Reusable web hosting control panel interface

Carlos Acedo Rodríguez
Universitat Politècnica de Catalunya (UPC)

A thesis submitted for the degree of
Diploma in Computer Software

June 2012
Abstract

Communication is part of who we are, as individuals and as a group, the Internet has already changed the way we communicate, we have tools to access and send information everywhere and every time. Web and mail are still the most common communication technologies on the Internet, unfortunately not everyone possesses the knowledge to effectively use those technologies, and this is why many companies and organizations offer their help to their users, who otherwise, couldn’t access those technologies. This help often comes in form of a web control panel, which allows the user to manage and setup his web sites, mail accounts and other related tools.

This thesis introduces a new web hosting control panel interface, different from any other because of its simple to use nature, allowing users with non technical knowledge to accomplish typical web hosting tasks, which otherwise, would be difficult to achieve.

The project has been sponsored by Pangea, a non profit organization from Barcelona.

The main features of this hosting control panel includes personal data management, mail management, web management, database management and billing overview.

A research of how users interact with machines, mainly computers, has been made in order to present an intuitive Graphical User Interface (GUI). To allow a similar user experience, users with slight physical disability, like color blindness or partial blindness has been taken into account at designing time.

Portable devices like smartphones and tablets use are quickly increasing, an effort has been made to include a suitable access using dynamic layouts and limiting UI controls size.
1 Introduction
  1.1 Overview ......................................................... 1
  1.2 Motivation ..................................................... 2
    1.2.1 Pangea ..................................................... 2
    1.2.2 Necessity for a web hosting control panel ............... 3
    1.2.3 Goals ..................................................... 4

2 Background and related work ........................................ 5
  2.1 The psychology of user interface design ......................... 5
  2.2 Usability ....................................................... 5
  2.3 Goal oriented design ........................................... 6
    2.3.1 Design process ............................................. 6
    2.3.2 Personas ................................................... 7
      2.3.2.1 Pangea’s persona ..................................... 7
    2.3.3 Goals ..................................................... 8
  2.4 Related solutions ............................................... 9
    2.4.1 Conclusions ............................................... 11
  2.5 Relationship with previous project ............................. 11

3 Design and implementation ...................................... 13
  3.1 Overview ....................................................... 13
    3.1.1 Resources and tools ..................................... 14
  3.2 User interface design ......................................... 15
    3.2.1 Widget design ............................................. 17
    3.2.2 Dynamic layout .......................................... 18
  3.3 Categories ..................................................... 20
    3.3.1 Home ..................................................... 20
    3.3.2 Mail ..................................................... 21
List of Figures

1.1 Pangea logo ................................................................. 2
1.2 Pangea’s current control panel ................................. 3

2.1 Goal oriented process overview ................................. 6
2.2 Home screen of ISPConfig. Note the redundancy of menu items and unnecessary “Latest news” information. ................................. 9
2.3 Home screen of GNUPanel. Gives some information about the control panel and the author, most of the space is wasted. ................................. 10
2.4 Start page of Hepsia Control Panel ................................. 11

3.1 Gantt diagram of the project timeline ................................. 14
3.2 Home screen mockup ................................................................. 15
3.3 Current implemented home screen ................................................................. 16
3.4 Aliases widget title ................................................................. 18
3.5 Aliases widget ................................................................. 18
3.6 Home screen at 1920x1080 pixels ................................................................. 19
3.7 Home screen at 800x600 pixels ................................................................. 19
3.8 Traditional model and AJAX model ................................. 25
3.9 Store add function sequence diagram ................................. 27
3.10 Store remove function sequence diagram ................................. 27
3.11 View, storage and REST server communication ................................. 28
3.12 View creation sequence diagram ................................................................. 29
3.13 Files and directories tree ................................................................. 30

4.1 Color blind filter ................................................................. 55
List of Tables

3.1 REST Application Programming Interface . . . . . . . . . . . . . . . 37
Chapter 1

Introduction

1.1 Overview

With more than 2.4 billion users worldwide\footnote{http://www.itu.int/ITU-D/ict/statistics/} connected to the Internet, the need to have presence online is growing fast, besides social networks, web and mail are still the more common communication technologies.

Despite those technologies being so common, they need specialised knowledge and resources, for instance, to proper serve a web site, it is needed a high speed internet uplink connection, a computer running 24h a day, a web server, a DNS server and the knowledge to put up this together.

Many individuals or entities don’t have the knowledge, time or just the economic resources to setup a web or mail server, this is where hosting providers and web hosting control panels become useful.

A web hosting control panel (from now on control panel), or Internet hosting control panel, is a web interface which allows its users to manage and control their hosted services. It is commonly offered by hosting companies, and usually offers mail, web, ftp and database management, quota (as in space and bandwidth usage) information, software installation, statistics and so forth.

This project is an analysis, design and implementation of a web hosting control panel. It is different from any other web hosting control panel available as today on the Internet because it is focused on simplicity, visual appealing and task oriented.

One problem of web interfaces is its need to load the entire page when just a part of the content changes, despite this being solved by the introduction of XMLHttpRequest API in 2006\footnote{http://en.wikipedia.org/wiki/XMLHttpRequest#History_and_support}, only a few sites profit from this API. AJAX, which uses the XMLHttpRequest API, allows a web page to communicate asynchronously with
a server, allowing a web to never reload completely and offering more desktop application experience.

The client side uses the Dojo toolkit, which is a set of JavaScript libraries which eases the use of AJAX, providing widgets, data models and much more, allowing to develop a modular and scalable control panel. The control panel will interact with a RESTful server through a well defined API using JavaScript Object Notation (JSON) for serializing data.

The API has been designed to be simple and open, REST software architecture is both, platform and language independent, thus allowing any web hosting software to use this control panel as a frontend.

A backend has been implemented in Python as an example of use of this API. Work has already been started to allow this control panel to be used as a frontend of a previous project by Universitat Politècnica de Catalunya (UPC) student Marc Aymerich, which implements a sophisticated web hosting control panel.

1.2 Motivation

1.2.1 Pangea

To understand better the motivation of this project, the organization which I work for needs an introduction. Pangea\footnote{http://pangea.org} is a non profit and non governmental organization (NGO) from Barcelona, Spain. Since its beginnings in 1993, its main purpose has been to help individuals and organizations on their work on social rights, education, ecology, etc. As experts in information technology, Pangea offers services like web hosting, mail, newsletters, calendars.

Although most of its members are from Catalonia, it works with international organizations like Association for Progressive Communications (APC)\footnote{http://apc.org} or European
CHAPTER 1. INTRODUCTION

Digital Rights (EDRI) Promotes open source software and net neutrality.

Pangea’s members pay an annual subscription to cover the infrastructure expenses, like high speed internet, servers, co-location and technical and administrative personnel. Because of the expertise working with those kind of organizations and the size of Pangea, members find a dedicated support not found in any hosting company.

With currently more than 700 member, their backgrounds are very diverse, from different ages, social positions or education levels.

My role at Pangea, alongside with my co-worker Marc is to maintain and improve the technical infrastructure. Currently, Pangea offers its services on a virtualized environment from a co-location center in Barcelona.

1.2.2 Necessity for a web hosting control panel

Pangea’s members have already a control panel, it is part of a previous project and was patched by the technical team to add some functionalities. Despite the added functionalities, many user would find it very basic and visually poor.

Figure 1.2: Pangea’s current control panel

http://edri.org
CHAPTER 1. INTRODUCTION

Pangea is working on the migration to another control panel designed and developed by Marc Aymeric, presented on fall 2011 as his thesis for UPC. This control panel is more oriented to advanced users, here the need for a usable and accessible control panel for Pangea’s members.

1.2.3 Goals

As stated in section 1.2.1 Pangea’s members have very different levels of technical knowledge, therefore the design should satisfy these conditions:

- **Intuitive user interface (UI).** Users should find a “natural” way to do things, this means things are where the user is looking for.

- **Helpful.** The interface should have context help without being intrusive.

- **Accessible.** When possible, the interface design should take into account some disabilities like partial blindness or color blindness.

- **Visually appealing.** Users are more comfortable when the visual design has been taken care of.

Other goals more technical related should be:

- **Backend detachment.** Dispose of a logical and documented API for the backend communication.

- **Extendible.** It should be easy to add new components in case of necessity.

- **Scalable.** It should allow the use of scalability solutions.

- **Dynamic layout.** Displayed components should adapt to different screen resolutions.
Chapter 2

Background and related work

2.1 The psychology of user interface design

User interface design is a subset of the human-computer interaction (HCI), which is the study, planning and design of how people and computers work together, so that a person’s, or user, needs and expectations are satisfied in the most effective way.

The user interface is the part of computer interaction that the user can see, hear, touch, talk to, understand or direct. It is really important to understand that, from the user’s point of view, all the application is the user interface.

HCI designers must consider what limitations and abilities the users possess, what are their needs and expectations, how they perceive and process information in their workflow, and what they find attractive and enjoyable. It is, as well, the HCI designer job to know the hardware and software characteristics and limitations.

The most effective user interface is one that is not noticeable, which allows the user to focus on the performing task instead of the mechanics of the user interface.

2.2 Usability

Usability refers to the ease of use and the ability of learning of any human made object, being the object, in this case, a web application. Some key points of usability are:

- **Learnability.** How easy is for users to accomplish basic tasks?
- **Efficiency.** How quickly users can perform their tasks?
- **Memorability.** After a period without using the interface, are they capable of performing efficiently?. Do they need to use short time memory?
• **Errors.** Do error messages make sense to users? Can they act upon those errors?

• **Satisfaction.** How pleasant is to use the interface?

## 2.3 Goal oriented design

Every design requires to know the user profile, a research about their lives, motivations and environs will give a lot of data, but will not be useful by itself, there is a need for a **model**. Models are a concept widely use in science, it represent a complex phenomena with a useful abstraction, in this case, a user model is needed and it is commonly referred as **personas**.

### 2.3.1 Design process

A goal oriented design involves techniques of ethnography, user interviews, market research, user models, scenario design and interaction patterns.

<table>
<thead>
<tr>
<th>Research</th>
<th>Modelling</th>
<th>Requirements</th>
<th>Framework</th>
<th>Refinement</th>
<th>Support</th>
</tr>
</thead>
</table>

Figure 2.1: Goal oriented process overview

The **research** phase implies an ethnographic study to provide data of the potential product users, market research, competitive products audits and interviews with users, developers and experts on the subject matter. The objective of this phase is the awareness of user behaviour and workflow patterns.

The **modelling** phase is where an user model is created with the research input data, domain models may be created with information flow and workflow diagrams.

Next phase is the **requirements** definition, which connects the user model with the framework, focusing on facilitating user’s goals through defined tasks.

The **framework** phase defines the user interaction, and propose one or more visual designs. It is important to follow general design patterns.

The **refinement** part is where details about visual design and implementation such as visual coherence and task walkthroughs (key paths).

At the end of the process, developers are in situations where no design has been defined, deadlines may force them to make design by them selves, designer **support** has to be provided in those situations.
2.3.2 Personas

The first thing to define a persona is to choose the individuals which best represent the needs of the group, prioritizing their needs but, as much as possible, without compromising the ability to meet the needs of a secondary group of users. This will at least resolve 3 development issues:

1. **The elastic user.** In any development team, each member may have a different view of the users, ending up with an inconsistent product.

2. **Self-referential design.** Developers reflect their own motivations, skills and goals on the design. They understand perfectly how the data is structured and the workflow and are comfortable with the product, but many users may not find the design adequate.

3. **Edge cases.** There are situations that may happen, but usually will not for the target personas. Edge cases have to be accounted for, but never be the design focus.

2.3.2.1 Pangea’s persona

Data gathering for defining personas implies interview and workflow observation of a large group of representative users, unfortunately there is nor the time or the resources to do so, but my experience (more than 4 years interacting with Pangea’s members) and that one of my co-workers, lets me answer on an accurate enough fashion key topics to define the persona:

1. **Gender, occupation and age.** Both genders are equally represented, worker or volunteer with mid level education would be the most representative, with about one third of high level studies, age range from 25 to 70 with the most representative range from 35 to 55.

2. **Cultural background.** Most of the members are from Catalonia, about a 15 per cent from other Spanish regions. Most used language would be Catalan, with Spanish as a second language, a small group with Spanish as first language with no knowledge of Catalan, 70 per cent have basic knowledge of English. A very small group use Galician and Basque as their native language with Spanish as a second language and no knowledge of Catalan.

   Connection times are from 9h to 18h Central European Time (CET)
CHAPTER 2. BACKGROUND AND RELATED WORK

3. **Skills.** Most of them have from low to average computer user skills. Basic internet knowledge. About 50 per cent has attended specific courses like mail client software usage. Most of them are self-taught learners.

4. **Impairment.** The majority does not have difficulties of sight or reading, but a very small group need some special needs

5. **Personal achievements.** They are workers and volunteers, their goal is to contribute on a cause they believe in, a small part work for revenue and are not so attached.

6. **Motivation towards this web application.** Publish content online and manage their mail accounts are the main goals. Get information about the status of their services would be a secondary goal, as well as administrative information.

7. **Personal priorities and expectations.** They expect content in their native language, ideology neutral, simple terminology, simplicity over number of features, context help, those are common desirable priorities and expectations.

8. **Previous experience.** Most of them don’t have previous experience with web hosting control panels.

9. **Training.** They will not receive any training, only support by mail, telephone or in person if it is needed

10. **Usage frequency.** Web application usage will be irregular, with an average of twice per month.

11. **Origin.** They will access to the panel directly by manually input the web address or by a link on Pangea’s website.

2.3.3 Goals

Personas provide a context for groups of behaviours, goals are the drivers behind those behaviours. Goals are indicators for the designer of the functions of the product. The function and behaviour of the product must be achieved through tasks, which are the steps necessary to those goals.

Goals, not tasks, are the key to a product success.
2.4 Related solutions

In order to properly evaluate any existing solutions, a set of desired characteristics have to be defined, goals in section 1.2.3 and having an open source software license, are used as an evaluation criteria.

Usually hosting control panels have three types of users, administrators, reseller and client, client type. The client type is used if available.

The following are the web hosting panels found worth mentioning, they are presented in order of relevance:

1. **ISPConfig**
   - Developed by projektfarm, offers support for some linux distributions, is multi-server and is written in PHP.
   - **Pros.** Clean user interface, multi-language (although does not support Catalan), open source license (BSD), easy to learn.
   - **Cons.** No context help, not very accessible as it uses grey fonts over white, frontend and backend tightly integrated, no support for low resolutions.

![Figure 2.2: Home screen of ISPConfig. Note the redundancy of menu items and unnecessary “Latest news” information.](http://www.ispconfig.org/)

![Figure 2.2: Home screen of ISPConfig. Note the redundancy of menu items and unnecessary “Latest news” information.](http://www.projektfarm.de/)

![Figure 2.2: Home screen of ISPConfig. Note the redundancy of menu items and unnecessary “Latest news” information.](http://www.opensource.org/licenses/bsd-license.php)
2. GNUPanel

GNUPanel was founded and is currently developed mainly by Ricardo Marcelo Alvarez, despite the GNU[5] prefix GNUPanel is not part of the GNU project. Supports the management of only one server.

**Pros.** Context help, clean user interface, multi-language (although does not support Catalan), open source license (GPL[6]).

**Cons.** Very small fonts, frontend and backend tightly integrated, no support for low resolutions, context help usually far from the user visual focus.

![Figure 2.3: Home screen of GNUPanel. Gives some information about the control panel and the author, most of the space is wasted.](image)

3. Hepsia Control Panel[7]

This commercial project does not fall in the category a final user control panel, but it is worth mentioning because of the effort made on easing the use by choosing a logical ordering of sections, elaborated documentation (including audio) and visual appealing.

**Pros.** The home page gives quick access to most used sections with intuitive icons. It is visually attractive and very documented.
Cons. Being full featured, its user target requires a high level of technical knowledge, intrusive help. Every section has an intrusive popup with the documentation. Private license. Total integration of frontend and backend.

Figure 2.4: Start page of Hepsia Control Panel

2.4.1 Conclusions

None of the analysed 2.4 or any other web hosting control panel tried in the research has any detachment of the frontend from the backend, making impossible to reuse its interface with any other hosting solution. Since hosting server configurations can vary one from each other, many times it is difficult to adopt any existing solutions.

Many details show user interface design has not been studied enough, leading to usability problems and a poor user experience.

Support for touch screens or low resolution displays are non-existent as well.

2.5 Relationship with previous project

Marc designed and developed a full featured administrative web hosting control panel named UCP for Pangea as his thesis for UPC on fall of 2011. It is written in Python using the Django web framework. For more information, the paper and the code can be found at [http://ucp.pangea.org/svn/ucp/trunk/](http://ucp.pangea.org/svn/ucp/trunk/)

[https://www.djangoproject.com/](https://www.djangoproject.com/)
The target audience of UCP are Pangea’s employees and advanced users who are familiar with technical web hosting terminology and its software performance.

Many Pangea’s members, however, doesn’t have the knowledge level needed to feel comfortable using the UCP interface. This project will be used, in a future work, as an “user friendly” interface for UCP. Work for supporting a RESTful architecture in UCP is being done as the writing of this thesis.
Chapter 3
Design and implementation

3.1 Overview

This chapter introduces the development process, the tools and frameworks involved in the project.

This thesis organization can be explained in 4 phases:

1. **Documentation.** One of the main concerns in this thesis is usability applied to user interface design. Matters like user interface design, usability and user psychology seem topics for a successful web application, other topics were modern web technologies like HTML5, AJAX, CSS3 and web services. Details about the consulted books can be found in the bibliography section. The time spent in this phase was about 2 months. It is unavoidable, though, reading documentation while performing other thesis phases.

2. **Research.** This phase sets the bases for the following phases, it involves three types of research:

   - **User research.** A Research about the people that will become user, mandatory to define user models.
   - **User interface research.** An Evaluation of already existing graphical user interfaces, not only web, but from desktop and mobile devices, usability is a special concern on mobile devices. Apple\(^1\) and Google\(^2\) have style guides for their respective embedded platforms.

---


• Development tools. A research of the more convenient development tools, like editors, revision control system or frameworks.

This phase took about one month.

3. Design. From all the knowledge collected from the documentation and research phases, needed designs are made, these include user interface design, web model design and API design.

The time needed for this phase was about one month.

4. Development. In this phase the development platform is setup, familiarization and testing the newly learned technologies is done and all the needed code is written and tested. This is by far the most time consuming phase, taking about five months of the thesis time.

5. Paper. This phase includes the writing of this thesis documenting all the work done for the project. Time needed for this phase is one month.

<table>
<thead>
<tr>
<th>2011</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep</td>
<td>Oct</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.1: Gantt diagram of the project timeline

3.1.1 Resources and tools

As for the hardware resources used, three computers, one laptop and two desktop computers were used as both workstations and servers. A BlackBerry Playbook tablet was used as a client for testing layouts and platform support.

For the software tools, the following were used:

• General purpose and latex editors. VIM 7.3, Gedit 3.4 and Texmaker 3.2.

CHAPTER 3. DESIGN AND IMPLEMENTATION

- **Graphic edition.** Inkscape 0.48 and GIMP 2.6.

- **Web servers.** Nginx 1.1 and Apache 2.2.

- **Web browsers.** Google Chrome 18 and 19, Firefox 12 and 13 and Internet Explorer 9.

- **Version control system.** Dropbox was used until near the end of the development process because provides simple control versioning and synchronization for a single developer, but was migrated to GIT to allow third parties to contribute to the project in a future development.

3.2 User interface design

Before any code was written a series of mockups where design with Inkscape, these mockups represents how the interface is going to be like, and it helps on the design evaluation. This is, for example, the mockup for the home screen:

![Figure 3.2: Home screen mockup](image)
CHAPTER 3. DESIGN AND IMPLEMENTATION

Figure 3.3 shows how the current implementation looks like:

![Current implemented home screen](image)

Figure 3.3: Current implemented home screen

The interface is composed of three parts:

1. **Information bar.** On the top of the screen, shows the name of the logged user, the last access time, number of unreaded mails for the account and an exit button. The user name is also a link to a popup where the user can edit the account profile, unreaded mails is also a link to the webmail, and the exit button ends the session.

2. **Categories bar.** Bellow the information bar is the category bar. Following the model of goal oriented design and simplicity principle, all the control panel functions can be encapsulated into four big categories:
   - **Home.** Is a summary or an overview of the account status, information like quotas or notification falls in this category.
   - **Mail.** Any functionality related with mail, like vacation or aliases management, are found in this category.
   - **Web.** Not only web related functions like virtual host management belong to this category but databases as well. Databases are generally used as a web content information source, relation with web is close enough to include databases in the web category. This helps to simplify the interface without affecting workflow efficiency.
• Bills. All financial functionality like current contracted services, or billing information is found in this category.

The current selected category is highlighted with a white triangle below the category icon. By default home category is selected.

3. Content panel. The rest of the screen shows the widgets related to the selected category, every time the user clicks on a category, the content pane will change to show the related widgets to the selected category. If the content is greater than the available screen scrollbars are shown. In a category may be widgets that fall in not direct related categories, for instance, virtual hosts and database management are two subcategories of the web category, widgets are separated by tabs in these situations.

3.2.1 Widget design

Widgets are basic units of functionality, they perform a single task, for instance, the vacation widget only allows the management of the user out of office message.

Every widget must contain a title which clearly specifies the purpose of the widget. The title font has a H2 font size, black foreground color and white background for best readability (there are two exceptions, Notifications and Pending tasks which color are used for quick identification). The title must have 20px of left padding, context help (if needed) is indicated through a help icon next to the title. Input data for adding new items is asked for with the help of a dropdown dialog button, other components like activation or deactivation widgets may be added as well.

Widgets must have a maximum size, the size depends on the content of the widget, and never can be more than 740 pixels with padding and margins included, the purpose of this limit is to improve readability on low resolution tablets and smartphones.

All headers will be wrapped with a div of class headerSection.

As an example, this is the aliases widget title HTML code:

```html
<div class="headerSection">
  <h2> <div class="normalHeader">Aliases</div></h2>
  <div id="info_alias" class="infoIcon"></div>
  <div id="new_alias_drop"></div>
</div>
```

\(^4\) <H2 >HTML tag
Figure 3.4 shows the result of the above code:

![Image](image_url)

**Figure 3.4:** Aliases widget title

After the title comes the **content** of the widget, this area shows useful information about the widget functionality, it is possible to input additional data from the content area.

The following code belongs to the content of the aliases widget:

```html
<ul id="alias_list"></ul>
```

Figure 3.5 shows the look of a complete widget:

![Image](image_url)

**Figure 3.5:** Aliases widget

All widgets are wrapped with a *div* of class *widget*.

### 3.2.2 Dynamic layout

Widgets are automatically arranged depending on the horizontal resolution, by default it is used all the horizontal space available, if there is not more space, most right widgets will float below. Widget overflow is handled by scrollbars.

Figure 3.6 shows how the *home screen* looks like on a 1920x1080 pixels display resolution:
CHAPTER 3. DESIGN AND IMPLEMENTATION

Figure 3.6: Home screen at 1920x1080 pixels

Figure 3.7 shows the how the home screen looks like on a 800x600 pixels display resolution:

Figure 3.7: Home screen at 800x600 pixels
3.3 Categories

On section 3.2 a brief introduction to the four categories is done. In this section widget functionality of every category is explained.

3.3.1 Home

- Notifications

Notifications allow to keep a record of all important events, this way the user can keep track of previous user actions. An event can be a service activation or adding an item. Notifications can be generated by widgets and by the backend, for example, if the backend has generated a new bill, a new notification can be added.

Notifications are ordered by time, from most recent to the oldest. The backend must provide the notification list in the correct order.

Notification list height is 100 pixels, vertical overflow is handled by hiding old content and allowing scrolling.

It is the backend responsibility to set the number of remembered notifications. Storing all notifications is not a good practice as could lead to a oversized list.

- Pending tasks

A common behavior in modern web hosting control panels is to add tasks to a queue instead of execute them immediately, this widget shows the pending to be executed tasks for the current user.

Pending tasks are ordered by time, from the oldest to the most recent. The backend must provide the pending tasks list in the correct order.

Pending tasks list height is 100 pixels, vertical overflow is handled by hiding old content and allowing scrolling.

- Domains

This widget shows a list of the user domains.

- Mail space

Shows used and free available mail space in disk with a pie chart, hover or click reveals the size in Megabytes.
CHAPTER 3. DESIGN AND IMPLEMENTATION

- **Web space**
  Shows used and free available web space in disk with a pie chart, hover or click reveals the size in Megabytes.

- **Web bandwidth usage**
  This widget shows a graph of the bandwidth used by every domain for the past twelve months, each domain is represented with a line. Exact size in Megabytes is show on hover or click on the points of the line.

### 3.3.2 Mail

This category is divided in two tabs, *Mail account settings* and *Other accounts*

#### 3.3.2.1 Mail account settings

- **Vacation**
  Allows the activation or deactivation of an _out of office_ message, the user can select the response address, start date, end date and message in rich text format. The dates selectors do not allow to be the end data before the start date.
  If the vacation message is not active, a message notifying that is not active is shown instead of the message editor.
  Context help is provided.

- **Forwards**
  This widget shows a list of the user address forwards. New forwards can be added with a dropdown button, and deleted clicking the cross button next to the forward address.
  Context help is provided.

- **Aliases**
  This widget shows a list of the user address aliases. New aliases can be added with a dropdown button, and deleted clicking the cross button next to the alias address.
  Context help is provided.
• External access settings
  Shows the connection settings to be input in a mail client. Settings list is collapsed by default to maintain clean the interface.

3.3.2.2 Other accounts

• Other mail accounts
  This widget allows to add and delete other mail accounts. Only users with the required permissions will have access to this widget. The input dialog asks for the email address (which is used as user name) and account password.

3.3.3 Web

3.3.3.1 Websites

• Websites
  Allows the management of virtual hosts, each virtual host is represented with a box with these fields:

  -- Address. It is automatically generated by the backend and is the basic URL to access the virtual host.

  -- Description. The user can input a description of that virtual host. This field is optional.

  -- Directory. Path on the server where the website files are. This field is mandatory.

  -- Main domain. It represents the ServerName directive of Apache. This field is mandatory.

  -- Other domains. Is a list of aliases of the main domain, is equivalent to the Aliases directive of Apache. This field is optional.

  -- Database. If a database in this virtual host is used, the user can input the name in this field. This field is optional.

  In addition to the fields, the user has the option to upload a website in zip format using the Upload web in a zip file button, this will upload and unzip the file to the virtual host directory. If the files or directories already exists on the server, no files will be overwritten and the user will be informed.
CHAPTER 3. DESIGN AND IMPLEMENTATION

3.3.3.2 Databases

• Databases

This widget allows the user to manage databases. A table with three columns, database name, user and functions is shown. A dropdown button with three fields allows the user to add new databases, these fields are database name, database user and user password. Database name and database user are mandatory, if the user password is blank, an existing user with the database user name is used, if it doesn’t exists, an error is shown. If the user already exists and a password is supplied, an error is shown. The third column shows shows two icons, the first one is for changing the database user password, the second one is for deleting the database. If there are no more databases using the deleted database user, the user is also removed. Backend is the responsible of deleting databases and users.

• Connection settings

Shows database connection information.

3.3.4 Bills

• Services

Show a table with current contracted services. Columns showed are date, reference, description, status, quantity, price and tax.

• Bills

Shows a table with all user bills. Columns showed are date, reference, type, status, price, tax and a link to download the bill in PDF format.

• Billing data

This widget allows to edit the user bank account name and account number in Spanish bank number format.

3.4 Architecture design

The control panel, like any other web application, runs in a client-server architecture. Traditionally web applications are generated on the server, using languages like PHP, java or perl, then are transmitted to the client who requested the web application.
and rendered in the client browser, every section of the web application is generated on the server and transmitted every time the client requests it, even if it is the same section but with some content changed. For example, if you fail to login in a web login form, the complete page will be downloaded again exactly the same just with the login failed message added.

Since the introduction of XMLHttpRequest (XHR) 6 a web browser with JavaScript, can make a request to a web server and handle the response directly into the script, the response can be in JSON, XML, HTML or plain text. XHR, then, avoids many unnecessary web reloads, reducing waiting times and improving the user experience, as the web does not “flicker”.

XHR is the base for AJAX 7 a group of techniques that allow web applications to make asynchronous requests, some examples of web applications using XHR is gmail 8 or facebook 9.

The developed control panel downloads the web just once, and all the client requests are made with AJAX, the “feeling” the user gets is the one of using a desktop application.

Figure 3.4 shows a comparison of the traditional model and the AJAX model used in this project.

---

6Web browsers usually cache web pages, but this technique is very limited
6urlhttp://en.wikipedia.org/wiki/XMLHttpRequest
7http://www.w3schools.com/ajax/ajax_intro.asp
8http://gmail.com
9http://facebook.com
There are four XMLHttpRequest methods which allows the backend what action to perform with the request data:

- **GET.** Request for one or more objects.
- **POST.** Request to add one or more objects.
- **PUT.** Request to update an object.
- **DELETE.** Request to remove an object.

REST architecture requests and responses are built upon resources, a resource is any meaningful concept that may be addressed and is typically represented by a document that captures the current state of the resource. For example, a group of users could be resource represented by the string “/users/”, obtaining a user by the user identification could be “/users/id”, where id is the identification string. The name of the resources, also referred as URI, are totally arbitrary, but as a general rule, are meant to be readable and logical from a human point of view. The list of this URIs combined with the data format specification, for example JSON or XML, it is called the Apication Programming Interface (API).
Once the control panel is downloaded and rendered by the browser, all interactions with the backend are made through REST resources.

### 3.4.1 Storage model

A Model-View-Controller (MVC) is implemented on the client through JavaScript, this means there is a separation of the data management and the view management. Storages represent REST resources and are able to “talk” to the REST backend through its API and the view controller, which is the HTML and CSS manipulation layer.

The controller **view** accepts two functions:

- **add.** Adds an object to the view, creating the necessary HTML, CSS and JavaScript content, and keeping track of the object by its identification.
- **remove.** Removes all related HTML, CSS and JavaScript from the view by the object identification.

**Storage** accepts three functions:

- **get.** Obtains one or more objects from the RESTful server and calls view’s **add** for each object.
- **add.** The add function adds or updates an object in the view by calling the view’s **add** function and the *POST* or *PUT* RESTful server requests if adding or updating an object. If the object contains an identification, the object is updated, otherwise it is added in both, the view and the RESTful server.

Figure [3.4.1](#) shows the sequence diagram of the *add* storage function.
• **remove.** Removes an object from the view by calling view’s *remove* and from the RESTful server by requesting a *REMOVE* of the object by its identification.

Figure 3.4.1 shows the sequence diagram of the remove storage function.
CHAPTER 3. DESIGN AND IMPLEMENTATION

Figure 3.4.1 represents the interaction between the view, the storage and the RESTful backend:

![Diagram of view, storage, and RESTful server communication]

**Figure 3.11:** View, storage and REST server communication.

### 3.4.2 Views

Each widget implements its own creation view function. This function takes care of the widget data initialization and the implementation of the functions for adding or removing elements from the widget if necessary. These two functions will be called by the storage in case it needs to add or remove items from the widget.

The creation view function takes the name of `create<Widget name>View` where `<Widget name>` is the name of the widget.

For better understanding, figure 3.4.2 shows the sequence diagram of a generic creation view function:
CHAPTER 3. DESIGN AND IMPLEMENTATION

3.5 Client implementation

The client part uses HTML5 for document structure, CSS3 for styling, JavaScript for communication, DOM\textsuperscript{10} manipulation and the underlying program logic. JSON format is used for client and server communication.

Because of the importance of JavaScript in this control panel (90% of the all the code is JavaScript), a JavaScript toolkit was needed. A total of 7 toolkits were evaluated, features like components, graphics, graphs and dialog creation, DOM access abstraction and REST communication were looked for. Dojo\textsuperscript{11} was the chosen toolkit because integrates all needed modules, supports AMD\textsuperscript{12}, is lightweight and provides many advanced user interface components.

3.5.1 Files and directories structure

Figure 3.5.1 represents a tree of relevant files and directories:

\textsuperscript{10}Document Object Model is an interface that gives JavaScript access to the content, structure and style of a document.

\textsuperscript{11}http://dojotoolkit.org

\textsuperscript{12}Asynchronous Module Definition, http://dojotoolkit.org/documentation/tutorials/1.7/modules/
Almost all HTML structure is held in index.html file, the only exception is VirtualHost.html which is a template for creating virtual hosts forms.

Most of the styles are in style.css, although a few of them are set from JavaScript files.

Icons are stored in the images directory.

All JavaScript code, including the dojo toolkit is stored under the directory js, except some code snippets in index.html. A JavaScript file for each category can be found in this directory, with an additional file functions.js, containing helper common functions.

Translations are stored in the nls directory. Every language has its own directory named after its two character language name representation, e.g., es stands for Spanish. Inside each directory a file named resources.js will hold the translation strings. English language is an exception and its resource.js file is in the root of directory nls.

### 3.5.2 HTML structure

A single HTML file is used: index.html

The basic HTML structure of index.html is as follows:
JavaScript libraries are loaded inside the <head> tags, the <header> tags contains the information and category toolbar, <article> tags holds the content of each category. The profile dialog is defined in HTML between the </header> and <article id="home"> tags.

### 3.5.3 The dojo toolkit

The dojo toolkit is loaded with a single line of code in index.html:

```html
<script src="js/dojo/dojo/dojo.js"
  data-dojo-config="async: true, parseOnLoad:true, isDebug: true, locale:'en'">  
</script>
```

`data-dojo-config` is a variable used to configure some aspects of the toolkit, this is the meaning of the variables used:

- **async.** Defines if dojo core, which contains the basic functionality of dojo, should be loaded asynchronously.

- **parseOnLoad.** If true, parses the page when the DOM and all initial dependencies have loaded.
• **isDebug.** Tells *dojo* to be more verbose, allowing a better debugging with *firebug* or *chrome developer tools*.

• **locale.** Specifies the language, this also affects formats like dates and numbers.

**User interface components**

The *dojo toolkit* can enhance HTML user interface components by styling and giving additional functionality, for example, adding value verification to an input type component. *Dojo* provides additional components not found in HTML like calendar and graphs.

There are two ways of creating *dojo* components:

1. Using an existing HTML component, for example:

   ```html
   <input type="text" data-dojo-type="dijit.form.TextBox" id="firstName">
   ```

   This will “convert” an HTML input text box to a *dojo* styled and enhanced text input component.

2. Replacing a *div* or *span* tag programmatically:

   ```html
   <div id="inputbox"></div>
   <script>
   require(["dojo/form/TextBox"],function(textbox){
   new textbox({}, "inputbox");
   });
   </script>
   ```

   All components used have to be referenced inside the *require* directive.

**Events**

*Dojo* provides an improved alternative to the normal JavaScript events. *Dojo* can connect functions to one another, creating a link that calls one function when another fires.

A function can be connected to:

• A DOM event, such as when a link is clicked.

• An event of an object, such as an animation starting or stopping.
CHAPTER 3. DESIGN AND IMPLEMENTATION

- A custom function.
- A topic, which acts as a queue that other objects can publish objects to.

Stores
Section 3.4.1 describes the storage model used in this project, the core of the storage model are dojo stores.
Stores provides an interface to store objects in a structured manner. It is based on HTML5 IndexedDB object store API. Advantages of using dojo stores are:

- **User interface separation.** Data manipulation is done in the store, user interface components become just a representational view of the data.

- **JavaScript objects.** Data is passed and obtained with JavaScript objects, this simplifies the access to the data.

- **Layered.** Layers of functionality can be added, some examples are, JsonRest, which provides abstraction to a JSON REST server backend, and Cache, which provides caching capabilities to the store. Layers don’t change the store interface.

3.5.4 Widget implementation

Section 3.2.1 shows an introduction to widgets, in this section, a more deep analysis is done.

The basic structure of a widget is:

```html
<div class="widget">
  <div class="headerSection">
    <h2>
      <div class="normalHeader">Header name</div>
    </h2>
    <div class="header_underline"></div>
  </div>
  <div id="content"></div>
</div>
```

The only new code in the above example is `<div class="widget">` which is the `div` needed to make its contents behave like a widget, i.e., float the components and keeping distances.

For explaining how the content div is set, an example of creating a simple list is shown:
This example, retrieves the list items from a REST server and shows them in a list where `<div id="content"></div>` is.

Creating the view function

This function is responsible of creating the HTML components needed for the list and the add and remove functions to manipulate that list:

```javascript
function(JsonRest, dom, Observable, Memory, Cache, domConstruct, on, html){

// html container, store name
function createView(containerName, store){

//Get a reference to our container
var container = dom.byId(containerName);
var rows = [];

var results = stores[store].query({});

results.forEach(insertRow);

results.observe(function(item, removedIndex, insertedIndex){
    // this will be called any time a item is added, removed, and updates
    if(removedIndex > -1){
        removeRow(removedIndex, containerName, item.id);
    }
    if(insertedIndex > -1){
        insertRow(item, insertedIndex);
    }
}, true); // we can indicate to be notified of object updates as well

function insertRow(item, i){
    var row = domConstruct.create("li", {
        className: "SimpleList",
        innerHTML: item.message + "\&nbsp;" +
            '<div class="link closeIcon" id="rm_'+item.id+'"></div>"
    });

    container.appendChild(row);
}

��
```
CHAPTER 3. DESIGN AND IMPLEMENTATION

```javascript
row.itemId = item.id;
rows.splice(i, 0, container.insertBefore(row, rows[i]));

// Remove item event
on(dom.byId("rm_"+item.id), "click", function(){
    stores[store].remove(item.id);
});
}
```

The `observe` function links the store’s `add` and `remove` functions to the view’s `add` and `remove` functions.

Creating the store

Each REST resource needs a new store, the following code creates a store for a REST resource:

```javascript
newStore("aliases/", "aliasStore");
```

The first argument is the store URL, the second one is the name of the store, the purpose of this name is to have a reference to the store. Stores are kept in an array named stores, access to this store is done by accessing `store[‘aliasStore’]`.

Using the widget

The final step to setup the widget is to call the `createView` function:

```javascript
createView("content", "aliasStore");
```

Where `content` is the content `div` name and `aliasStore` the store name.

Note that the `createView` function uses a JavaScript object called `item`, which has two properties, `id` and `message`. The `id` property is used to keep track the the item, the `message` attribute is the string shown on each item of the list.

To add a new item, the store’s `add` function is called:

```javascript
stores[‘aliasStore’].add({message: ‘this is a new item’});
```

This will create an item in the REST resource with a new `id` and with the `message` property, and a new item in the user interface list. Figure 3.4.1 shows the sequence diagram of the `add` function.

To remove an item, the store’s `remove` function is called:
stores['aliasStore'].remove(id);

Where \( id \) is the item identification to remove. This will remove the item in the REST resource with identification \( id \) and remove the item from the user interface list. Figure 3.4.1 shows the sequence diagram of the \texttt{remove} function.

For updating an item, a properly existence control of the \( id \) property is needed in the \texttt{addItem} function.

### 3.5.5 Security

Users are authenticated with basic HTTP authentication. Once the user is logged in, the server generates a temporal “session” string in guid format. The session will expire in three hours or when the user logs out.

All communications uses SSL to avoid the stealing and use of the session string.

### 3.6 JSON REST Application Programming Interface (API)

The API is a specification for the communication between the client and the backend. The advantage of using an API is that any third party software could access the backend without knowing its internal details and making sure communication syntax will not change in the backend’s future development. An API makes feasible to develop a mobile application or to integrate an existing site with the backend. The method to allow modifications to the API without braking client compatibility is using versions, every request will include the API version. This specification is version 0.1 and and is specified in the URL with the string “v0.1”.

Usage of the API is done by accessing the following URL:

https://<server address>/<version>/<resource>

Where <server address> is the REST backend, <version> is the API version and <resource> is a backend resource.

An example of a valid URL is:

https://admin.pangea.org/panel/v0.1/aliases/

Note that the server address can contain subdomains and subdirectories.

**Supported operations**
CHAPTER 3. DESIGN AND IMPLEMENTATION

- **GET** Gets one or more objects from the server.
- **POST** Adds and object to a resource.
- **PUT** Updates an existing object.
- **REMOVE** Removes an existing object.

### 3.6.1 API definition

#### Considerations

- The string `{user}` refers to the user id, `{id}` refers to the object identification
- Dates use the format **YYYY-MM-DD**
- Dates and hours use the format **YYYY-MM-DD HH:SS**
- When adding new objects, parameters are optional if not specified otherwise.
- All POST and PUT calls returns the added or updated object. In case of the POST call, it returns an additional `id` parameter containing the object identification.
- Every string passed to the API must be in utf-8 encoding.

#### Table 3.1: REST Application Programming Interface

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
</table>

Continued on next page.
### Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GET {user}/profile</code></td>
<td>Returns the user profile.</td>
<td><code>id</code> user id</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>firstName</code> user first name</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>surname</code> user surname</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>altMail</code> alternative mail</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>cif</code> VAT identification number</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>country</code> user country</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>province</code> user province</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>town</code> user town</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>postalCode</code> user postal code</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>address</code> user address</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>telephone</code> user telephone</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>telephone2</code> user alternative telephone</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>language</code> interface language</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>lastAccess</code> last login date</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT {user}/profile</td>
<td>Update user profile data, all fields are optional.</td>
<td><code>firstName</code> user first name&lt;br&gt;<code>surname</code> user surname&lt;br&gt;<code>altMail</code> alternative mail&lt;br&gt;<code>cif</code> VAT identification number&lt;br&gt;<code>country</code> user country&lt;br&gt;<code>province</code> user province&lt;br&gt;<code>town</code> user town&lt;br&gt;<code>postalCode</code> user postal code&lt;br&gt;<code>address</code> user address&lt;br&gt;<code>telephone</code> user telephone&lt;br&gt;<code>telephone2</code> user alternative telephone&lt;br&gt;<code>language</code> interface language&lt;br&gt;<code>lastAccess</code> last login date</td>
</tr>
<tr>
<td>GET {user}/services</td>
<td>Get a list of contracted services.</td>
<td><code>reference</code> service reference number&lt;br&gt;<code>date</code> service request date&lt;br&gt;<code>description</code> a description of the service&lt;br&gt;<code>price</code> service price&lt;br&gt;<code>quantity</code> number of services contracted&lt;br&gt;<code>tax</code> tax percentage (if applicable)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET {user}/billing/bills/</td>
<td>Get a list the user bills</td>
<td>reference bill reference number, date bill request date, type bill document type (Invoice, membership fee,...), status bill status (Open, closed, send, paid...), price bill price amount, tax tax percentage (if applicable), pdfLink HTTP link to the downloadable bill document</td>
</tr>
<tr>
<td>GET {user}/billing/account</td>
<td>Get user’s billing account information. Spanish account number format.</td>
<td>name billing account name, entity financial entity number (4 digits), branch entity’s branch number (4 digits), controlDigit control digit (2 digits), accountNumber account number (10 digits)</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT {user}/billing/account</td>
<td>Update user’s billing account information. Spanish account number format. All fields are optional.</td>
<td><strong>name</strong> billing account name  <strong>entity</strong> financial entity number (4 digits)  <strong>branch</strong> entity’s branch number (4 digits)  <strong>controlDigit</strong> control digit (2 digits)  <strong>accountNumber</strong> account number (10 digits)</td>
</tr>
<tr>
<td>GET {user}/sites/</td>
<td>Get user’s virtual hosts. <em>description</em> and <em>databases</em> are free and doesn’t have any influence whatsoever</td>
<td><strong>id</strong> virtual host identification  <strong>description</strong> user’s description of the site  <strong>directory</strong> virtual host document root  <strong>domain</strong> virtual host server name  <strong>aliases</strong> list of aliases names  <strong>database</strong> database name</td>
</tr>
<tr>
<td>POST {user}/sites/</td>
<td>Add a new virtual host. <em>description</em> and <em>databases</em> are free and doesn’t have any influence whatsoever</td>
<td><strong>description</strong> user’s description of the site  <strong>directory</strong> virtual host document root (mandatory)  <strong>domain</strong> virtual host server name (mandatory)  <strong>aliases</strong> list of aliases names  <strong>database</strong> database name</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT {user}/sites/</td>
<td>Updates an existing virtual host. description and databases are free and doesn’t have any influence whatsoever</td>
<td>id virtual host identification (mandatory)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>description user’s description of the site</td>
</tr>
<tr>
<td></td>
<td></td>
<td>directory virtual host document root</td>
</tr>
<tr>
<td></td>
<td></td>
<td>domain virtual host server name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>aliases list of aliases names</td>
</tr>
<tr>
<td></td>
<td></td>
<td>database database name</td>
</tr>
<tr>
<td>DELETE</td>
<td>Removes an existing virtual host.</td>
<td>None</td>
</tr>
<tr>
<td>{user}/sites/{id}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET {user}/databases/</td>
<td>Get user’s databases.</td>
<td>id database identification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name database name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user database user name</td>
</tr>
<tr>
<td>POST {user}/databases/</td>
<td>Remove existing database.</td>
<td>name database name (mandatory)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user database user name (mandatory)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>password database user password</td>
</tr>
<tr>
<td>DELETE {user}/databases/{id}</td>
<td>Add new database.</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET {user}/quota/web</td>
<td>Get a list of domains with their annual bandwidth usage.</td>
<td>name domain name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data twelve item list representing each month of year, each item has the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x month number, 1 for January, 12 for December</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y bandwidth usage in Megabytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET {user}/quota/web/space</td>
<td>Get user’s web space usage.</td>
<td>used used web space in Megabytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>free free web space in Megabytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET {user}/quota/mail</td>
<td>Get a list of mail accounts with their annual bandwidth usage.</td>
<td>name account name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data twelve item list representing each month of year, each item has the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x month number, 1 for January, 12 for December</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y bandwidth usage in Megabytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET {user}/quota/mail/space</td>
<td>Get user’s mail space usage.</td>
<td>used used web space in Megabytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>free free web space in Megabytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET {user}/domains/</td>
<td>Get user’s domain list.</td>
<td>id domain identification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name domain name</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET {user}/notifications/</td>
<td>Get user’s notifications.</td>
<td>date notification creation date and hour message notification description</td>
</tr>
<tr>
<td>POST {user}/notifications/</td>
<td>Create a new notification.</td>
<td>message notification description</td>
</tr>
<tr>
<td>GET {user}/tasks/</td>
<td>Get user’s pending tasks.</td>
<td>date task creation date and hour message task description</td>
</tr>
<tr>
<td>GET {user}/forwards/</td>
<td>Get user’s forward addresses.</td>
<td>id forward identification address forward address</td>
</tr>
<tr>
<td>POST {user}/forwards/</td>
<td>Create new forward address.</td>
<td>address forward address</td>
</tr>
<tr>
<td>DELETE {user}/forwards/{id}</td>
<td>Remove a forward address.</td>
<td>None</td>
</tr>
<tr>
<td>GET {user}/aliases/</td>
<td>Get user’s alias addresses.</td>
<td>id alias identification address forward address</td>
</tr>
<tr>
<td>POST {user}/aliases/</td>
<td>Create new alias address.</td>
<td>address alias address</td>
</tr>
<tr>
<td>DELETE {user}/aliases/{id}</td>
<td>Remove an alias address.</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /accounts/mail/</td>
<td>Get user’s other mail accounts addresses.</td>
<td>id  account identification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>address account mail address</td>
</tr>
<tr>
<td>POST /accounts/mail/</td>
<td>Create new mail account.</td>
<td>address mail account address and login user name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>password mail account password</td>
</tr>
<tr>
<td>DELETE /accounts/mail/{id}</td>
<td>Remove a mail account.</td>
<td>None</td>
</tr>
<tr>
<td>GET /vacation</td>
<td>Get user’s “out of the office” message.</td>
<td>active vacation status, values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True vacation is active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False vacation is inactive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fromAddress vacation’s mail origin address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fromDate vacation’s activation date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>toDate vacation’s deactivation date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message vacation’s message</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT {user}/vacation</td>
<td>Update user’s “out of the office” message.</td>
<td>active vacation status, values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True vacation is active</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False vacation is inactive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fromAddress vacation’s mail origin address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fromDate vacation’s activation date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>toDate vacation’s deactivation date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message vacation’s message</td>
</tr>
</tbody>
</table>

#### 3.6.1.1 Error handling

**Return HTTP codes**

- **200** OK.
- **400** Bad request.
- **401** Unauthorized.
- **403** Forbidden.
- **404** Resource not found, doesn’t exists.
- **409** Resource conflict, already exist.
- **500** Internal server error.

When an error occurs on the backend, a response with an HTTP code different from 200 will be received, in that case a description of the error is included in the parameter `message` of the returned data.
CHAPTER 3. DESIGN AND IMPLEMENTATION

3.6.1.2 API usage examples

For these examples, john@pangea.org is used as user id.

- Get a list of John aliases
  1. Method: GET
  2. URL: https://admin.pangea.org/panel/v0.1/john@pangea.org/aliases/
  3. Returned HTTP code: 200
  4. Returned data:
     
     ```
     [
     {'id': 1253, 'address': 'john@gmail.com'},
     {'id': 1254, 'address': 'snow@yahoo.com'},
     {'id': 1255, 'address': 'loki@hotmail.com'}
     ]
     ```

- Add a new alias for john@pangea.org
  1. Method POST
  2. URL: https://admin.pangea.org/panel/v0.1/john@pangea.org/aliases/
  3. POST data:
     ```
     {address: 'fatboy@mountainview.com'}
     ```
  4. Returned HTTP code: 200
  5. Returned data:
     ```
     {'id': 1256, 'address': 'fatboy@mountainview.com'}
     ```

- Add an already existing alias for john@pangea.org
  1. Method POST
  2. URL: https://admin.pangea.org/panel/v0.1/john@pangea.org/aliases/
  3. POST data:
     ```
     {address: 'fatboy@mountainview.com'}
     ```
  4. Returned HTTP code: 409
5. Returned data:

{
    'message': 'Alias fatboy@mountainview.com already exists'
}

- Remove alias fatboy@mountainview.com for john@pangea.org

  1. Method DELETE
  2. URL: https://admin.pangea.org/panel/v0.1/john@pangea.org/aliases/1256
  3. Returned HTTP code: 200

- Activate “out of the office message” for john@pangea.org

  1. Method PUT
  2. URL: https://admin.pangea.org/panel/v0.1/john@pangea.org/vacation
  3. POST data:
  4. Returned HTTP code: 200

    {active: 'True'}

  5. Returned data:

    {
    'active': 'True',
    'fromAddress': 'john@pangea.org',
    'fromDate': '2012-05-09',
    'toDate': '2012-05-15',
    'message': 'Hi everyone!, I’m on <b>vacation!!</b>'
    }

3.7 REST Backend

A RESful backend has been implemented basically to aid in the debugging process of the client side, data can be added, removed and modified, but any changes are lost when stopping the backend, some examples are loaded on the start up, though.

The backend has been implemented in the python programming language with the help of the bottle framework, which simplifies the task of implementing a REST server.

\[^13\]http://bottlepy.org
Its only purpose is to provide the API interface, it does not serve web pages, for this end, *nginx* and *apache* has been used. Requests, though, are made through the web server, which acts as a proxy between the client and the REST backend. Request calls are identified by URL name, i.e., request to /api/ are redirected to port 8080, where the REST backend is listening.
Chapter 4

Conclusions and future work

This chapter covers an evaluation about the project from different points of view, a personal overview of this thesis development, and finally, a list of features and thoughts about future work on this project.

4.1 Evaluation

4.1.1 Reusability evaluation

This web hosting control panel interface covers all typically functions offered by any web hosting control panel as a user level profile, i.e., personal information, mail, web and billing management, the client (interface) communication with the server is done through a platform independent and widely used technology which is REST, examples of services using REST are twitter.com or gmail.com, this means no matter the platform, the language or the software used on the server, the interface and the backend will be able to communicate, specifics of this communication are covered in this document.

For anyone who wishes to use this web hosting control panel interface, the work to be done is implement a RESTful backend following the API specification on this document. This backend will trigger actions upon client requests through the API, these actions depends on the server configuration. For example, if a request of a new alias is made to the backend, the triggered action could be adding the address to a file, or adding a row to a database table.

The REST server implementation should not be a problem, there are already many open source implementations online for many platforms and many languages.
4.1.2 Usability evaluation

A big concern in this project is usability, a lot of research has been done in this matter, but the developer perspective of the interface may be biased, therefore some usability tests were done with people not involved in the development.

The goals of those tests are the measurement of the performance, that is, how much time and steps are required to complete a basic task, the accuracy, which means how many mistakes the user did, and emotional responses like level of stress or confidence.

For those tests, some basic tasks where asked for, questions were ask in random order, asking by category may give unwanted tips of the tasks place location, these are the questions:

- Password change.
- Change billing information.
- Find free and used web space.
- Activate the “out of the office message” for the following week.
- Add a new database, change password after that.
- Add mail account.
- Find currently contracted services.
- Find connection settings in order to configure a mail client.
- Add a new website.
- Deactivate the “out of the office message”.
- Find out how many mail accounts are left to create.
- Bandwidth used by a domain in May.

Four subjects happily help with the testing, two of them had never used a web hosting control panel before. Were daily users of email and had minimum knowledge of how the web works. The other two, had used a web hosting control panel in the past.
• Tester #1

30 years old, uses computers for email and web browsing, have some knowledge about content management systems as an editor and have never used a web hosting control panel.

Points out there is not enough feedback when saving or adding items, suggests a popup window notifying all changes. Doubts about the input directory when adding a new website (should have “/”?).

A constraint when adding dates makes not possible to select a from date after the to date in when configuring vacation, fails to acknowledge the restriction.

Appreciates to have a notification log with all his actions. All requested tasks completed without external help.

• Tester #2

27 years old, uses computers for work and for personal use, has knowledge about email, web content management and is an avid user of social networks, has never used a web hosting control panel.

All asked tasks are done quickly except password change task, doesn’t find out is in profile clicking the user name. Misses feedback on save buttons.

• Tester #3

38 years old, uses computers for work and for personal use, has advanced skills as email and as web user, use different content management systems. Experienced with DNS control panels.

Fails to find the password change dialog, which is shown at the top left corner clicking a link with the user name, the reason given is that was not expected on the left.

When asked for changing the bank account number, first click is on the profile, once she realized it was not there, cancelled the dialog and went to billing.

Like tester #1 has difficulties to acknowledge the date selector constraints where the vacation final date must be after the initial date.

Has difficulties to find where to add new accounts, she says it is expected in mail settings tab. According to the tester, the message informing about the creation limit of mail accounts may be indicating that no more accounts can be created, suggests a message informing just about the included accounts.
CHAPTER 4. CONCLUSIONS AND FUTURE WORK

Points out that when adding new websites, the directory field should have context help.

Uses the Y axis of the bandwidth graph to know the traffic usage of May, instead of clicking on the graph to show the exact amount.

• Tester #4

26 years old, advanced user of email and web, knows internally how they work. He is an advanced system administrator and has programming experience. Has developed a web hosting control panel.

Looks in the profile to change the billing information, then goes to the billing category, suggests to have them in both places. Points out the profile link should be at the top right of the screen.

When asked for web space usage, first clicks on web category, then goes to home page.

Suggests to hide the message editor when activating the vacation message. The reason given is to give more sense on status change.

Expected to have a category for databases instead of being in web category.

Finds unusual to have the mail connection settings for configuring a mail client, suggests to clarify which protocol, either POP or IMAP should be use.

Suggests removing the remaining mail accounts message and just inform the user of the new account cost if the account exceeds the included accounts of the user web hosting plan.

Completes all the tasks without help and give some suggestions:

– Some kind of menu to summarize the contents of each category.

– Domains should have its own category.

– More redundancy, for example graphs on the home page and web and mail category.

– Collapsible virtual hosts.

The overall usability test is satisfactory, almost every task is completed without doubt, but from the observed behaviour, evaluating the answers and suggested features by the testers, some interface changes need to be done in order to improve usability, these are (ordered by priority):
1. Change profile link from top left corner to top right corner.

2. Add billing data to the profile dialog.

3. Change name of “Other accounts” tab to “Mail accounts” or change the location of other accounts management.

4. Add more context help, for example to website directory input field.

5. Change plan limits information message to just show a message of how many accounts are included in with the plan.

6. Add statistics to web and mail categories.

7. More visual feedback.

4.1.3 Accessibility evaluation

Excluding the top bar which is white font over black background and notifications and pending tasks, all text uses black font color over white background.

No font in the interface is smaller than 0.8em, \textit{em} is a scale relative to configured user font size, 0.8em is 1/8 the size if user configured font size.

For understanding how blind coloured people see the interface colors, a filter to an screenshot can be applied, figure shows a comparison without and with this filter:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{interface_colors.png}
\caption{Figure 4.1: Left image shows interface normal colors, right image shows the same image as seen by a person affected by Deuteranope (a form of red/green color deficit).}
\end{figure}
CHAPTER 4. CONCLUSIONS AND FUTURE WORK

4.2 Personal overview

All the main goals have been achieved, although some desired features has been left out because of the project complexity has been greater than anticipated.

One subject with great complexity is usability, which can be sometimes confusing, many guides and other documents have been written about the subject, some concepts are recurrent and this is where one can make sense out of it. Aesthetics, as well, is difficult to master, and there is not a unique rule to follow.

Implementation took most of the thesis time, only basic concepts about the JavaScript framework dojo where known and some details, like usage and bugs, which delayed the development arise during the implementation, this, somewhat is expected.

Because of the variety of languages (HTML, CSS, JavaScript and Python), formats and browser compatibilities, an automatic debugging environment was not feasible with the time available.

4.3 Future work

The next step is to use this project as an interface for Marc’s web hosting control panel, currently he is implementing the REST server. Once this is done, the following features are the objective for the next release (ordered by priority):

1. Language translation.

2. Software installation.

3. ftp accounts support.

4. Ticket support.

5. Domain management.

6. Services report.

7. FTP client.

8. Payment gateway.
Bibliography


Online resources


[8] *w3schools*. [http://www.w3schools.com/](http://www.w3schools.com/)
