bigger emphasis on time control than on budget control, ensuring the project development was smooth and finished in time for the deadline. This goal was achieved, as the project followed the schedule perfectly and actually managed to finish some tasks before the expected date, fact that encouraged us to add more functionality.

### 9.2.5 Viability

The development of this project has been done within the context of a thesis, and therefore did not consider the search for funding resources. Nevertheless, the project itself will be continued after this
thesis is completed by two major parties, the research group interested in the development of the generic crowdsourcing platform and the team behind Dyseggxia, interested in further data analysis. In this case, we should consider the viability of the project in the future.

The crowdsourcing platform will be kept and further developed by the crowdsourcing research group at UPC, and therefore will be sustained by their research at the university. The maintenance of the platform will not require special resources that are not already available, and therefore will not represent an increase in economic or environmental resources. A proper documentation has been generated to ensure that it is easy to comprehend for people working on it in the future.

The game will be further used in studies by the Dyseggxia team, helping them improve their research in the field of Dyslexia and helping them achieve a real social impact. In order to continue with this area of research, the team will have to supply the same resources they have done up to date, and will probably require the aid of external funding to secure their financial viability.

9.3 Methodology

As stated at the beginning of the project, we have followed an iterative evolution. The functionality to be developed was split into several iterations, where functionality from both domains that is interrelated was kept together. Each iteration expanded the specification, design and implementation of the project, and was accompanied by the necessary tests to validate the functionality developed. Starting by the core functionality necessary for both platforms, we iteratively extended the project to include all the functionality described in the project scope.

In order to ensure that the correct pace was followed, we established periodic meetings with the supervisor of the project. Following the initial planning, we met every other week to comment the work done and discuss improvements. This methodology worked, and allowed us to move forward while maintaining regular discussions on possible improvements.

The code generated by the project has been synchronized with two version control repositories, one for each platform developed. These can be accessed online in order to facilitate code exploration and analysis, and reflect the evolution of the project in a transparent manner.

Furthermore, we covered the platform code with unit tests from the very beginning. This ensured that the code was properly tested as it was developed and tests reflected the intention of the code. We also added to our methodology code coverage measuring tools, to detect pieces of code that were not covered by tests and we ensured that the code coverage was as complete as possible. This was useful to produce reliable functionality.

9.4 Laws and Regulations

This project collects personal data from users, and must therefore comply with the Spanish laws related to personal data collection, specifically the Organic Law 15/1999 of 13 December on the Protection of Personal Data (in Spanish, Ley Orgánica de Protección de Datos).

The Spanish organization in charge or personal data protection states the steps to be followed in order to legally collect personal data from our users [32].

First, we must take into account that the creation of a file containing personal data must be notified to the corresponding organization. This must be done before collecting any personal data, and is free of charge. The notification must include:
• Identification of the file and its purpose

• The source of data, specifying what people we intent in collecting data from

• The structure of the file, including detailed description of data specially protected and data that can identify the user

• The target beneficiary of the data collected

• The services that can be used to correct or erase collected data

• The minimum required security level

The organization describes three security levels, according to the typology of the data collected. In our case, since we collect data on the diagnosis of dyslexia which is considered a health matter, we must apply the maximum security level.

The Personal Data Protection Act establishes that the user must be informed whenever his personal data will be included in a file. This message given to the user must include the file to which his data will be incorporated, the use that will be made to this data, information on why it is being collected, information on what the consequences are if this data provided is not truthful and whether he will be able to modify and delete the data once it is collected.

We must also bear in mind that we might have users under the age of 14. In this case, the law states that the approval for collecting personal data must be given by the user’s parents.

Our project is clearly affected by these regulations, and we must comply with them before collecting any personal data. Given that the user will interact with the game platform, this system will be responsible for informing the user as stated by the law before collecting any personal data. This is done in the register page, and therefore we will have to include the necessary information and make it visible to the user. We should collect the user’s consent, or the user’s parents’ consent, before registering a user.

The law specifies that the user should be able to modify and delete the personal data we have on file. Therefore, we must extend our functionalities to include this capability. This has been done through the Delete user use case.

Given that the nature of the projects included in our platforms might lead to an end in the use of data, we must inform the user if the file will cease to include his personal data. This applies to our case when a project in the crowdsourcing platform is deleted.

Finally, we must remember our obligation to register the files containing personal data with the regulating organization before we start to collect data from our users. Given that we are storing data in two different platforms containing different levels of detail, we will have to register both platforms and ensure that they are clearly delimited.

Up to date, we have complied with these regulations almost entirely. We have informed users during registration of the data we collect and how it is used, providing a policy publicly available. Furthermore, we have allowed the user to delete his user, removing his personal data from the Game platform and removing all identification to his person from the data collected. We lack however the correct registry for the databases, which should be done in the near future when the platform end its beta testing and achieves a stable state.
9.5 Sustainability and Social Compromise

This project clearly has a strong social impact. The team behind Dyseggxia have been working for some time to help people, and specifically children, with dyslexia and other learning disabilities. By building the Game contained in this project, we are directly helping them to improve their research and therefore contributing to improve the possibilities for people with dyslexia.

The results derived from the use of our two systems combined will provide analytical data that can be studied and introduced into their application Dyseggxia with the aim of improving the effectiveness of their exercises on children with dyslexia. The team have already expressed their intentions to maintain the project alive and to keep improving it in the near future.

The system has been announced to several Dyseggxia users with the aim of user testing, and the feedback received is not only constructive but also very heart warming. Several users have messaged the author to express their gratitude for the work done in favor of their children and to praise the work done.

Furthermore, we have made a contribution to the Crowdsourcing Research Group at UPC, providing them with the base project for the generic crowdsourcing platform they aim to build. We have provided extensive documentation with explanations on the created platform to ensure it can be continued by others in the near future.

The project has also been carefully designed to make the lowest impact on the environment possible. By using cloud solutions to deployment instead of proprietary servers, we have reduced the amount of energy needed to keep the systems alive and the costs that energy consumption and hardware maintenance would carry.
10 Results

In this section we will present several topics that bring closure to our project. We will describe in a general manner the solution achieved, and how it meets the initial goals. We will talk about the feedback received from users, as well as present a small analysis made with the data collected during the last few weeks. Finally, we will present the general project conclusions and discuss future work.

10.1 Solution

This project has produced two related systems from analysis to specification, design and implementation. The two systems are aimed to fulfill the requirements set by two major stakeholders, the Crowdsourcing Research Group and the Dyseggxia team, but work together towards a common goal centered on data collection and processing.

Initially, we considered producing one system that contained both sides. Nevertheless, this option gave us much less flexibility and would have become a much more complex solution. In order to maintain the generalization desired from the point of view of crowdsourcing, we would have had to design a means to define interfaces for each project and include our game as a specific case of this design, which would have been very complex.

The Crowd platform produced fulfills the initial requirements and goals set by the Crowdsourcing Research group. It allows to manage projects, which are general enough to include our specific case or other cases considered during design such as image labeling. The functional requirements Introducing tasks, Obtaining results, Progress visualization, Execution control, Data visualization, Task providing and Batch deletion are covered by this system.

We therefore can say the initial goals "Provide a first approach to a generalized Crowdsourcing platform that allows the creation of other platforms from its structure by May 2013." and "Provide a tool that allows the Dyseggxia team to upload exercises and obtain statistical data about their performance with users by May 2013." have been met.

The Game produced fulfills the initial requirements and goals set by the Dyseggxia team, interested in a web game with a similar interface to Dyseggxia where users can try out their exercises. It allows users to solve all types of exercises included in Dyseggxia as well as to view their personal data provided, a legal requirement. The functional requirements Task solving, User data visualization and User deletion are covered by this system.

With this system, we can say the initial goal "Provide an engaging game that attracts users and encourages them to solve linguistic exercises by May 2013." has been met.

Finally, we have a fourth goal "Develop a functional system that can be presented as a thesis project by June 2013.". The project as a union of the two systems has met this goal, providing a full project for a thesis.

The combination of both platforms is a fully functional system that allows us to define exercises, execute them with users and save the data collected for later analysis. We have been able to collect real data for a few weeks and have proved that everything works as expected, providing a solution to the needs that fueled this project.

The Game produced is publicly available, and continues collecting data on exercise executions for analysis. It can be accessed through the following link:

http://mapuche.clarabayarri.com
10.2 User feedback

The final stage of our project, dedicated to user testing, has provided us with extremely valuable feedback. We have been able to improve our product as well as note future improvements that were not possible within our time restrictions but will be added in the future.

The first phase of user testing was done right after deployment, and consisted on handling our Game to a few controlled users. We sent the link to the users one by one, asked them to play for a while and received their feedback. Then, we corrected everything we could according to the feedback received and sent the link to the following user, therefore iterating and improving our interface and global user experience. We proceeded with this methodology eight times, testing with eight different users and received very heterogeneous feedback, from interface styling to exercise content.

Once this phase was complete, we proceeded by sending the link to small collectives such as other teachers in the supervisor's department and students. This generated a considerable amount of new users, and provided us with a different kind of feedback, related mostly to research and data processing.

Once we had more confidence on our system and had made the changes we considered a priority, we released the link to the Game on a social network. Given that social networks provide users during a limited time and then the link is lost, we repeatedly published the link to the Game along with an explanation of our project along several days. This allowed us to recruit a large number of users and start collecting real valuable data. At this point, the feedback we received was not only from users themselves, but from their executions. We made sure that the Game continued to work smoothly and that all data was correctly stored.

We would like to illustrate some comments received during the testing phases that have been constructive and have been included as improvements in the final project:

- Font and font size: the researcher in the Dyseggxia team informed us of new results achieved with her tests earlier this year on font impact on dyslexic users and we changed the game font to match the font that gave best accessibility to dyslexic users according to her research.

- Instruction colors: we were asked to distinguish exercises better as some people got confused, so we changed the background color of the instructions for each type of exercise as is done in Dyseggxia.

- Exercise skipping: users complained that when they did not know the answer to an exercise they wished to be able to skip instead of reloading the page. We added a skip button on the bottom right corner of the screen.

- Interaction bugs: some users found bugs in the interaction implementation that caused tiles to disappear. These were fixed.

Other comments that have been noted for future work include

- Improve the separation interaction by drawing the line that the user depicts to cut
- Allow the users to share score on Facebook
- Create a global scoreboard with high scores
- Change the cursor type to a hand when hovering over drawn buttons
- Improve animations
10.3 Data analysis

Even though data analysis is not strictly included in the goals of this project, we thought it would be convenient to briefly look at the data collected during the first few weeks and verify that everything is working as expected. The results obtained from this data analysis are not representative, as we have not collected enough data yet. However, these results will give us a first insight into the possibilities of our project.

First of all, we can take a look at the analysis done by the Crowd platform and represented in its project graphs. We see in figure 50 we have had a fairly constant participation rate, kept alive by our regular posts on social networks.

![Figure 50: Calendar with frequency of executions stored in the Crowd system. From lower to higher: purple, blue, green, orange, red.](image)

If we look at the graphs for the output variables in figures 51 and 52, we will see that all the numerical variables follow a similar distribution, high at low values and with some outlier higher values. This gives us a hint that when analyzing the data we might want to purge these values.

![Figure 51: Graph generated by the Crowd platform with the distribution of the time-spent output field.](image)

Finally, the word cloud produced from the String variable that represents the wrong answers given, shown in figure 52, gives us several clues on what words are most common errors and what future exercises we might want to consider.

We can also visualize this data at a batch level, but we believe it is more interesting to move on to data analysis.

The Dyseggxia project in the Crowd platform consists of four batches, each one with the exercises suggested for one particular level of difficulty. We have downloaded the data for the four batches and present here a simple overview.
Since the initial deployment of the game 4 weeks ago, we have achieved 114 registered users in the game, who have solved over 4500 exercises. We consider this an accomplishment, given that we did very little advertising of our game. Nevertheless, we must consider this is little data given the amount of exercises loaded on the platform. On average, each task has been executed 3 times, which means the data we have is extremely biased. Nevertheless, we would like to take a first look into what we have gathered.

We have cleaned the data to ensure the results were as accurate as possible. We have removed the executions made by the author, as there was certain advantage knowing how the game works and having seen the exercises. This data could distortion the real data, and therefore has been removed. We have also removed the executions from the first iteration of user testing, as we asked our users to test the game, find bugs and look for security flaws and therefore the data generated is not real. Finally, we have removed the executions of users from who we do not possess the dyslexic and spanishSpeaker attributes, as these are a key point in our data analysis.

A first overview of the cleaned data, shown in figure 54, confirmed us that, as foreseen, there are several outlier values in the timeSpent attribute, causing the third percentile and maximum value of...
this attribute to be very distant. We have taken a value fairly over the third percentile but much smaller
than the maximum value and have purged all executions with values over it.

Figure 54: First analysis of the collected data.

At this point, we are ready to take a deeper look into the data.

We clearly expect that, in general, non-dyslexic users take less time to solve exercises than dyslexic
users. We have done a simple analysis of the average time spent solving an exercise per user, and
plotted the distribution for dyslexic and non-dyslexic users as seen in figure 56.

As expected, the majority of dyslexic users produce a higher average execution solving time than
non-dyslexic users.
Finally, we wanted to take a first approach to the final goal of the Dyseggxia pipeline, to rank exercises by their usefulness for dyslexic people. Even though a real analysis would be very complex, we have taken a first approach by comparing the execution time of each task for dyslexic and non-dyslexic users. For this, we have calculated the average time spent for each task id and value of dyslexic, discarded the tasks that did not have data regarding both user bases and then plotted the obtained values as seen in figure 57. Each point in the graph is an exercise, its horizontal value indicates the average time it takes to complete that exercise for a dyslexic user and the vertical value indicates the average time it takes for a non-dyslexic user. We notice that there are some exercises that remain near the diagonal and are therefore equally long to solve for dyslexic and non-dyslexic users. However, exercises that appear in the lower right corner of the plot tell us they are fast for non-dyslexic users and slow to solve for dyslexic users. These are the exercises we are most interested in, as it seems they pose a bigger problem to dyslexic users.

In order to take a better look, we can make this analysis by batch, but given the current amount of data we believe it will yield non-significative results.

This is just a first approach to data analysis, and we hope to have managed to get across the meaningfulness of the data we are collecting and the great potential it has for further analysis and support to research. The R script used to analyze the data and obtain the plots here presented can be found in the appendix section 12.4.

**Figure 56**: Distribution of the `timeSpent` attribute for dyslexic and non-dyslexic users.
Figure 57: Distribution of the exercises according to the average timeSpent value for dyslexic and non-dyslexic users.
10.4 Conclusions

This project presents the creation of two interrelated systems that work together to collect data on exercise executions, contributing to the research on technology for dyslexia and crowdsourcing.

First, in section 1 we have introduced the project topic and have explained in detail the background project Dyseggxia. Then, we have analyzed all the influences on the project, including the problems to be solved, the requirements set by the stakeholders and the goals to be met in section 2. Once this was settled, we have proceeded in section 3 to specify the functionality that our project will include, separated in two fully functional systems: a crowd platform and a game. In section 4 we have introduced the general design of the project, to then follow in sections 5 and 6 with the design of the two systems, including specific component behavior and interaction. We have then explained the current deployment for both systems in section 7, and rounded up the work done by describing testing on both systems in section 8.

At a less technical level, we have explained the general management of the project in section 9, discussing the planning and budget as well as the changes they have suffered since the beginning of the project. We have also seen the methodology followed, laws and regulations that affect the project and the social compromise taken. Finally, we have analyzed the results of our work and have concluded with section 10.

A comprehensive list of references can be found in section 11, and additional documents in the appendix in section 12.

During the development of this project we have reached some insights we would like to present.

First of all, we consider we have learnt very valuable lessons at many levels.

At a management level, we have learnt to deal with a big software project, trying to foresee problems before they occur and keeping up with a schedule. We have had requirement changes along the way, and have negotiated and incorporated those we thought appropriate into our planning. For example, we started the project with 5 types of exercises to be displayed, and halfway through the project the Dyseggxia team came up with new types of exercises we incorporated and designed the interaction for. Furthermore, we took care of legal requirements we weren’t aware of at the beginning of the project, such as policies and personal data deletion. We even managed to include a new iteration that implied migrating the data layer from one of the systems, all within the planned schedule.

At a software engineering level, we have seen the whole software engineering process working, from analysis to implementation, a first for the author. Along the bachelor degree, we see all the parts in the process, but we are not able to properly combine them into a single process until a project of this size is done.

At a technical level, we have learnt about many new technologies and have adapted our work to them. The author was not familiar with Spring, or web applications, at the beginning of the project. Similarly, we learnt first-hand about database solutions and their limits, and how to choose the appropriate solution for each case. We even took the challenge of implementing Map Reduce processing simply to learn and see how it can be done.

There are some functionalities in this project which simply respond to the urge to learn from the author, and have proven to be useful tools for learning such as email sending or Map Reduce processing.

Furthermore, we have learnt to integrate our work with other existing platforms, such as Google Fusion Tables [20] or the d3 [19] graph rendering library.

In second place, we can consider conclusions related to data. We have achieved some preliminary results for our purpose of data gathering, analyzed them and obtained a good insight into what is
waiting ahead of us. The data retrieved up to date seems encouraging, and we hope to be able to extract many results from future data analysis. For now, the Dyseggxia team has considered our results to be useful in their research, and therefore validated the work done.

Similarly, users seem to be happy with the Game, as we receive positive feedback and they keep on playing.

In conclusion, we have developed a project capable of helping research in the field of dyslexia and in the field of crowdsourcing, providing indirectly a high social impact, and have courage to keep on working to further improve the tools we can provide these two fields with better tools in the near future.
10.5 Future work

The project here presented has met its goals and has produced two functional systems as required. Nevertheless, there is still work that can be done on both systems for future work, which would improve their capabilities.

First of all, the Crowd system has been designed as a first step towards a generic crowdsourcing platform given a first concrete case, our need to collect data for Dyeggxia. The system has fulfilled the needs for this first use case, but has left functionality that was not explicitly needed or that was not even discovered pending. By constructing other use cases to be included in the Crowd platform, we will be able to extend its functionality and generalize it better to crowdsourcing problems. With this, topics covered by the crowdsourcing taxonomies state of the art that were not necessary for our project such as user remuneration would be covered.

Additionally, with more time we could take more interest in the project deployment, migrating it to more flexible services such as Amazon. This requires some work configuring the environment and setting up the necessary dependencies which was done for us automatically by Heroku, but will increase the flexibility and autonomy of the project. By migrating to a more flexible system, we will also be liberated from restrictions given by Heroku's add ons on price, and will be able to grow better at a smaller price.

We should also consider future work involving the database. At this point, we have collected a considerable amount of data and the system has worked well. Nevertheless, with time we will be able to tune the system to better perform given the data we have, once this data is a larger amount.

Another area of work could be the currently defined API. With more projects using our crowd platform, we could discover new needs such as requesting tasks from a specific batch and therefore extend the current API.

Secondly, we can look at the Game system. The Game system has been produced to cover the basic functionality necessary to execute exercises as needed and cover legal restrictions. With more time, we can extend the Game to include engaging functionality, which would take care of user recruiting for us. A few examples of this possible work include integrating with social networks, creating leader boards, creating levels and achievements and other similar functionalities.

The Game interface can be improved at many levels, studying the interaction done by users. The current interface gives a basic interaction, but may lead to unnecessary errors that can be addressed with more time.

We could also extend the Game to allow users to suggest exercises. With this, we could include them in the Crowd platform project and obtain execution data, providing us with a new source of input. Given that users who relate to this project are mostly willing to help with research, we believe this could be a great opportunity to collect suggestions, for example from parents who observe the errors committed by their children when writing.
11 References


12 Appendix

12.1 Database schema for the Game system

Table "gameuser"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>character varying(255)</td>
<td>not null</td>
</tr>
<tr>
<td>age</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>dyslexic</td>
<td>boolean</td>
<td>not null</td>
</tr>
<tr>
<td>email</td>
<td>character varying(255)</td>
<td></td>
</tr>
<tr>
<td>isspanishspeaker</td>
<td>boolean</td>
<td>not null</td>
</tr>
<tr>
<td>password</td>
<td>character varying(255)</td>
<td></td>
</tr>
<tr>
<td>platformid</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>score</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>passwordresetrequest_id</td>
<td>bigint</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:

- "gameuser_pkey" PRIMARY KEY, btree (username)

Foreign-key constraints:

- FOREIGN KEY (passwordresetrequest_id) REFERENCES passwordresetrequest(id)

Referenced by:

- TABLE "gameuser_roles" FOREIGN KEY (gameuser_username) REFERENCES gameuser(username)

Table "gameuser_roles"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>gameuser_username</td>
<td>character varying(255)</td>
<td>not null</td>
</tr>
<tr>
<td>roles</td>
<td>character varying(255)</td>
<td></td>
</tr>
</tbody>
</table>

Foreign-key constraints:

- FOREIGN KEY (gameuser_username) REFERENCES gameuser(username)
Table "passwordresetrequest"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>bigint</td>
<td>not null</td>
</tr>
<tr>
<td>generationdate</td>
<td>timestamp without time zone</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:

- "passwordresetrequest_pkey" PRIMARY KEY, btree (id)

Referenced by:

- TABLE "gameuser" FOREIGN KEY (passwordresetrequest_id) REFERENCES passwordresetrequest(id)

Table "platformdata"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>uid</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>projectid</td>
<td>character varying(255)</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:

- "platformdata_pkey" PRIMARY KEY, btree (id)

Table "problem"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>batchid</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>display</td>
<td>character varying(255)</td>
<td></td>
</tr>
<tr>
<td>taskid</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>character varying(255)</td>
<td></td>
</tr>
<tr>
<td>word</td>
<td>character varying(255)</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:

- "problem_pkey" PRIMARY KEY, btree (id)
Referenced by:

- TABLE "problemcollection_problem" FOREIGN KEY (problems_id) REFERENCES problem(id)
- TABLE "problem_answers" FOREIGN KEY (problem_id) REFERENCES problem(id)
- TABLE "problem_displaytext" FOREIGN KEY (problem_id) REFERENCES problem(id)

**Table "problem_answers"**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem_id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>answers</td>
<td>character varying(255)</td>
<td></td>
</tr>
<tr>
<td>position</td>
<td>integer</td>
<td>not null</td>
</tr>
</tbody>
</table>

Indexes:

- "problem_answers_pkey" PRIMARY KEY, btree (problem_id, "position")

Foreign-key constraints:

- FOREIGN KEY (problem_id) REFERENCES problem(id)

**Table "problem_displaytext"**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem_id</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>displaytext</td>
<td>character varying(255)</td>
<td></td>
</tr>
<tr>
<td>position</td>
<td>integer</td>
<td>not null</td>
</tr>
</tbody>
</table>

Indexes:

- "problem_displaytext_pkey" PRIMARY KEY, btree (problem_id, "position")

Foreign-key constraints:

- FOREIGN KEY (problem_id) REFERENCES problem(id)

**Table "problemcollection"**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>internalid</td>
<td>integer</td>
<td>not null</td>
</tr>
</tbody>
</table>

Indexes:
- "problemcollection_pkey" PRIMARY KEY, btree (internalid)

Referenced by:

- TABLE "problemcollection_problem" FOREIGN KEY (problemcollection_internalid) REFERENCES problemcollection(internalid)

Table "problemcollection_problem"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>problemcollection_internalid</td>
<td>integer</td>
<td>not null</td>
</tr>
<tr>
<td>problems_id</td>
<td>integer</td>
<td>not null</td>
</tr>
</tbody>
</table>

Indexes:

- "problemcollection_problem_pkey" PRIMARY KEY, btree (problemcollection_internalid, problems_id)
- "problemcollection_problem_problems_id_key" UNIQUE CONSTRAINT, btree (problems_id)

Foreign-key constraints:

- FOREIGN KEY (problemcollection_internalid) REFERENCES problemcollection(internalid)
- FOREIGN KEY (problems_id) REFERENCES problem(id)
12.2 Gantt Diagrams

12.2.1 Initial planning
12.2.3 Final planning
12.3 Initial budget calculation

<table>
<thead>
<tr>
<th>Resource</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project manager</td>
<td>40 €/h</td>
</tr>
<tr>
<td>Analyst &amp; Designer</td>
<td>40 €/h</td>
</tr>
<tr>
<td>Developer &amp; Tester</td>
<td>20 €/h</td>
</tr>
</tbody>
</table>

We consider our working days to be 4 hours long, since the author of this project is also taking other courses and will not be able to work full time on the project. With this data, we can now estimate the cost of the tasks previously stated in the project planning.

<table>
<thead>
<tr>
<th>Task name</th>
<th>Duration</th>
<th>Resource type</th>
<th>Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements definition</td>
<td>5d</td>
<td>Manager</td>
<td>800</td>
</tr>
<tr>
<td>Project scope definition</td>
<td>3d</td>
<td>Manager</td>
<td>480</td>
</tr>
<tr>
<td>Use case definition</td>
<td>3d</td>
<td>Analyst</td>
<td>480</td>
</tr>
<tr>
<td>Project modelling</td>
<td>4d</td>
<td>Designer</td>
<td>640</td>
</tr>
<tr>
<td>Project management course</td>
<td>14d</td>
<td>Manager</td>
<td>2240</td>
</tr>
<tr>
<td>Project setup</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td><strong>System core</strong></td>
<td></td>
<td></td>
<td>1040</td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Game simple task execution</td>
<td>4d</td>
<td>Developer</td>
<td>320</td>
</tr>
<tr>
<td>Serve task to game</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Save results to platform</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Show batches on platform</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td><strong>System core improvements</strong></td>
<td></td>
<td></td>
<td>1200</td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Improve task interface</td>
<td>4d</td>
<td>Developer</td>
<td>320</td>
</tr>
<tr>
<td>Extend to 5 types of tasks</td>
<td>3d</td>
<td>Developer</td>
<td>240</td>
</tr>
<tr>
<td>Rework task interface</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Task name</td>
<td>Duration</td>
<td>Resource type</td>
<td>Cost (€)</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Tasks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Start / stop batches</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Create batches</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Results</td>
<td></td>
<td></td>
<td>1120</td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Download raw results</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Show results online</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Manipulate results online</td>
<td>6d</td>
<td>Developer</td>
<td>480</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>User management</td>
<td></td>
<td></td>
<td>800</td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Game login / logout</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Game registration</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Platform login / logout</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>User data</td>
<td></td>
<td></td>
<td>960</td>
</tr>
<tr>
<td>Analysis &amp; design</td>
<td>2d</td>
<td>Analyst/Designer</td>
<td>320</td>
</tr>
<tr>
<td>Data definition</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Update data</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Show user data on game</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Link account with Facebook</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Share score on Facebook</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Refactor &amp; debug</td>
<td>2d</td>
<td>Developer</td>
<td>160</td>
</tr>
<tr>
<td>Main refactor and debugging</td>
<td>4d</td>
<td>Developer</td>
<td>320</td>
</tr>
<tr>
<td>System deployment</td>
<td>1d</td>
<td>Developer</td>
<td>80</td>
</tr>
<tr>
<td>Write project document</td>
<td>10d</td>
<td>Manager</td>
<td>1600</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>108d</strong></td>
<td></td>
<td><strong>12,720</strong></td>
</tr>
</tbody>
</table>
### 12.4 Data analysis script

```r
# Data analysis script

dades1 <- read.csv("/Users/clara/Dropbox/tfg/data/batch-level-1.csv", header = TRUE, dec = ",")
dades2 <- read.csv("/Users/clara/Dropbox/tfg/data/batch-level-2.csv", header = TRUE, dec = ",")
dades3 <- read.csv("/Users/clara/Dropbox/tfg/data/batch-level-3.csv", header = TRUE, dec = ",")
dades4 <- read.csv("/Users/clara/Dropbox/tfg/data/batch-level-4.csv", header = TRUE, dec = ",")
dades <- rbind(dades1, dades2, dades3, dades4)
summary(dades)

# remove executions made by the author

table(dades[, 'userId'] == 3)
dades <- dades[dades[, 'userId'] != 3 & dades[, 'userId'] != 18,]
table(dades[, 'userId'] == 3)

# remove the executions done during the first testing phase

table(dades[, 'date'] > as.Date("2013-05-27"))
dades <- dades[dades[, 'date'] > as.Date("2013-05-27"),]
table(dades[, 'date'] > as.Date("2013-05-27"))

# remove timeSpent outliers as they distortionate the plot

table(dades[, 'timeSpent'] <= 30000)
dades <- dades[dades[, 'timeSpent'] <= 30000,]
table(dades[, 'timeSpent'] <= 30000)

# remove rows with NA value in either dyslexic or spanishSpeaker

table(dades[, 'dyslexic'] != "")
dades <- dades[is.na(dades[, 'dyslexic']) & !is.na(dades[, 'spanishSpeaker']) &
dades[, 'dyslexic'] != "" & dades[, 'spanishSpeaker'] != ","]
summary(dades)

# separate data by dyslexic

dades.dyslexic <- dades[dades[, 'dyslexic'] == 'true',]
dades.notDyslexic <- dades[dades[, 'dyslexic'] == 'false',]

library(plyr)

# time by user plot

aggr.time.dyslexic <- ddply(dades.dyslexic, .(userId), summarise,
mean.time.dys=mean(timeSpent))
aggr.time.notDyslexic <- ddply(dades.notDyslexic, .(userId), summarise,
mean.time.notdys=mean(timeSpent))

density.time.dyslexic <- density(aggr.time.dyslexic$mean.time.dys)
density.time.notDyslexic <- density(aggr.time.notDyslexic$mean.time.notdys)

plot(density.time.notDyslexic, col='blue', main="Density distribution average timeSpent attribute by user")
lines(density.time.dyslexic, col='red')
```

255
legend(x="topright", legend=c("Non dyslexic","Dyslexic"), lty=c(1,1),
  lwd=c(2.5,2.5), text.width = strwidth("2,000,000"), col=c('blue','red'), cex=0.8)

# time spent comparison plot
aggr.dyslexic <- ddply(dades.dyslexic, .(task_id), summarise, mean.dys=mean(timeSpent))
aggr.notDyslexic <- ddply(dades.notDyslexic, .(task_id), summarise,
  mean.notdys=mean(timeSpent))

# keep only those tasks we have on both sides
aggr.dyslexic <- aggr.dyslexic[aggr.dyslexic$task_id %in% aggr.notDyslexic$task_id,]
aggr.notDyslexic <- aggr.notDyslexic[aggr.notDyslexic$task_id %in% aggr.dyslexic$task_id,]

# combine results
results = merge(x = aggr.dyslexic, y = aggr.notDyslexic, by = "task_id", all = TRUE)

maxvalue <- max(c(max(results$mean.dys), max(results$mean.notdys)))

# Resulting plot
plot(results$mean.dys, results$mean.notdys, xlim=c(0, maxvalue), ylim=c(0, maxvalue),main="Tasks by average timeSpent",xlab="mean timeSpent for dyslexic users", ylab="mean timeSpent for non-dyslexic users")