Degree Final Project

Bachelor Degree in Informatics Engineering
Specialization in Software Engineering

Skimmer Tracker
Building a payment card skimmer database

Arnau Buch Torrents

**Project Director:** Nolen Scaife, Department of Computer Science, CU Boulder

**Internal examiner:** Ernest Teniente, ESSI Department, FIB-UPC

Defense date: 30th of June, 2020
Abstract

With the rise of credit cards as an integral part of the economy, crime related to them has risen accordingly. One of the most common ways to steal credit card data is through skimmers in gas-pumps. The skimmer device consists of a simple PCB (printed circuit board), and it is inserted inside the gas-pump to steal consumer’s credit cards. Incurred costs due to fraud can go well into the thousands per person.

Criminals install multiple skimmers across counties and states in the US. When skimmers are eventually discovered it is practically impossible for police to conduct a successful investigation on them. Police departments rarely collaborate on these sorts of cases that span different counties and states, which eliminates any possibility of them being solved.

Skimmer Tracker is a web application that lets law enforcement agencies publish the skimmers they find. With this sharing of evidence we aim to group different skimmers and connect them as part of the same case through computer vision based analysis.
Resum

Amb l’auge de les targetes de crèdit com a part integral de l’economia, els delictes relacionats amb elles ha augmentat corresponentment. Una de les maneres més comunes de robar les dades de targetes de crèdit és a través de *skimmers* als sortidors de gasolina. L’*skimmer* consisteix en una simple PCB (circuit imprès) que és insertada dins el sortidor per robar les dades de les targetes dels clients. Les despeses com a causa del frau poden arribar fins als milers per persona.

Grups criminals instal·len diversos *skimmers* a través de comtats i estats dels Estats Units. Quan els *skimmers* són descoberts eventualment, és pràcticament impossible dur a terme una investigació policial satisfactòriament. Els departaments policials rarament colaboren sobre aquests casos que abasten diversos comtats i estats, el qual elimina qualsevol possibilitat de ser resolts.

Skimmer Tracker és una aplicació web que permet a departaments policials publicar els *skimmers* que hagin trobat. Compartint l’evidència de diferents casos pretenem connectar-los com a part del mateix cas a través d’anàlisi basat en visió per computador.
To Nolen Scaife, for the selfless support and all the invaluable advice

To Ernest Teniente, for the assistance on the format of this document

To my family, for their constant love and encouragement in moments of doubt and despair

To my quarantine mates, for brightening up these troubling times
# Contents

1 Introduction .......................... 1
   1.1 Context ........................................ 1
   1.2 Terms and concepts ...................... 2

2 Context study and scope ................. 5
   2.1 Problem to be solved .................. 5
   2.2 Existing solutions ....................... 6
   2.3 Proposed solution and justification ...... 7
   2.4 Potential obstacles and risks .......... 8
   2.5 Limitations of the system .......... 8

3 Methodology and rigor .................. 10
   3.1 Methodology and validation methods .... 10
   3.2 Development environment and tools .... 11

4 Requirements analysis .................. 13
   4.1 Stakeholders ............................ 13
   4.2 Objectives .................................. 15
      4.2.1 Main objective ......................... 15
      4.2.2 Sub-objectives ......................... 15
      4.2.3 Functional objectives .................. 15
      4.2.4 Other objectives ....................... 16
   4.3 Characteristics of the system .......... 17
      4.3.1 Use case diagrams ..................... 17
      4.3.2 Functional requirements ............... 17
      4.3.3 Non-functional requirements .......... 24
   4.4 Conceptual schema ........................ 32

5 Software design ....................... 34
   5.1 Patterns and principles ................ 34
   5.2 Architecture .............................. 35
      5.2.1 Frontend ................................ 36
      5.2.2 Backend ................................ 38
   5.3 Computer vision .......................... 41
      5.3.1 Alternative analysis .................... 43
      5.3.2 Pre-processing .......................... 45
      5.3.3 Limitations .............................. 45
   5.4 System Workflow ........................... 46
# Implementation

6.1 Frontend .................................................. 50
6.2 Backend ................................................... 55
6.3 Image analysis algorithm .................................. 59
6.4 Testing ...................................................... 59
6.5 Summary of used technologies ............................... 60

# Time Planning

7.1 Project duration and deadlines ............................. 62
7.2 Description of tasks ......................................... 62
7.3 Table of tasks ............................................... 68
7.4 Gantt table .................................................. 69
7.5 Risk management ........................................... 71
7.6 Changes from the initial time planning ....................... 71

# Budget

8.1 Identification of costs ........................................ 73
8.1.1 Human resources ......................................... 73
8.1.2 General costs ............................................ 75
8.1.3 Incidentals and contingencies ............................. 77
8.1.4 Total costs ............................................... 77
8.2 Management control ......................................... 78

# Sustainability report

9.1 Environmental Dimension .................................... 80
9.2 Economic Dimension ......................................... 81
9.3 Social Dimension ........................................... 82

# Conclusions

10.1 Future work ................................................ 84
10.2 Completion of technical competences ....................... 86
1. Introduction

The project presented in this report is part of a Degree Final Project at FIB-UPC (Facultat d’Informàtica de Barcelona-Universitat Politècnica de Catalunya), with the following title: “Skimmer Tracker: Building a payment card skimmer database”. The project was developed in its entirety at the University of Colorado Boulder, as part of a mobility stay with the Balsells Mobility Program.

Balsells Mobility Program

Started in 1996 at the University of California, Irvine by Peter Balsells, a renowned Catalan expat and inventor, the Balsells program allowed Catalan students to pursue Graduate Studies there. The program expanded to other universities such as the University of Colorado Boulder in 2010, with the addition of the Balsells Mobility Program, which lets students complete their Bachelor’s or Masters theses under the supervision of a University Professor [11].

I was accepted by Professor Nolen Scaife to supervise my Final Degree Project. Professor Scaife is part of the Computer Systems Lab of the Department of Computer Science, where he performs research on topics such as payment systems and ransomware. My project falls on the subject of payment systems, and more specifically skimming devices, with which Prof. Scaife has performed extensive research.

1.1 Context

Most point-of-sale systems and ATMs across the US still primarily use magnetic stripes to receive payments with credit cards [73]. This technology encodes information in plain text, making it an easy target for criminals to steal the information and duplicate the card to commit fraud. The most common way for this information to be stolen is with the devices known as
payment card skimmers. These devices can be of many types and can be found in all sorts of payment stations, most notably at ATMs and at gas pumps. Scaife et al. [62] identify five main types: overlays, deep-inserts, wiretaps, EMV shimmers and internal skimmers. The first four require external-access and the last, as the name suggests, internal. External skimmers are usually found in ATMs, and internals in gas pumps. In our project we’ll center around those skimmers found at gas pumps, internals. The ways in which criminals retrieve the stolen card data ranges from SD cards (and thus having to physically extract the device) to Bluetooth modules or even mobile data enabled skimmers.

Having the large majority of gas stations using payment terminals with magnetic stripe as the only available type of payment system means that three-quarters of Americans use it every day. A clear solution to this problem is upgrading to EMV or contact-less technologies, but that has proven to be economically unfeasible for many gas stations [67]. Given that EMV technology is also vulnerable to skimmer attacks it is useless from a criminal standpoint to replace existing payment systems with it. What is currently missing in skimmer related cases is for law enforcement to pursue the cases to their full extent.

This project proposes a solution to enable law enforcement agencies to work on skimmer cases, by collaborating through our system.

1.2 Terms and concepts

- **Payment card fraud**: use (often with previous duplication) of a payment card by an individual other than the original owner, with a malicious end.

- **Magnetic strip technology**: most commonly used in credit, debit and gift cards, this technology encodes card data in plain text in a magnetic strip found on the back of the card.

- **Law enforcement agency**: public agency dedicated to the enforcement of laws, the pursuit of crimes; the police.

- **Payment card skimmer**: device that is introduced by criminals in a variety of ways in payment systems to steal credit card information. Scaife et al. identify five main types:
1.2. Terms and concepts

– Overlays are the most prevalent type, usually found in ATMs. These types of skimmers are placed over the card slot and mask themselves as being part of the terminal. Inside them is a read-head capable of reading the magnetic strip on the card.

– Deep-inserts have a shape similar to that of a credit card and are inserted inside the magnetic card slot and fixed inside the terminal. Just like overlays, they contain a read-head which can steal the credit card data.

– Wiretaps can be found on the communication path between the card reader and the terminal’s main board. It is a rare type of skimmer.

– EMV shimmers are placed between the EMV reader and the chip on the card, thus performing a man-in-the-middle attack. The information acquired here may not be usable to replicate a card.

– Internal skimmers require the terminal to be opened and are then inserted between the payment components (card reader and pin pad) and the main board, thus performing a man-in-the-middle attack. Of those the most prominent types are the ones found inside gas pumps, which we can see in Figure 1.1d. These usually consist of a PCB with a micro-controller (MCU) and a flash memory to store the card data, with the adequate connections to the card reader and the main board of the pump. These skimmers usually come with an added module from where to extract the data remotely, most typically Bluetooth or GSM enabled. These skimmers also come equipped with a heat-shrink around the PCB.

Note that in the case of external skimmers the PIN number also needs to be stolen by the criminals, and that is performed with either a hidden camera or an overlay over the PIN pad.

• Gas pump point-of-sale (or pay-at-the-pump terminal): payment system used at gas stations where users can pay for gas with a credit card, all in the same place where they fill their gas tank, without having to go into the store.
Section 1. Introduction

(a) Overlay skimmer. Source: NYPD
(b) Deep insert skimmer. Source: NYPD
(c) Internal gas-pump skimmer. Source: Parker PD
(d) Internal gas-pump skimmer. Source: [63]

Figure 1.1: Some of the types of skimmers

Figure 1.2: Gas pump with an included payment station. Source: Parker PD
2. Context study and scope

2.1 Problem to be solved

As credit card use rises across the US, so does crime attempting to exploit its vulnerabilities. The FTC reported a 72% increase in credit card fraud from 2018 to 2019 [18]. Credit card fraud is considered a misdemeanor and even a felony across most US states, and as such law enforcement agencies must pursue such crime [20]. In the case of fraud, a common way to steal the card’s information is through the use of skimmers. We interviewed two Colorado detectives, one from Parker PD and another from Denver PD, to learn firsthand about the procedures taken with skimmers. Skimmer cases present many difficulties for them to be solved. We were told that when police discover one of these devices they have no indication as to who installed it or when, thus making it an unsolvable case. Furthermore, on the local scale, the proportions of the crime are not big enough to deserve a full investigation (in the case of Parker PD, we were told they find 6-7 skimmers each year). Since the criminals travel across different counties and states, when the skimmers are found it’s practically impossible to trace the route taken by the criminals to install them. Finally, the device gets stored in a police department to never be seen again. Since skimmers are not found that often and the eventual card fraud can’t get traced back to a skimming device, these cases are rarely closed successfully. Another important dampener in pursuing skimming crimes is the low level of collaboration between agencies, as both detectives told us.

At Denver PD we were told about the case of a Cuban national who, at 24, was instructed on how to build a skimmer at home. After building a multitude of them he was sent to the US to install them across a specified route. Knowing this, we can extrapolate the fact that given two visually similar skimmers it’s very likely they were built or designed by the same person. This
knowledge can be very helpful in solving a case. The lack of tools to identify these similarities has been a major deterrent in pursuing cases, and something we set out to fix with this project.

Professor Scaife has been working on payment systems since 2016, and more specifically on skimmers (see [62], [63], [64]). In his work, Professor Scaife tackles the subject of detecting whether the skimmer is present on a payment system, but making skimmer cases easier to solve for police remains a pending matter. A good first step for this is to create an application with which law enforcement agencies can aggregate and collaborate on requisitioned skimmers to identify commonalities. More details on the scope and features of the application later on.

### 2.2 Existing solutions

Before explaining our proposed solution in depth we must first analyze a solution in the market similar to ours. Although we weren’t aware of its existence before starting the project, at the first meeting with Parker PD’s crime analyst he quickly identified a similar system to our proposed one called Graffiti Tracker [28]. The basic idea of the system is similar to ours: a database web application where law enforcement agencies can post data (images, location,...) about graffiti they find in their respective areas. Graffiti Tracker analysts then perform manual review of the images and try to connect them based on various features. This helps trace back different graffiti to their unique creator and build stronger cases for prosecution. Graffiti Tracker relies on manual expert review to connect two different graffiti [31], which is what differentiates it with our proposed system. Furthermore, graffiti can’t be requisitioned as evidence, and thus pictures need to be taken at the site itself. That is why Graffiti Tracker built a phone app to take photos of graffiti and upload them to the system.

In a report by the San Diego Association of Governments (SANDAG) from 2012 [14], a dramatic increase in court-ordered restitution for the removal costs of graffiti is outlined, mainly thanks to Graffiti Tracker. In the report, law enforcement officials state that such a system “can enhance their ability to identify graffiti taggers and gather evidence for prosecution of multiple acts of vandalism”. During a pilot phase, agents used GPS-enabled cameras on the field instead of their phones, since at the time such devices were not widespread yet. A survey was performed after the implementation of the system during the pilot year. The majority of users received
previous training on how to use the system from the Graffiti Tracker staff itself and found it useful, but also found they would have needed more training on the use of the GPS-enabled cameras. The majority of users also found the system as a whole easy to use, with the GPS syncing of pictures being the less easy part. Our proposed system would reduce some of the complexities of Graffiti Tracker by not having any external system other than the application itself. Finally, in the case of skimmers the number of instances is not as big as with graffiti, but the incurred costs can be much higher; the FTC reports an average loss of 320$ due to fraud, and a quarter of cases were over 1,000$ in 2019\[^{18}\].

\section*{2.3 Proposed solution and justification}

The proposed solution is to build a web application where law enforcement agents can see data about skimmer cases, as well as add their own. With it, agents could also filter and sort the skimmers in various ways. The application would take the pictures and run an image analysis algorithm that would let it find similarities between two different skimmers and build a connection. Since skimmers are taken in as evidence when they are found, it is not necessary to build a system to allow agents to take the pictures in the field (this was confirmed by the two detectives we interviewed).

The option of adapting Graffiti Tracker’s existing system was never considered since Graffiti Tracker is a private company and we have no access to their architecture or code base. Furthermore, their system differs to ours in many ways which would make it hard to adapt.

The choice of the platform where users will access the system is not trivial, since potential users of the system will be on different platforms. As such, a web application is the optimal solution to cover the most platforms possible. Other types of systems that were considered are a mobile application (in case the detectives needed to use the system on the road, but that turned out not to be the case) and a desktop application, which although it could be made available for any device, it would have supposed a slight increase in development difficulty over a web app.

The system architecture would then consist of two simple parts: the website where users can interact with the database with ease and a secondary server dedicated to storing and analyzing images of skimmers. Below in figure you can find a schema of the described architecture. The
terms *frontend* and *backend* are very loosely used here, with *frontend* simply referring to the part of the application that users directly interact with and *backend* to that with which they don’t.

![Proposed system architecture](image)

Figure 2.1: Proposed system architecture

2.4 Potential obstacles and risks

- **Low skimmer cases**: as we learned during our interviews with police departments, the number of cases of skimmers is relatively low. This could potentially make the system useless in the eyes of involved police agents. Also, a low number of skimmers would mean an increased difficulty in building a reliable algorithm to identify similarities.

- **Inexperience**: on that same note, the inexperience of the author with computer vision techniques will affect in some way or another the development of the project. This is the most advanced and modern part of the project, and so we must be conscious of this potential liability before undertaking this project. An algorithm that is not capable of identifying similarities between skimmers would render the entire project useless.

- **Time limitations**: the predicted time planning (see section 7) does not hold up and the project ends up taking longer than predicted.

2.5 Limitations of the system

- **Only gas pump skimmers**: we will limit the scope of the project to only gas pump skimmers. If the system were to be successful with this type of skimmers an expansion to other types would be considered.

- **Only PCBs analyzed from skimmers**: the only part that will be considered for analysis from the skimmers is going to the PCB (Printed Circuit Board). Most gas-pump skimmers
2.5. Limitations of the system

come with a cover for the PCB, and so users of the platform will need to upload pictures with only the circuitry. Pictures with the skimmer covered or showing other parts will make the system fail in its analysis task.

• Basic image analysis: due to time and experience constraints, we will not be performing any PCB detection in the images using computer vision based machine learning, although some successful tests were performed. This means uploaded images will need to be manually cropped and rotated to fit the PCB. More on that in sections 5.3 and 7.6.

• Closed system for police agents: the system will be intended for use only by police detectives and/or agents, and not by the general public. This is also necessary for two important reasons: to protect potentially confidential information and information that could help criminals in their efforts to evade law enforcement.

• Only Colorado law enforcement agencies: for this initial prototype we will only consider skimmer cases from Colorado. If the system is successful an expansion within the US would be considered.

• We can’t meet with future users regularly: since this is not a project funded by law enforcement itself, meetings with agents will be limited, with a guaranteed initial meeting but nothing else promised.
3. Methodology and rigor

3.1 Methodology and validation methods

The project will use the agile methodology Kanban [35] throughout its design and implementation phases. Kanban originated in the late 1940s as the Toyota Production System, used in Toyota factories as the name implies [19]. The system works by limiting production to consumer demand, applying the "just in time" (JIT) principle. At the start of the 21st century the methodology was converted to the Kanban methodology, which coupled with the popular Agile manifesto was started to be used by software development teams. When applied to software engineering, the methodology expands on the values and principles of the manifesto by using a continuous flow of development. With it we can be very flexible to any sort of obstacle that might pop up since we are not bound by a tight schedule, all while maintaining an organization of tasks.

The basic element of the Kanban methodology are cards, which represent work items. Cards are then distributed across columns of a board to keep track of their status. A Trello board [9] will be used for managing cards, with the following columns:

- Backlog: serves as a list of features or ideas that can be worked on when ready.
- Design: cards that are currently being designed will sit on this column.
- Development: cards that are currently being developed will sit on this column.
- Testing: cards that are being tested will sit on this column.
- Done: for storing cards that have been finished.
Cards will be passed from column to column as the project progresses, normally in the order of Backlog to Done. That being said, exceptions could be made to move cards backwards in case Kanban uses the concept of WIP limits to control how many tasks are being done concurrently and stop bottlenecks from happening. We will set a limit of 2 tasks for the columns Design, Development and Testing. In contrast to the Scrum methodology, Kanban does not follow a set cadence of intervals (called sprints in Scrum) nor a fixed set of roles. This facilitates a much more flexible flow of the project and allows for continuous delivery of the software. Furthermore, some aspects of Scrum are very team-based and applying them in a project with a single developer would be unproductive. With all of that being said, a time planning table will still have to be developed as an estimate of project development time, necessary to create a budget.

Incremental testing of the software will be employed so that we are always aware of any errors or bugs the product might have. This will be employed with various techniques such as Unit Testing and Integration Testing.

The git code versioning tool will be used to maintain versions of the software and to store the code remotely (with Github). No code shall be committed to git without passing all tests beforehand.

Finally, to make sure objectives are met, a weekly meeting with the Project Director will be conducted, where progress of the project will be discussed and future goals will be set.

3.2 Development environment and tools

The work of the project will be conducted at the Systems Lab of the Engineering Center at the University of Colorado, Boulder. All the work will be done on the writer’s laptop. Here follows a list of tools that will be used throughout.

**Google Docs and Sheets**

All documentation for the project will be written in Google Docs as a word processor and Google Sheets as a spreadsheet editor. They are free, web based and automatically keep the documents on Google’s servers backed up, which makes it ideal for any sort of vital work such as a Degree
Final Project. Both will be used to quickly write drafts of the project’s documentation.

**Overleaf**

Overleaf is an online LaTeX editor that will be used to prepare the definitive documents for the documentation of the project and the presentation.

**Trello**

As mentioned in the previous section Trello will be used to keep an updated Kanban board. Trello is a simple yet powerful web based service that lets the users organize tasks in boards. There are other tools for maintaining a Kanban board, such as the highly professional Jira, but there’s no need for complicated equipment in a project of this size. Two boards will be set up, one for the development of the *frontend* server and one for the *backend*.

**Git and Github Desktop**

Git is currently the most popular version-control tool in the world and will be used for maintaining versions of the code during its development. Github Desktop is a companion app that makes using Git easier and integrates well with the web service of its same name, Github, where Git repositories can be stored for free.

**Visual Studio Code**

Visual Studio Code is a code editor from Microsoft that is widely used in the industry. It integrates nicely with all sorts of programming languages and is very extensible through a large library of extensions.
4. Requirements analysis

Throughout the next sections we use the Volere Requirements Specification Template [58]. With it we achieve a standardized format that industry members and academics alike will be familiar with and that makes this document more accessible.

4.1 Stakeholders

We now move on to identify the main stakeholders of this project. These are not necessarily those that will use the system but rather those that will be affected by it in some form.

- **Law enforcement agents**: the main users of the system, they will be tasked with not only using the system but also evaluating it and pointing out its shortcomings. Due to time and scope constraints the only agencies that will be considered will be those in Colorado.
  
  - **Roles**:
    
    - Use the developed system to manage his skimmers and compare them
    - Provide feedback on the system
  
  - **Objectives**:
    
    - Use the system efficiently and without committing many errors
    - Increase the number of skimmer cases closed through collaboration between agents
  
  - **Subject matter experience**: The users of the system are expected to be familiar with skimmers. They should know, for example, that gas-pump skimmers have a PCB surrounded by the heat-shrink cover.
Section 4. Requirements analysis

- Technological experience: The users are expected to have some basic experience using computer interfaces.

- **Gas station clients**: as the main group of people affected by skimmer crime, their objective in the project is to reduce skimmer stemmed fraud.

- **Gas station owners**: as owners they share the main objective of their clients, but theirs comes from the need of avoiding the bad publicity caused by police removing a skimmer from their station. The agent from Denver PD we interviewed mentioned how much gas station owners fear a police patrol might come to their station to remove a skimmer.

- **Developer**: tasked with designing, implementing, testing and maintaining the proposed system, as well as writing out its documentation and other supporting documents. The developer will also have to interview the law enforcement officers during the meetings conducted. This entire role will be done by myself, the writer.
  
  - **Roles**:
    - Build and design the product
    - Maintain the system running after the project’s end
  
  - **Objectives**:
    - Learn new technologies and methods that fulfill the course objectives
    - Complete the project in a timely manner

- **Project director**: the role of the project director, Professor Scaife at CU Boulder, will be to supervise and advise on the technical aspects of the project. He will also be tasked with connecting the developer with the law enforcement officers and setting a date to conduct the meetings. His main objective in this is to get the student to develop a working product before the end of the project.

- **Project speaker**: the role of the speaker (Professor Teniente at FIB-UPC) will consist of advising the project on its academic aspects (documentation, final presentation,...). Note that his location throughout the project will be different from the rest of stakeholders, but given his different role in the project, that should pose no difficulty in the development of the project.
4.2 Objectives

In this next section we will go over some of the objectives of the project.

4.2.1 Main objective

The main objective of the project is to design and implement an application to help identify commonalities between skimmer cases from different police departments, so that solving skimmer cases becomes easier.

4.2.2 Sub-objectives

- Design and implement a web application where users can view skimmers, see similar ones and filter and sort them by various parameters (date found, location,...)
- Design and implement a server application that handles skimmer image analysis and its storage.
- Design and implement an algorithm capable of identifying similarities between skimmer’s PCBs.
- Help law enforcement agencies combat skimmer crime by making cases easier to solve.
- Evaluate the importance of skimmer case data in pursuing a case and in matching skimmers as part of the same case.

4.2.3 Functional objectives

- The final system is scalable and does not fail under heavy usage.
- Code is extensible, this way the system can be extended.
- The code has a high test coverage to ensure changes don’t break it.
- The documentation accompanying the software is easy to follow and adequate.
- The skimmer similarity algorithm has a minimum of 75% accuracy when identifying similar skimmers.
4.2.4 Other objectives

This project will not only be conducted in the scope of CU Boulder, but will also be presented at FIB-UPC in front of a designated panel as part of the undergraduate Degree Final Project. As such, I intend for this project to adhere to the institution’s requirements by fulfilling the specific objectives set by the FIB, all while delivering the project in a timely manner. Furthermore, given the context where this Project will be undertaken (in the United States inside a prestigious University) I have also set the following objectives:

- Learn new technologies in both the fields of Software Engineering and Computer Vision.
- Expand knowledge about American culture, through interactions with both Project Director, law enforcement officers and other students.
4.3 Characteristics of the system

4.3.1 Use case diagrams

In figure 4.1 below we can see a summary of all the use cases in the system. We explain them in the next section.

![Use case diagram]

Figure 4.1: Use cases

4.3.2 Functional requirements

1. **Insert data about a new skimmer into the system:**

   Primary Stakeholder: law enforcement agent

   Precondition: the user has logged into the system
Trigger: a law enforcement agent wants to add a new skimmer into the system

**Principal scenario:**

(a) The user indicates it wants to add a new skimmer to the system.
(b) The system displays a skimmer form.
(c) The user inserts into the form the date the skimmer was found, its location and the type of security the pump.
(d) The user selects a number of images of the skimmer’s PCB.
(e) The user submits the form.
(f) Upon submitting the form, the system validates it and registers it.
(g) The system starts analyzing the added images to find similarities with existing skimmers from the system.

**Extensions**

*The inputted data is not valid*

(a) The user inserts invalid data in the form and submits it.
(b) The system displays a validity error and explanation for its cause.

2. **Modify data about a skimmer in the system:**

Primary Stakeholder: law enforcement agent

Precondition: the user has logged into the system, the skimmer exists and the user or someone else from his agency added it to the system

Trigger: a law enforcement agent wants to modify a skimmer

**Principal scenario:**

(a) The user indicates it wants to modify a skimmer in the system.
(b) The system displays a skimmer form with the skimmer’s current data.
(c) The user modifies the form and submits it.
(d) The system validates the data and updates the skimmer accordingly.

**Extensions:**
4.3. Characteristics of the system

The inputted data is not valid [as in requirement 1]

3. **Delete a skimmer from the system:**

   Primary Stakeholder: law enforcement agent

   Precondition: the user has logged into the system, the skimmer exists and the user or someone else from his agency added it to the system

   Trigger: a law enforcement agent wants to delete a skimmer from the system

   Principal scenario:

   (a) The user indicates it wants to delete a skimmer from the system.
   (b) The system deletes the skimmer from the system.

4. **Add a law enforcement agency into the system:**

   Primary stakeholder: administrator user/developer

   Precondition: the user is logged in and has admin privileges

   Trigger: an admin user with permissions wants to add a law enforcement agency into the system

   Principal scenario:

   (a) The admin user indicates it wants to add a law enforcement agency to the system.
   (b) The system shows a law enforcement form to the user.
   (c) The user fills the form with the name, the location and the email prefixes of the agency, and submits the form.
   (d) The system validates the form and registers the agency to the system.

   Extensions:

   The inputted data is not valid [as in requirement 1]

5. **Modify a law enforcement agency from the system:**

   Primary stakeholder: administrator user/developer
Precondition: the user is logged in, has admin privileges and the agency exists in the system

Trigger: an admin user with permissions wants to modify a law enforcement agency in the system

**Principal scenario:**

(a) The admin user indicates it wants to modify a law enforcement agency in the system.
(b) The system shows a law enforcement form to the user with the current data.
(c) The user fills the form with the updated data and submits it.
(d) The system validates the form and registers the changes to the system.

**Extensions:**

*The inputted data is not valid* [as in requirement 1]

6. **Delete a law enforcement agency from the system:**

Primary stakeholder administrator user/developer

Precondition: the user is logged in, has admin privileges and the agency exists in the system

Trigger: an admin user with permissions wants to delete a law enforcement agency from the system

**Principal scenario:**

(a) The admin user indicates it wants to delete a law enforcement agency from the system.
(b) The system deletes the agency.

7. **Register a user into the system:**

Primary stakeholder: law enforcement agent

Precondition: none

Trigger: a user (law enforcement agent) wants to get registered into the system
4.3. Characteristics of the system

**Principal scenario:**

(a) The user indicates it wants to get registered to the system.
(b) The system shows a user register form.
(c) The user fills the form with his email, password and law enforcement agency and submits it.
(d) The system validates the form and registers the user to the system.

**Extensions:**

*The inputted data is not valid* [as in requirement 1]

*The user’s email does not belong to the selected law enforcement agency*

(a) The user fills the form with an email address that does not belong to the specified agency.
(b) The system displays an error indicating an invalid email address.

8. **Visualize nearby skimmers:**

Primary stakeholder: law enforcement agent

Precondition: the user is signed in

Trigger: a user wants to view nearby skimmers in the system

**Principal scenario:**

(a) The user indicates it wants to view skimmers in the system sorted by their distance to the user.
(b) The system shows a list of nearby skimmers.

**Extensions:**

*The system has no skimmers registered yet*

(a) The user requests a list of nearby skimmers but the system has no skimmers.
(b) The system displays a message indicating there are no skimmers.

*The system is unable to determine the user’s location*
(a) The user requests a list of nearby skimmers but has its location obfuscated.
(b) The system displays an error indicating the inability to get its location.

9. **Visualize skimmers sorted by their date found:**

Primary stakeholder: law enforcement agent

Precondition: the user is signed in

Trigger: a user wants to view skimmers in the system sorted by their date found

**Principal scenario:**

(a) The user indicates it wants to view skimmers in the system sorted by their date found, either ascending or descending.
(b) The system shows a list of skimmers sorted according to their date found, in descending or ascending order.

**Extensions:**

*The system has no skimmers registered yet*

(a) The user requests a list of sorted skimmers but the system has no skimmers.
(b) The system displays a message indicating there are no skimmers.

10. **Visualize skimmers filtered by the security on the gas pump:**

Primary stakeholder: law enforcement agent

Precondition: the user is signed in

Trigger: a user wants to view skimmers in the system filtered by the security on the gas pump

**Principal scenario:**

(a) The user indicates it wants to view skimmers in the system filtered by the type of security found on the gas pump.
(b) The system shows a list of skimmers filtered by the type of security on the gas pump.

**Extensions:**
The system has no skimmers registered yet

(a) The user requests a list of filtered skimmers but the system has no skimmers.
(b) The system displays a message indicating there are no skimmers.

11. Perform analysis on a skimmer’s pictures:

Primary stakeholder: law enforcement agent

Precondition: the user is signed in

Trigger: a skimmer is added with images of its PCB

Principal scenario:

(a) The user adds a skimmer to the system (see requirement 1).
(b) The system analyzes the image/s against each other skimmer image using the similarity algorithm.
(c) For each image pair, the system registers their percentage match.

12. Visualize a skimmer’s data:

Primary stakeholder: law enforcement agent

Precondition: the user is signed in

Trigger: a user wants to view a certain skimmer’s data

Principal scenario:

(a) The system displays a list of skimmers (as in requirements 8 and 9).
(b) The user indicates it wants to view a skimmer from the list in detail
(c) The system displays the skimmer’s details, including its images, its location on an interactive map and any matches with other skimmers, if there are any.

Extensions:

The system has no skimmers registered yet

(a) The user requests a list of skimmers but the system has no skimmers.
(b) The system displays a message indicating there are no skimmers.
13. **Visualize a set of skimmers on a map:**

Primary stakeholder: law enforcement agent

Precondition: the user is signed in

Trigger: a user wants to view a set of skimmers overlaid over a map

**Principal scenario:**

(a) The user indicates it wants to view a set of skimmers in the system with their location overlaid on a map.

(b) The system shows a list of skimmers sorted by their proximity to the user, from which the user selects a subset.

(c) The system displays a map with the selected skimmers overlaid in their respective locations.

**Extensions:**

*The system has no skimmers registered yet* [same as with requirement 11]

*The system is unable to determine the user’s location* [same as with requirement 8]

### 4.3.3 Non-functional requirements

**Look and Feel Requirements**

- **10a. Appearance Requirements**

  **Description:** The product needs to appear pleasant and professional to the eyes.

  **Motivation:** As a system to be used by professionals as part of their jobs, the product they will use must have a professional look that does not distract from its purpose and does not put itself in front of the product’s functionalities. That being said, the final appearance must not be too monotone or rigid so that it also makes using the product a hard task.

  **Fit criterion:** The law enforcement agents find the appearance of the product to be simple, pleasant and professional so that it is easy to use.
Usability and Humanity Requirements

- **11a. Ease of Use Requirement: Efficiency of use**
  
  **Description:** The product has a high efficiency of usage for its users.
  
  **Motivation:** Law enforcement agents are very busy people, as we saw for ourselves during our meetings. As such, the time they dedicate to their jobs must be highly efficient. Using a product that is not up to par with their standards would not only make them dissatisfied with our product but also reduce the productivity of the rest of their work. Furthermore, with the product being an added part of their workflow, it will need to occupy that space as efficiently as possible.
  
  **Fit criterion:** The law enforcement agents find the product to have a high efficiency of usage.

- **11a. Ease of Use Requirement: Ease of remembering**
  
  **Description:** The product’s usage is easy to remember for its users.
  
  **Motivation:** Following the argument made in the previous requirement, a product that is not easy to remember would imply a reduced productivity for the users due to having to relearn some features every time they use the product.
  
  **Fit criterion:** The law enforcement agents find the product easy to remember in its usage.

- **11a. Ease of Use Requirement: Error rates**
  
  **Description:** The users commit a low number of errors while using the product.
  
  **Motivation:** As with the previous requirements, lowering the number of errors committed by the agents will mean an increased productivity in using the product.
  
  **Fit criterion:** The law enforcement agents find that the product makes them commit few errors while using it.

- **11a. Ease of Use Requirement: Feedback**
  
  **Description:** The product gives the user an adequate amount of feedback.
**Motivation:** It is important that the user knows what is happening with the system and he never gets disoriented with it. Regular feedback from the product will ensure that the user is never confused with the system and has ways of navigating through it and of fixing his mistakes.

**Fit criterion:** The law enforcement agents find that the product gives them the right amount of feedback about what the product is doing.

- **11b. Personalization and Internationalization Requirements: Language and units**

  **Description:** The product uses all units in the customary U.S. system and the English language throughout.

  **Motivation:** As explained in 2.5 the product will only be used by American officers, and as such the product will need to have its units in the customary way and use the English language throughout, with the necessary particular terms.

- **11b. Personalization and Internationalization Requirements: Filtering and sorting choices**

  **Description:** The product will remember the user’s choices in the way the skimmers are sorted in the skimmer list display.

  **Motivation:** Keeping the user’s choice of sorting of the skimmer list display will help in increasing productivity, even in the slightest.

- **11c. Learning Requirements**

  **Description:** The product shall be easy to learn for any law enforcement agent familiar with skimmers.

  **Motivation:** The product needs to be easy to learn for all users so that they can get it up and running quickly. A short tutorial will be included in the welcome page for new users so that they can quickly learn the basics of the system.

  **Fit criterion:** A user will be able to add their first skimmer within 10 minutes of opening the product.
4.3. Characteristics of the system

- 11d. Understandability and Politeness Requirements

Description: The skimmer matches shall be expressed in a percentage that is self-explanatory and does not reveal the internals of the product.

Motivation: Since the users of the product will not be experts in computer vision, to make it easy for them to grasp the nature of a match between skimmers a simple percentage will be used to represent it. In case they find it too simplistic, in the future it could be replaced with an in depth overview of similarities.

- 11e. Accessibility Requirements

Description: The product shall conform to the Americans with Disabilities Act.

Motivation: To account for any sort of disability that might limit the user when using the product, we will conform to the ADA so that these users can also use the product.

Performance Requirements

- 12a. Speed and Latency Requirements: List of skimmers

Description: The list of skimmers shall be displayed in less than 2 seconds, filtered or not.

Motivation: Given the modern day response times of websites, our product will need to be up to par with them. We find a response time of less than 2 seconds to be fast enough so as not to hamper the productivity of the user.

- 12a. Speed and Latency Requirements: Skimmer match background task

Description: The product shall provide an update every second for the progress of the background analysis task for a skimmer.

Motivation: The task of analyzing the pictures provided by the user could potentially be long, especially after a large number of skimmers gets added to it. That’s why it is important to keep the user updated with the progress of the analysis.

- 12c. Precision or Accuracy Requirements: Skimmer image match precision
Description: The product shall provide an accuracy of 75% in matching skimmers to similar ones.

Motivation: Matching two skimmers based on an image of their PCB is a hard task, and that’s why we need to set the right expectations for it. We do not expect the accuracy to be this high at the end of this project, but it shall be later on.

- 12d. Reliability and Availability Requirements

Description: The product shall be available for use 24 hours per day, 365 days per year.

Motivation: Each law enforcement agency uses a different schedule, and some might even be on different time zones. To account for such differences we will need for the product to be available at all times of the year.

- 12e. Robustness or Fault-Tolerance Requirements: Server failure

Description: The frontend server shall continue working in case of a failure of the backend server.

Motivation: In case the backend server were to break down, the frontend server needs to stay alive so that users can still see the skimmers in the system. Any new skimmer will be put on a queue to be analyzed once the backend server is up running again.

- 12e. Robustness or Fault-Tolerance Requirements: Backup and data loss

Description: A backup of the entire system shall be kept and updated regularly.

Motivation: If the system were to break down and its data wiped, we would lose all of it. That is why a regularly updated backup of the system will be kept. This way we can restore the system in case of data loss.

- 12f. Capacity Requirements

Description: The product can handle up to 100 simultaneous users.

Motivation: Our proposed product is not intended for use by a large number of users, as such our requirement for concurrent users using it is relatively low.
4.3. Characteristics of the system

- **12g. Scalability or Extensibility Requirements**

  **Description:** The product shall be able to handle an expansion to new regions of the U.S.

  **Motivation:** The initial scope of the project is small, limiting the system to law enforcement agencies in Colorado, but we must account for a possible expansion, which would entail a growth in the number of users.

- **12h. Longevity Requirements**

  **Description:** The product shall be expected to operate for at least a year after the project’s end.

  **Motivation:** One of the objectives of the project is to explore how and if this system will be useful to the agents. That’s why after completion of the project we expect a short running time where the product shall be used by the agents. Afterwards, the product shall be evaluated and its future assessed.

Operational and Environmental Requirements

- **13b. Requirements for Interfacing with Adjacent Systems: Browser versions**

  **Description:** The products shall work on the last ten releases of the four most popular browsers.

  **Motivation:** Each system in the law enforcement offices will likely use a different browser and because of that our product needs to be compatible with as many as possible to reduce setup times and the need for support.

  **Fit criterion:** The compatibility will be tested among the five most popular browsers currently.

Maintainability and Support Requirements

- **14a. Maintenance Requirements: Extensibility**

  **Description:** The product shall be easily extendable by developers other than the main developer.
Motivation: If the product were to expand and grow, new developers would need to be brought in. That’s why their potential job of expanding the product will need to be made easy through extensible code.

• 14a. Maintenance Requirements: Testability

Description: The product shall be easily tested by automatic and manual tests.

Motivation: To ensure a working product during all phases of development, the product will need to be made available for testing of any kind.

• 14b. Supportability Requirements

Description: A support email shall be made available for any kind of feedback or help the users might need.

Motivation: It is highly important that the users have a communication channel from where they can obtain any help if requested, as well as send their feedback. Without it they would be left alone to face any challenges the product might present them.

Security Requirements

• 15a. Access Requirements: Law enforcement agents

Description: The main users of the product (law enforcement agents) shall be able to view and add skimmers, as well as delete or edit those skimmers added by someone else from their agency.

Motivation: To keep users from tampering with the data in the system, only a user will be able to edit or delete his respective skimmers.

• 15a. Access Requirements: Administrator users

Description: Administrator users shall be able to perform all actions that regular users can do, as well as add, edit and delete law enforcement agencies.

Motivation: To keep users from tampering with the data in the system, only an administrator user will be able to add, edit and delete law enforcement agencies.
4.3. Characteristics of the system

- **15b. Integrity Requirements: Invalid data**
  
  **Description:** The product shall prevent incorrect data from being introduced.
  
  **Motivation:** All modern techniques of data verification will be used by the product to ensure the data in the system is correct. Furthermore, regular checks of the data will be performed to check for any incorrect or malicious data.

- **15c. Privacy Requirements: User data policy**
  
  **Description:** The product shall not keep or share any data on the system other than that necessary to the application.
  
  **Motivation:** Our product is not designed to keep any highly private information about the users. That being said, the data that is collected, such as emails, will never be shared with any third party. Furthermore, all the skimmer data will never be shared outside of the system. It is important to make the user aware of this and any future change to the data policy.

- **17a. Privacy Requirements: Data encryption**
  
  **Description:** Password data shall be encrypted throughout the system.
  
  **Motivation:** Modern security standards in software require that all passwords throughout the system are encrypted so as to protect the user and the system itself.

- **15e. Immunity Requirements**
  
  **Description:** The product shall run on a secure environment that can fend off any sort of malicious attack.
  
  **Motivation:** Nowadays, with our ever growing environment of malicious attacks against software products, it is paramount that our system can handle any possible attack and is ready to be upgraded for new threats that might appear in the future.

**Legal Requirements**

- **17a. Compliance Requirements: Skimmer data**
Description: The skimmer data shall comply with all laws that require it.

Motivation: Since skimmers are evidence in police cases, the legislation surrounding them varies across states or counties. That is why arrangements will have to be made to accommodate for any sort of requisite that a new law enforcement agency in the system might need, for example.

4.4 Conceptual schema

We now present the conceptual schema of the system, where we represent the object classes of the system. For each object class we can see its attributes, the relations between classes and its restrictions.

Agent:
Represents the main user of the system, has a username, a name, an email and if it is an administrator user. It is identified by its username and belongs to a police department.

Police Department:
Identified by its city, has an address, a phone number and an email. Has many agents associated
to it.

**Gas-pump skimmer:**
The principal object of the system, has a location, an address (for the sake of redundancy),
the type of security on the pump, the date it was found and an optional description. It is
associated to the agent that added it to the system, and in turn to a single police department.
Has a composition of images associated to it.

**Image:**
Each image belongs to a single gas-pump skimmer. Has many matches to other skimmer images.

**Match:**
Match between two skimmer images, represented by its percentage.

**TypesOfSecurityOnPump:**
Enumeration of the types of security that can be found on gas pumps.

**Integrity restrictions**

- Primary keys: (Police Department, City), (Agent, Username), (Gas-pump skimmer, id),
  (Image, id + Gas-pump skimmer::id), (Match, image1::id + image2::id)
- The *Address* of a Police Department must be in the city that identifies it.
- The *Email Address* of a Police Department must be unique.
- The *DateFound* of a skimmer cannot be greater than the present date.
- The *Location* and *Address* fields of a skimmer must refer to the same geographical position.
- There cannot be a match between images of the same skimmer.
- An Image cannot be matched against itself.
- A Match *percentage* must be between 0 and 100.
5. Software design

In this section we will go over the design aspects of the product, explaining the two parts of the system and the image analysis algorithm.

5.1 Patterns and principles

Before proceeding with describing the design aspects of the system we need to explain a couple of patterns and principles used throughout its design.

Separation of concerns

This software design principle states that a system’s parts should each have one concern [38]. This principle is also concerned with the way a system is organized and the implications its organization might have in future changes to it. With a good separation of concerns, modifying a large system becomes an easier task.

Model-View-Controller

The Model-View-Controller (MVC) pattern was introduced in 1988 for the language Smalltalk-80 as part of its interface design paradigm [36], [32]. The basis of this pattern is the division of the application logic into three parts:

- **Model**: responsible for handling the data and state of the application.
- **View**: manages the graphical output of the application.
- **Controller**: handles the user commands and interacts with the model and the view to convey the user’s intended actions.
5.2. Architecture

In a typical application we will have multiple models, views and controllers, with each model representing a class from the conceptual schema, and subsequent controllers and views for each model.

**Active Record**

An architecture pattern (on the basis of Object-Oriented Programming) that establishes that objects in the system carry both data and behavior which operates on such data \[51\]. This pattern helps in simplifying access to the database.

5.2 Architecture

As we described in section 2.3 our system will consist of two main parts, a *frontend* server and a *backend* server. The *frontend* Ruby on Rails server will be the main point of interaction for the user and where most of the data will be stored. The images will be stored on the Azure Blob Storage service. The *backend* Flask server will serve as a image analysis task runner and storage for the results of the image analysis. We can see a diagram of the described structure below in figure 5.1.

![System architecture diagram](image)

**Figure 5.1: System architecture**

This separation of responsibilities is not gratuitous. It stems from the need of building a scalable system (as mentioned in the first functional objective in 4.2.3) and following the separation of concerns principle. Having the image analysis server as a separate service will mean that the *frontend* server will be able to handle a higher amount of users all while processing images in
the background.

We will now move on to describe each part of the system in more detail.

### 5.2.1 Frontend

For the *frontend* of the system we will be using Ruby on Rails, a web development framework ideal for building simple applications like Skimmer Tracker [60]. As the name implies, the framework runs on the Ruby language, a high-level language that enables the paradigms of functional and object-oriented programming [61]. This framework relies heavily on the Model-View-Controller pattern previously described. While the MVC pattern wasn’t originally designed for web applications it has been adapted to be used in them and has proved to be very good at that. The view will represent the web page, the controller will act as the server handling user requests and the model will be interacting with the database and keeping the application state.

The *frontend* will be tasked with storing all classes from the conceptual schema (section [4.4]) except for the image matches, as that will be relegated to the *backend*.

**Access control**

To ensure our system is robust against various sorts of malicious attacks, we will implement the following restrictions.

- All users’ emails will be checked to be of the same domain as the email from their police department. Their email will also be verified to check if the person registering it is the owner of it. This will work by sending them a link after registration which after being clicked will activate the user. A non-activated user will be unable to log into the system, this way only police agents and detectives from a police department will be able to access it.
- Only the creator of a skimmer will be able to edit or delete it. This way, in case of a malicious agent getting access to the system as a user, he will not be able to tamper with the data in the system. This was one of the non-functional requirements described in section [4.3.3] of type 15a.
5.2. Architecture

- The system will have admin users, which will be able to manage police departments and to manage other non-admin users. These users are not intended to be owned by police agents, but rather by external trusted actors. That is why they are not allowed to modify skimmers. This was one of the non-functional requirements described in section 4.3.3 of type 15a.

Interface

The design of the system will need to accommodate to the usability and appearance non-functional requirements established in section 4.3.3. The final design reflects these requirements by staying simple throughout, all without being hard on the eyes. In figure 5.2 we can see the interface of the homepage for logged in users.

![Figure 5.2: Homepage with skimmer feed](image)

At the top of the page we find the navigation bar (figure 5.2), with links to various pages and the user icon, which lets users log out. We then have the title of the page and two buttons, one for adding a skimmer and the other to open the site guide. Finally below we can see two main columns for the skimmer feed, the first for the filtering options and the second for the list of skimmers, with two in each row. For each skimmer we can see their first image, location, description, types of security on the pump and the police department it is stored in.
For the skimmer page we opt for a two column design (figure 5.3), one for the images and information about the skimmer and another for a map of its location. Below that we can find a gallery of the matches with other skimmers, just one in this case.

All views in the site will be adaptive to different screen sizes. In figure 5.4 we have the skimmer page on a narrow screen, where the two column design shifts to a single column one. The TailwindCSS styling framework was used for all views in the system [69].

5.2.2 Backend

For the backend of the system we will use the Flask web framework [72], based on the highly popular Python programming language [54]. The principal reason for this choice is that with it
we will need to execute the image analysis algorithm. As we will see in section 5.3, the choice for running the algorithm will be MATLAB, which is quite trivial to execute from Python. This is another reason for separating the system in two parts; running a MATLAB program from Rails would be cumbersome and impractical.

As for the sort of web service that the backend will be, the choice is for it to be a RESTful API. Using the newer GraphQL was considered as an option and briefly tested, but discarded due to the risk of spending too much time learning the technology. Furthermore, coupling Flask with the Flask-RESTX extension library presents many advantages, such as automatic OpenAPI documentation generation with Swagger, which in turn provides automatic client code generation [26], [8]. OpenAPI is a standard for describing REST APIs, and Swagger provides an interface for testing documentation directly and generating client code [49].

We will follow a similar pattern to MVC for organizing the Flask-RESTX code, removing the views and adding services that handle the database queries. The controller will be tasked with serving directly the "views", which in this case will be JSON responses.

The backend will be tasked with keeping record of skimmer image matches, which means it will also need to store basic info about skimmers and images (more specifically the id of the
Section 5. Software design

skimmers and their images with their respective URLs). This gives us a duplication of data that will need to be managed accordingly. To make sure there are no data inconsistencies throughout the system the frontend will be responsible for communicating any changes to the backend. We will use the popular SQLAlchemy library as an ORM to map the API data to an SQL database [68]. SQLAlchemy is database agnostic, which means that the type of DBMS we end up using will not matter as long as it shares the data schema with the API.

The final part of our backend architecture will be that tasked with running the image analysis algorithm. Some of our defined requirements from section 4.3.3 mentioned the need for the background task to not interrupt the user’s workflow as well as report the progress in real time. For that purpose we will be using the Celery task runner [15], which integrates very well with Flask. Celery requires a message broker to store task information, for that we will be using the Redis in-memory database, which requires almost no configuration [56]. In figure 5.5 we can see a simple schema of the backend architecture.

![Backend architecture](image)

Figure 5.5: Backend architecture

Endpoints

In the following list we see the endpoints the API will have. They cover the basic needs of adding the images and getting the matches (if any) and the status of the analysis task. They also are on the second level of the Richardson Maturity level, being a CRUD API [57].

- **GET /skimmer/:** returns a list of all the skimmers with their images and their matches.
5.3. Computer vision

In section 2.3 we described the proposed system and explained how our product will rely on computer vision techniques to perform analysis on the skimmer images.

First, we will briefly discuss how exactly we are going to tackle this problem. The first necessary task that was considered in analyzing skimmers was to detect them inside an image. This way users of the system would not have to worry about editing the image to only display the PCB. Modern computer vision techniques have become highly efficient at tasks such as detection, but...
detecting PCBs has not been a concern. Still, those new techniques were developed so that they could be easily extendable. One of the more recent object detection solutions is Detectron, which can be easily extended with a dataset of the object to detect [23]. The dataset choice was the one from Kuo et al. [37]. In figure 5.6 we can see the Detectron results of an image of skimmer PCBs, with a 98% match in the middle one.

![Figure 5.6: Segmentation results from Detectron. Source: [27](image)](image)

We soon realized that refining the results of this method would require a longer project time than the available one, which made us turn to other computer vision techniques.

**Feature extraction and matching**

One of the more useful techniques from computer vision is feature extraction (or detection) and matching, which dates back to the 1980s (with inventions such as Moravec’s [46]). Feature extraction methods work by analyzing different aspects of an image, such as edges, corners or blobs. Once these aspects have been examined, we can build features from these aspects. Features constitute the most “interesting” parts of the image, a “measurable piece of data in your image which is unique to this specific object” as Elgendy describes it [22]; and the task of feature extraction algorithms is to determine a feature’s importance. In the case of a skimmer’s PCB the extracted features will consist of the corners of components, connection pins and the corners of the circuits.

Given two PCB images and their extracted features we can then compare them and match
them, thus giving us a reasonable measure of the similarity between two skimmer’s PCBs. This method might sound basic and naive, but it is the basis of the most advanced algorithms in use nowadays. It certainly comes with its tradeoffs (more on that in section 5.3.3), but it should prove enough to identify if two skimmers are the same. Furthermore, since feature extraction methods were started more than 30 years ago they are very fast to run on modern computers.

5.3.1 Alternative analysis

Now that we have explained the method that we will undertake to compare skimmer PCBs we will describe some of the available options of feature detection algorithms and then the environment where they can be run.

- **Harris**: developed in 1988 by Chris Harris, this corner detection algorithm was a one of the first attempts at feature detection [29]. It was an improvement over the previous Moravec corner detector [46] to establish itself as the standard in corner detection.

- **SIFT**: presented in 199 by David Lowe, the SIFT algorithm was developed for object recognition, but has proved valuable for many other applications that require feature detection. It is invariant to translation, scaling and rotation [42].

- **SURF**: first developed in 2006, this feature detection algorithm was inspired by SIFT and proved to be faster at its job [12].

- **BRISK**: presented in 2011 as a faster alternative to SIFT and SURF [40].

- **ORB**: based on the feature descriptor BRIEF, ORB also presented itself as an alternative to SIFT and SURF, with an added noise invariance [59].

- **KAZE**: a feature detection algorithm developed in 2011 [4].

All of these algorithms have been implemented in various programming languages and frameworks, most notably in Python with OpenCV [50] and in MATLAB with the Computer Vision Toolbox [45], [17]. The writer is familiar with using MATLAB and the Toolbox and thus that will be the language of choice.

We will now compare and benchmark the presented methods using said Toolbox. Note that the SIFT algorithm is not implemented in MATLAB, that is why we will be using a third-party
Section 5. Software design

implementation [7]. The images used for the benchmark can be found in [5.7]. They are of two instances of the same skimmer, with different orientation and focus.

![Image](image1.jpg)

![Image](image2.jpg)

Figure 5.7: Images used for benchmarking each feature matching method. Source: [27]

In table 5.1 below we show the results of running each of the described methods. For each we find and match the features found on the images in figure 5.7. In the table we can see for each method the number of matched features, the number of relevant features found on each image, the percentage of matched features for the first image and the time it took to analyze and match both images. In the final two columns we simply state if the algorithm is invariant to scale and rotation, respectively.

As we can observe, the SIFT method is one of the fastest and the most efficient in matching features, narrowly ahead of the SIFT method, which despite having more matched features is exceedingly more slow. This difference is due to the improvements made by the SURF method and the third-party implementation of the SIFT method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Num. Matched Ft.</th>
<th>Num Ft. 1</th>
<th>Num Ft. 2</th>
<th>Percent</th>
<th>Time</th>
<th>Inv. Scale</th>
<th>Inv. Rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harris</td>
<td>1</td>
<td>803</td>
<td>1052</td>
<td>0%</td>
<td>0.12s</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SIFT</td>
<td>22</td>
<td>1153</td>
<td>1913</td>
<td>1.9%</td>
<td>22.54s</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SURF</td>
<td>18</td>
<td>611</td>
<td>835</td>
<td>2.95%</td>
<td>0.75s</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BRISK</td>
<td>0</td>
<td>471</td>
<td>1500</td>
<td>0%</td>
<td>2.74s</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ORB</td>
<td>4</td>
<td>3931</td>
<td>7300</td>
<td>0.1%</td>
<td>2.74s</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>KAZE</td>
<td>1</td>
<td>2365</td>
<td>3410</td>
<td>0.04%</td>
<td>2.2s</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.1: Benchmark results

We will use the SURF method in our implementation of the product as it is the best alternative.
5.3.2 Pre-processing

Image pre-processing is the task of modifying an image to make it more suitable for analyzing it. It is highly important that when analyzing images that will be later matched some sort of pre-processing is performed before. The types of pre-processing available are quite literally endless, but we will only perform the following two for our images.

- Convert the image to grayscale (from RGB): there have been instances of feature detection algorithms being RGB compatible, but these implementations are fringe and not available to the public. In all the algorithms presented above the processing is done with grayscale images, so a conversion step will be performed on all images before being analyzed for features.

- Sharpen to highlight borders: to offset possible distortions or blurriness in the skimmer images provided to the system we will sharpen the edges of the skimmer. MATLAB uses the method of unsharp masking, which subtracts a blurred version of the image to itself. This helps make the job for the algorithm easier. In table 5.2 we can see an increase of 1% in the number of matched features for the SURF algorithm, as well as a decrease in execution time.

<table>
<thead>
<tr>
<th></th>
<th>Num. Matched Ft.</th>
<th>Num Ft. 1</th>
<th>Num Ft. 2</th>
<th>Percent</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harris</td>
<td>0</td>
<td>762</td>
<td>1113</td>
<td>0%</td>
<td>0.16s</td>
</tr>
<tr>
<td>SIFT</td>
<td>22</td>
<td>1153</td>
<td>1913</td>
<td>1.9%</td>
<td>8.9s</td>
</tr>
<tr>
<td>SURF</td>
<td>21</td>
<td>658</td>
<td>915</td>
<td>3.19%</td>
<td>0.12s</td>
</tr>
<tr>
<td>BRISK</td>
<td>0</td>
<td>687</td>
<td>2086</td>
<td>0%</td>
<td>0.7s</td>
</tr>
<tr>
<td>ORB</td>
<td>2</td>
<td>4774</td>
<td>8672</td>
<td>0.04%</td>
<td>0.57s</td>
</tr>
<tr>
<td>KAZE</td>
<td>1</td>
<td>2515</td>
<td>3589</td>
<td>0.04%</td>
<td>0.7s</td>
</tr>
</tbody>
</table>

Table 5.2: Benchmark results with added sharpening

5.3.3 Limitations

Finally we list the limitations using the SURF algorithm for feature matching will give us.

- Skimmer PCB analysis has not been part of law enforcement’s strategy in pursuing cases, which means there are not many publicly-available skimmer PCB images. The evidence
that was provided to us by Parker PD included multiple shots of the skimmer, but none of its PCB. Without more evidence our algorithm will suffer from overfitting. To counteract that we will have to use publicly-available PCB image datasets for further tuning.

- Given two skimmer PCBs with the same components but with a different component distribution our algorithm will most likely identify them as being the same. This independence of component organization is unlikely to happen but is still a limitation of the system.

- As explained in the previous section, we will have to convert the images to grayscale to perform analysis. This opens up the possibility of the algorithm mistaking two PCBs with the same components but with different colors to be the same. Given that most PCBs are green, it is a unlikely risk we are willing to take.

- Our algorithm does not perform any detection to ensure that the image contains a PCB. That means we will need to rely on the goodwill of the agents to crop and rotate the image to fit the PCB.

### 5.4 System Workflow

To end this section we will take a look at how a normal workflow of the system would look like, with screenshots of the system to accompany it.

Upon entering the website users are greeted with a welcome message and two buttons, one for registering and the other for logging in.

![Welcome to Skimmer Tracker](image)

Upon clicking on the "Sign up" button users are taken to the sign up form, as we can see in figure already filled out correctly.
Once the user submits the form the activation mail is sent and the user is told to check his email.

In the email there is a link that once clicked activates the account.

**Skimmer Tracker**

Hi John Appleseed,

Welcome to Skimmer Tracker! Click on the link below to activate your account:

 Activate

Now the user is able to access the system. He can, for example, decide to add a skimmer by clicking the corresponding button.
Clicking the button takes the user to the skimmer form.

The form requires a short description, the date the skimmer was found, its images, the address where it was located and the types of security its gas-pump had. For the images to be analyzed successfully they need to be previously cropped and rotated accordingly to only display the PCB. Once filled we submit it and we get taken to the newly created skimmer’s page. There we can see a purple bar that reports on the progress of the analysis task.
When the task finishes the page reloads and shows us if there were any matches. We can see that a match was found with a skimmer 1.5km from the one we just added.
6. Implementation

In this section we will go over the implementation aspects of the system in its two main parts.

6.1 Frontend

I was familiar with the Ruby language and the Ruby on Rails framework, but to get up to speed with the latest techniques I used the Ruby on Rails book by Michael Hartl\cite{30} as a starting point. Following it was not only helpful in getting a lot of experience with Rails but it was especially helpful for getting the system for handling account activation and recovery features. The Ruby version used is 2.7.0 and the Rails version 6.0.3.

The first step in implementing the *frontend* is to define the database schema, according to the conceptual schema created in section 4.4. And as we explained in section 5.2.1 the *frontend* will keep record of all the classes in the schema except for the image matches. Rails uses database migrations to progressively describe the schema. Migrations make up a timeline of changes that can later be navigated to, moving to a previous status of the schema. I started with the Police Departments and users (law enforcement agents) and increasingly added skimmers and their images to the schema. In figure 6.1 we can see the final database schema in the ERD format. The two *ActiveStorage* classes represent skimmer images, this implementation is the one Rails uses for storing attachments of any type (more on it later).

This schema can be plugged into any kind of DBMS thanks to Rails’ ActiveRecord ORM\footnote{Object Relational Mapping} which maps our objects on the system to tables on a relational database\cite{3}. For implementing the system we will be using a simple SQLite database for that purpose, which is a fast and a small. In case the system were to ever be put in production we would have to use a more
robust DBMS such as PostgreSQL or MySQL.

We will now describe the relevant parts of our implementation, starting with the models, controllers and views.

**Description of models**

- **police_dept.rb**: this model represents Police Departments, it validates that its city is present and the email address is unique, and it creates the has_many relationship with the user model.

- **skimmer.rb**: represents Skimmers and so it is the biggest model out of the three in terms of lines of code. It includes validations such as ensuring the date_found is not future, ensuring the coordinate fields are valid and that the description is no longer than 250 characters. It also defines the four possible types of security on pump for the field security_on_pump. As for the relationships for the skimmer class, the model establishes the belongs_to relation with the user model and the special has_many_attached relation for attached images. We also verify that the attached images are of a valid image format and no more than 10MB each. Finally, we define all possible filters and sorting options for skimmers with the Filterrific library (more on it later).
Section 6. Implementation

• **user.rb**: the user model creates the `belongs_to` relation with the police department model and a `has_many` relation with the skimmer model. It is also tasked with ensuring the email attribute is an actual email that belongs to the same domain as the associated police department’s email and is unique, and to ensure the password is between a certain length and gets stored encrypted. Finally, the model contains various methods to handle account activation, password reset and session storage.

**Description of controllers**

• **account_activations_controller.rb**: this controller has a single method that is executed when users click the link sent to their email on account creation and activates their user. Without having their accounts activated users can’t access the platform.

• **password_resets_controller.rb**: this controller handles password reset requests. When users request that their password be reset, provided their email address, an email is sent with a link that enables the user to change his password.

• **police_depts_controller.rb**: to access all actions in this controller the currently logged in user must be administrator. It contains all the standard CRUD operations to manage police departments in the system.

• **sessions_controller.rb**: handles all the operations related to user sessions, its creation and deletion. The creation method ensures the provided email and password match one of the users in the system and then creates the session.

• **skimmers_controller.rb**: this controller handles the CRUD operations for skimmers, all of which require the user to be signed in. The creation method also sends the images to the API to start their analysis and the `show` method updates the status of the analysis task.

• **static_pages_controller.rb**: this controller simply serves the static pages for the site (Home, Map, Help, About and Privacy Policy). For the homepage it also applies the filters and sorting requested in case the user is logged in, and in the map page it also provides the coordinates for the selected skimmers to be displayed in the map.

---

2Create, Read, Update, Delete
• **users_controller.rb**: a simple controller that again handles CRUD operations for users. It loads a user’s skimmers when viewing their profile and sends the activation email after creating the user as well.

**Description of view types**

- **layouts**: contains all views that are used in all other views, such as the site header, footer and navbar.
- **password_resets**: contains the two forms needed to perform a password reset. The first for requesting the reset by providing an email and the second for providing the new password.
- **police_depts**: contains all the views for managing police departments, only accessible to administrator users. Most important are the form view for both creating and updating police departments and the index view for displaying a list of all the departments.
- **sessions**: contains the log in form with the email, password and remember me fields.
- **shared**: contains some views that are either used by multiple other views or simply don’t belong in any of the other types of views. These include the skimmer feed, the map, the list of matches and the tutorial modal.
- **skimmers**: contains all views for skimmers, including its creation and editing form and the display view, which is responsible for not only showing all information about the skimmer but also its matches, its location and the live updates on the analysis progress.
- **static_pages**: contains the static views for the Home, Map, About, Help and Privacy Policy pages. All are accessible without being signed in the system, except for the map view which is simply hidden and the home view which displays different content depending if the user is signed in or not. If the user is signed in in the home view we display a feed of skimmers and the described filtering options. In the map view we simply show a map in which multiple skimmers can be displayed together, as chosen by the user.
- **user_mailer**: contains the email views for the activation and password reset services.
- **users**: contains all the necessary views for user management.
Libraries used

**Algolia Places**: service for implementing autocomplete of addresses into a website input [5]. Used in both the skimmer and police department forms. The service automatically provides us with coordinates for the inserted address, which makes displaying markers on the map trivial.

**Kaminari**: Ruby gem[^3] that provides pagination for any sort of model and the necessary view elements to add page numbering to the parameters sent to the controllers [34]. In our system we use it for limiting the number of skimmers per page in the home feed. We can see its use in code listing 6.1, where the user requested page is selected. *params* is the Rails object accessible from all controllers that contains the user parameters, in this case we are accessing the page parameter.

```
@feed_items = Skimmers.page(params[:page])
```

**ActiveFlag**: Rails has no native support for enumeration fields in which multiple values are possible at the same time. The ActiveFlag gem provides this functionality with some added helper methods for using those values in forms [21].

**Image Processing**: this gem uses the vips library to provide image transformations inside views [44], [41], with it we reduce load times by decreasing the size of images before sending them.

**Leaflet**: this popular library makes adding maps to a website trivial [39]. It also includes functionality for adding markers and shapes over the map. We use it in the skimmer view and in the map page with its Rails package [33].

**Rack CORS**: this gem adds CORS compatibility to our system, which allows us to perform cross-origin requests such as those necessary for displaying Leaflet maps [74].

**Dynamic Form**: this simple gem adds functionality to improve native display of form errors [47].

**Active Storage and Azure**: Active Storage is the included Rails gem that handles file storage and modifies the schema like we saw in figure 6.1 with both ActiveStorage classes. The Attachment class is polymorphic and that enables it to associate with any class type. Active

[^3]: Ruby packages are also known as gems ([https://rubygems.org/](https://rubygems.org/))
Storage is able to store the files in various services, such as the disk itself or Azure Storage. Uploading to a remote service is performed as a background task and so the users perceive no performance cost. To use the Azure storage service we need to use the `azure-storage` gem. 

**Filterrific**: this gem adds live filtering and sorting of Rails models. Filterrific works with Rails scopes, which are database queries defined in a model, and adds live reloading of the view. It is mainly used in the home view to sort and filter skimmers, where we have the filtering and sorting options as described in the functional requirements (section 4.3.2). We also use it in the map page to select which skimmers to display on the map.

**Testing libraries**

- **Minitest**: the main testing framework we will use for running unit and integration tests against the frontend.
- **Faker**: Ruby gem that generates fake random data such as names, emails or addresses, very useful for testing.
- **Simplecov**: this gem generates reports for code coverage of the tests, needed for fulfilling the objective from section 4.2.3.

**Communication with the backend API**

To communicate with the backend the Swagger API client code generator was used. With it we obtain a Ruby package that can be used very easily to perform API calls, such as the one in code listing 6.2. In it we simply initialize an instance of the client with which we call the function `get_a_skimmer_analysis_status` with an `id` parameter of a skimmer. This will automatically call the skimmer status backend API endpoint, parse the response and raise an error in case something fails.

### 6.2 Backend

The basic building block of the backend is the Flask framework in conjunction with Flask-RESTX. The resulting API shares some aspects with the MVC structure of the frontend as we explained in section 5.2.2 while being a fundamentally different system. We will now describe
Code listing 6.2. Code using the auto-generated swagger API client

```ruby
api_instance = SwaggerClient::SkimmerApi.new
begin
  result = api_instance.get_a_skimmer_analysis_status(id)
  return result
rescue SwaggerClient::ApiError => e
  puts "Exception when calling ImageApi->add_a_new_skimmer_image: #{e}"
  return false
end
```

details the most notable parts of its implementation.

Models and database

The **backend** is responsible for storing the image matches, for which it also needs to store their associated images and skimmers. In section 5.2.2 we established the use of SQLAlchemy as the ORM. SQLAlchemy is not as straightforward as Rails’ ActiveRecord and needs a manually defined database with the desired schema. We can see such schema in code listing 6.3, with the three relevant integrity restrictions in it:

- “There cannot be a match between images of the same skimmer”, ensured with this constraint: `skimmerID1 != skimmerID2`.
- “An Image cannot be matched against itself”, which we check with the following constraint: `imageID1 || skimmerID1 != imageID2 || skimmerID2`. The double vertical bar is an SQL operator for concatenating strings, and since all ids are strings in the database we can ensure that no same skimmer will be matched.
- “A Match percentage must be between 0 and 100”, checked with `percentage >= 0 AND percentage <= 100`.

With this schema defined upon a DBMS (SQLite in this case) we can then extract the SQLAlchemy models with the tool **flask-sqlacodegen** [66], which defines all the columns and constraints automatically. It is then necessary to manually add the proper serializing to convert the data to JSON with their associated sub-objects. With it when a skimmer is requested we will also
give its images, and in turn its matches. This reduces the number of requests the API will need handle.

Code listing 6.3. SQL database creation statements for the backend

```sql
CREATE TABLE Skimmer (  
id varchar(255) NOT NULL PRIMARY KEY,  
analyzed BOOLEAN NOT NULL DEFAULT 0  
);

CREATE TABLE Image (  
id varchar(255) NOT NULL,  
imgURL varchar(255) NOT NULL UNIQUE,  
skimmerID varchar(255) NOT NULL,  
FOREIGN KEY (skimmerID) REFERENCES Skimmer(id),  
PRIMARY KEY (id, skimmerID)  
);

CREATE TABLE ImageMatch (  
imageID1 varchar(255) NOT NULL,  
skimmerID1 varchar(255) NOT NULL,  
imageID2 varchar(255) NOT NULL,  
skimmerID2 varchar(255) NOT NULL,  
percentage INTEGER CHECK (percentage >= 0 AND percentage <= 100),  
FOREIGN KEY (imageID1, skimmerID1) REFERENCES Image(id, skimmerID),  
FOREIGN KEY (imageID2, skimmerID2) REFERENCES Image(id, skimmerID),  
CHECK (imageID1 || skimmerID1 != imageID2 || skimmerID2),  
CHECK (skimmerID1 != skimmerID2),  
PRIMARY KEY (imageID1, skimmerID1, imageID2, skimmerID2)  
);
```

Flask-RESTX introduces the use of DTO\footnote{Data Transfer Objects} for response marshalling \cite{52}. The aim of response marshalling is to verify the data that goes through the API (both in and out) follows a specified schema, which we define with DTOs. In the API we will have a DTO for each model, used for outwards validation (on GET requests), another for skimmer image input parameters and one for status requests.
Description of controllers

As with Rails, controllers are responsible for handling user requests. We have the following two in the backend.

- **skimmer_controller.py**: handles the endpoints for the root skimmer path (which returns all skimmers), the route for a single skimmer and the route for a skimmer’s analysis tasks status. Only the lat of the three presented some implementation challenges, since it must aggregate the status of all analysis tasks.
- **image_controller.py**: handles the endpoints for getting a skimmer’s images and adding one, for getting a specific image and deleting it, for getting its matches and for getting its task status.

Both controllers interact with the data through their respective services, **skimmer_service.py** and **image_service.py**. Services provide all the necessary methods for each endpoint to perform their respective data modifications on the database.

Swagger

Flask-RESTX provides functionality to interact with the API directly through the Swagger documentation. This is automatically generated with the following method decorators:

- **@api.marshal_with()**: takes a DTO model as a parameter, from which swagger gets the expected contents of the return object.
- **@api.expect()**: also takes a DTO model but for incoming parameters
- **@api.response()**: defines possible HTTP responses such as 200 for a successful response or 404 for a record not found error.

Background analysis task

Once skimmer images have been added to the backend a celery task is started to analyze the image against all other images from different skimmers. We execute a MATLAB function for each pair and put the results on the database. For the celery task to be able to access the
6.3 Image analysis algorithm

The implementation of the image analysis algorithm ended up being very simple in relation to the research and design time put on it. MATLAB includes all the necessary functions to perform feature extraction and matching with the SURF algorithm. From that it was only needed to add the two preprocessing steps: converting to grayscale with `rgb2gray` and sharpening the edges of the image with `imsharpen`.

6.4 Testing

In section [3.1] we explained the testing methodology we would take during the implementation of the system. We employed incremental testing throughout this phase in both the frontend and the backend. By incremental we mean that tests were added as features were implemented, and not all together at the end.

We previously described some of the frameworks used for testing the frontend, most notably minitest. With it we implemented a multitude of unit tests and integration tests. Unit tests are those that only test an individual module or part (models, controllers, helpers and mailers in our case) and integration those that concern themselves with testing how various parts of...
Section 6. Implementation

the system work together (log in, registering and adding a skimmer through the interface). The views are indirectly tested throughout all tests, but not explicitly.

With the simplecov gem we calculated the total code coverage of tests (the percentage of code lines executed by the test suite) which at the completion of the project rests at a high 78%.

The parts of the code not covered by the tests are those for the image upload to the backend and for applying some of the filters (which are activated from the interface). Both of these parts remaining to be tested require a different kind of testing from the rest of the code which is why we can say that a 78% code coverage is very good.

For the backend we implemented unit testing using the standard Python testing library unittest and flask_testing. With it we tested the controllers and the models.

6.5 Summary of used technologies

To finalize this section we will list the used technologies for designing and implementing Skimmer Tracker, excluding the already listed libraries.

- **Ruby on Rails**: main framework for the frontend, used on Ruby.
- **Flask and Flask-RESTX**: main frameworks for the backend REST API, written in Python.
- **GraphQL**: API framework that was used to probe the API, but was eventually discarded.
- **Swagger**: integrated with Flask-RESTX, contains many functionalities such as interactive OpenAPI documentation and API client code generation. Used both in the backend and the frontend.
- **SQLAlchemy**: database ORM for the backend.
- **Celery and Redis**: task runner and in-memory database, used in the backend to run image analysis tasks.
- **SQLite**: database management system used in both the frontend and backend for development purposes.
6.5. Summary of used technologies

- **MATLAB**: desktop computing environment and programming language. We used in this project to research, design and develop the image algorithm with the Computer Vision Toolbox extension [17].

- **Detectron**: object detection and segmentation framework, used to probe the possibility of detecting skimmers before analyzing them [23]. Its use was discarded due to complexity and time constraints.

- **Google Colab**: web-based Jupyter Notebook interface that uses Google’s servers to run notebook Python code. It was used in parallel with MATLAB to research and design the image algorithm.

The code for the entire implementation can be found on the following Github repository: [https://github.com/cudsl/skimmer-tracker/](https://github.com/cudsl/skimmer-tracker/) with the backend code on a submodule named img-server. It is under the MIT license, which allows for any sort of use with a copyright notice.
7. Time Planning

7.1 Project duration and deadlines

Before identifying the tasks performed during the development of this project we need to contextualize its duration by setting the start and end dates, among others.

- Start date: 21st of January. Upon my arrival to Boulder, CO on the Monday 20th of January the work immediately begins on Tuesday.
- End date: 22th of June. A week before the earliest possible defense date.
- Oral defense date: 29th of June. Earliest possible defense date, which could possibly be later, with the 3rd of July as the latest date.

This gives us a total of 110 work days (with weekends excluded), and with a total of about 536 hours that means a workload of 5 hours a day, approximately.

7.2 Description of tasks

- [T1] Project inception and context study

  First phase of the project, where an initial meeting is conducted and some research about the topic is performed.

  - [T1.1] Initial meeting with Project Director (1h): in this first meeting with Prof. Scaife a discussion of the Project will be performed, also consisting of an introductory seminar to the topic of the project, gas-pump skimmers. A weekly meeting will also be set up.
7.2. Description of tasks

– [T1.2] Research about project topic (20h): research performed after the meeting with the Project Director, it includes learning about skimmers themselves as well as looking for existing solutions and helpful tools for the development of the project. This should take no more than 4 hours each day of the week.

Estimated time: 21 hours in total

Deadline: since the Project will start on the week of the 20th of January, this task will have to be performed by the end of that week, January 24.

Resources needed: computer with Internet access, access to research papers and journals, text editor.

• [T2] Law enforcement interview (from January 21 to February 7)

Interviews with law enforcement will be conducted, in which skimmers will be discussed and the interviewed agents will be asked about the potential system. The following subtasks are sequential since each one is dependent on the previous one.

– [T2.1] Prepare questions for law enforcement (22h): task will include learning about requirements gathering so that questions provide value for the project.

– [T2.2] Interview with law enforcement agents (2h): the prepared questions are asked and the answers taken note of. Any extra information gathered will also be taken note of as it could provide unexpected value.

– [T2.3] Gather requirements from interview (6h): after each interview, a list of requirements for the system is written based on the answers from police agents.

Estimated time: in total, this task should take about 30 hours, 2 of which will be for the interviews and the rest will be office work.

Deadline: the interviews and subsequent requirement gathering should be performed by February 7.

Resources needed: computer with Internet access, text editor, academic book about Requirements Engineering, notebook, pen, car to go to meetings.
Section 7. Time Planning

• [T3] Project management (from February 10 to March 12)

Task that will consist of preparing different documents and defining different aspects of the project, as well as managing the project’s progress. The task has been divided into the following subtasks:

- [T3.1] Context and scope (20h): redaction of document that contextualizes the thesis and describes its scope. Objectives, stakeholders and a justification are described, among others.
- [T3.2] Time planning and task identification (15h): redaction of this same document, where tasks to be performed are identified and given a time estimate. The resulting estimate is visualized in a Gantt chart.
- [T3.3] Budget and sustainability (20h): redaction of an extensive cost analysis and a sustainability report.
- [T3.4] Weekly meeting with Project Director (10h): for 30 minutes in each of the 20 weeks of the project there is a meeting scheduled with the Project Director, where the status of the project and future work are discussed.

Estimated time: 65 hours in total

Deadline: the task should be completed by March 10, although task [T3.4] will run throughout the entirety of the project.

Resources needed: computer with text editor, meeting room

• [T4] Prior software research (from February 27)

Before undertaking any significant decision in the development of the project, a research step must be taken. It has been subdivided in the following three subtasks:

- [T4.1] Research frontend architectures (10h): available options for the frontend of the system are studied and compared, as well as testing techniques. Note that we use the term frontend loosely here, referring to the entire web application the users interact with.
7.2. Description of tasks

– [T4.2] Research backend architecture (6h): available options for the backend of the system are studied and compared, along with testing techniques. That does not include the image analysis algorithm, which belongs to the next subtask. The backend we refer to here encapsulates the system in charge of managing image storage and handling the execution of the image analysis algorithm.

– [T4.3] Research computer vision techniques (60h): possible options for the software and structure of the image analysis algorithm are researched, as well as considering if more modern techniques (such as CNNs) should be used over older techniques.

**Estimated time:** 76 hours

**Deadline:** This task as a whole should be completed before April, but each subtask is independent. More on that in tasks [T5] and [T6].

**Resources needed:** computer with internet access, text editor, MATLAB software for testing with computer vision techniques.

• [T5] Software design (from February 1 to May 10)

  – [T5.1] Design frontend architecture (30h): the system architecture is designed with a UML schema and other software architecture tools, adapting it to the chosen software and architecture. Definition of specific tasks for the implementation of this part of the system on the Kanban board.

  – [T5.2] Design backend architecture (10h): having chosen a software stack, a simple structure of the backend architecture is to be designed. We also set the specific subtasks for this part on the Kanban board.

  – [T5.3] Design image vision algorithm (15h): having chosen a method to use as the algorithm, a basic structure is devised for the algorithm.

**Estimated time:** 55 hours

**Deadline:** each subtask depends on the previous corresponding one ([T5.1] to [T4.1], etc...), with the frontend and backend starting earlier because of their shorter research times in contrast to the image vision algorithm.
Resources needed: computer with internet access, text editor, plotting/drawing software.

• **[T6]** Software implementation (from March 17 to June 12)
  
  – **[T6.1]** Frontend implementation (60h): based on the specified structure and tasks from [T5.1], the *frontend* system is implemented.
  
  – **[T6.2]** Backend implementation (45h): the *backend* system is implemented based on the tasks and structure defined on [T5.2].
  
  – **[T6.3]** Implement image vision algorithm (50h): having designed a basic structure for the algorithm in [T5.3], we implement it.

Estimated time: 155 hours in total

• **Deadline**: just like with the previous task, each subtask here depends on its corresponding previous subtask. As a whole this task’s deadline is about a week before the project’s end date, the 12th of June.

Resources needed: computer with internet access, code editor, terminal, MATLAB

• **[T7]** Validation, testing and showcase (from March 16 to June 12)
  
  – **[T7.1]** Frontend testing (20h): the *frontend* part of the system is thoroughly tested in all its components. This task can be done concurrently with **[T6.1]** since we are using the Kanban Agile methodology.
  
  – **[T7.2]** Backend testing (10h): the *backend* part of the system is tested in all its parts. Just like with [T7.1], we can conduct this subtask along with **[T6.2]** given the chosen Agile methodology.
  
  – **[T7.3]** Image algorithm testing (10h): the algorithm is tested against real world examples of skimmers and its effectiveness is asserted using advanced evaluation techniques. For this part of the system we are not using an Agile methodology and so testing should be performed after implementing the algorithm in **[T6.3]**.
7.2. Description of tasks

- **[T7.4]** Integration testing (5h): the main three components of the system are tested after being coupled together. This subtask depends on each of the previous implementation subtasks ([T6.1], [T6.2], [T6.3]).

**Estimated time:** 49 hours

**Deadline:** same as project’s end, June 22

**Resources needed:** computer with internet access, text editor, code editor with test runner, MATLAB, notebook, pen

- **[T8]** Documentation and presentation (limit June 22)

  - **[T8.1]** Documentation (70h): the project’s memory is written, with some parts of this document included in it with some extensions. The whole of [T.3] must be completed before this task gets started.

  - **[T8.2]** Presentation preparation and oral defense (20h): creation and writing of presentation, as well as rehearsing.

**Estimated time:** 90 hours

**Deadline:** 22nd of June

**Resources needed:** computer with text editor, presentation editor
### 7.3 Table of tasks

In table 7.1 below we can see a review of all the tasks presented with their total hours and dependencies.

<table>
<thead>
<tr>
<th>Task</th>
<th>Hours</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Project inception and context study</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>1.1 Initial meeting with Project Director</td>
<td>1</td>
<td>[T1.1]</td>
</tr>
<tr>
<td>1.2 Research about project topic</td>
<td>20</td>
<td>[T1.1]</td>
</tr>
<tr>
<td>2  Law enforcement interview</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2.1 Prepare questions for law enforcement</td>
<td>22</td>
<td>[T1.1]</td>
</tr>
<tr>
<td>2.2 Interview with law enforcement agents</td>
<td>2</td>
<td>[T2.1]</td>
</tr>
<tr>
<td>2.3 Gather requirements from interview</td>
<td>6</td>
<td>[T2.2]</td>
</tr>
<tr>
<td>3  Project management</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>3.1 Context and scope</td>
<td>20</td>
<td>[T2.3]</td>
</tr>
<tr>
<td>3.2 Time planning and task identification</td>
<td>15</td>
<td>[T3.1]</td>
</tr>
<tr>
<td>3.3 Budget and sustainability</td>
<td>20</td>
<td>[T3.2]</td>
</tr>
<tr>
<td>3.4 Weekly meeting with Project Director</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4  Prior software research</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>4.1 Research frontend architectures</td>
<td>10</td>
<td>[T2.3]</td>
</tr>
<tr>
<td>4.2 Research backend architecture</td>
<td>6</td>
<td>[T2.3]</td>
</tr>
<tr>
<td>4.3 Research computer vision techniques</td>
<td>60</td>
<td>[T1.2]</td>
</tr>
<tr>
<td>5  Software design</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>5.1 Design frontend architecture</td>
<td>30</td>
<td>[T4.1]</td>
</tr>
<tr>
<td>5.2 Design backend architecture</td>
<td>10</td>
<td>[T4.2]</td>
</tr>
<tr>
<td>5.3 Design image vision algorithm</td>
<td>15</td>
<td>[T4.3]</td>
</tr>
<tr>
<td>6  Software implementation</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>6.1 Frontend implementation</td>
<td>60</td>
<td>[T5.1]</td>
</tr>
<tr>
<td>6.2 Backend implementation</td>
<td>45</td>
<td>[T5.2]</td>
</tr>
<tr>
<td>6.3 Implement mage vision algorithm</td>
<td>50</td>
<td>[T5.3]</td>
</tr>
<tr>
<td>7  Validation, testing and showcase</td>
<td>45</td>
<td></td>
</tr>
</tbody>
</table>
### 7.4 Gantt table

<table>
<thead>
<tr>
<th></th>
<th>Task</th>
<th>Duration</th>
<th>Task Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td><em>Frontend</em> testing</td>
<td>20</td>
<td>[T5.1]</td>
</tr>
<tr>
<td>7.2</td>
<td><em>Backend</em> testing</td>
<td>10</td>
<td>[T5.2]</td>
</tr>
<tr>
<td>7.3</td>
<td>Image algorithm testing</td>
<td>10</td>
<td>[T6.3]</td>
</tr>
<tr>
<td>7.4</td>
<td>Integration testing</td>
<td>5</td>
<td>[T6]</td>
</tr>
<tr>
<td>8</td>
<td><strong>Documentation and presentation</strong></td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Documentation</td>
<td>70</td>
<td>[T3]</td>
</tr>
<tr>
<td>8.2</td>
<td>Presentation preparation and oral defense</td>
<td>20</td>
<td>[T8.1]</td>
</tr>
</tbody>
</table>

Table 7.1: Table of tasks
<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1.1</td>
<td>Initial meeting with Project Director</td>
</tr>
<tr>
<td>T1.2</td>
<td>Research about project topic</td>
</tr>
<tr>
<td>T2.1</td>
<td>Prepare questions for law enforcement</td>
</tr>
<tr>
<td>T2.2</td>
<td>Interview with law enforcement agents</td>
</tr>
<tr>
<td>T2.3</td>
<td>Gather requirements from interview</td>
</tr>
<tr>
<td>T3.1</td>
<td>Context and scope</td>
</tr>
<tr>
<td>T3.2</td>
<td>Time planning and task identification</td>
</tr>
<tr>
<td>T3.3</td>
<td>Budget and sustainability</td>
</tr>
<tr>
<td>T3.4</td>
<td>Weekly meeting with Project Director</td>
</tr>
<tr>
<td>T4.1</td>
<td>Research frontend architectures</td>
</tr>
<tr>
<td>T4.2</td>
<td>Research backend architecture</td>
</tr>
<tr>
<td>T4.3</td>
<td>Research computer vision techniques</td>
</tr>
<tr>
<td>T5.1</td>
<td>Design frontend architecture</td>
</tr>
<tr>
<td>T5.2</td>
<td>Design backend architecture</td>
</tr>
<tr>
<td>T5.3</td>
<td>Design image vision algorithm</td>
</tr>
<tr>
<td>T6.1</td>
<td>Frontend implementation</td>
</tr>
<tr>
<td>T6.2</td>
<td>Backend implementation</td>
</tr>
<tr>
<td>T6.3</td>
<td>Implement image vision algorithm</td>
</tr>
<tr>
<td>T7.1</td>
<td>Frontend testing</td>
</tr>
<tr>
<td>T7.2</td>
<td>Backend testing</td>
</tr>
<tr>
<td>T7.3</td>
<td>Image algorithm testing</td>
</tr>
<tr>
<td>T7.4</td>
<td>Integration testing</td>
</tr>
<tr>
<td>T8.1</td>
<td>Documentation</td>
</tr>
<tr>
<td>T8.2</td>
<td>Presentation preparation and oral defense</td>
</tr>
</tbody>
</table>
7.5 Risk management

- **Inexperience**: in section 2.4 we saw how inexperience in the field of computer vision could limit the project’s results. Computer vision learning resources will be undertaken in case it becomes apparent that a lack of experience in this field is missing. This would also mean an increase in the time dedicated to task [T4.3], increasing the project’s budget.

- **Time limitations**: in case the project ends up taking up more time than predicted then the only possible solution would be to reduce the scope of the project.

- **Work equipment breaks**: in any sort of project we must consider the possibility that the work equipment might break and make the project work stop. To account for this possible incidental we allocate a part of the budget for potential repair costs.

- **Not enough skimmers**: as explained in section 2.1 the number of skimmer cases is low on the local scale which could make the system useless in the eyes of police officers and limit the capabilities of the similarity algorithm. This could be solved with more meetings with law enforcement and by expanding the scope to include all nationwide law enforcement agencies, which would increase already existing travel costs.

7.6 Changes from the initial time planning

**COVID-19**

Due to the ongoing global pandemic caused by the COVID-19 virus, which has provoked the closing of all university facilities and the cancellation of all in-person meetings, the initial time planning has changed. This meant that the planned meetings with law enforcement to showcase the finalized project had to be canceled. This is reflected in the time planning with the removal of task [T7.5]. Secondly, with the university being closed down due to the virus, all work for the project had to move home, which caused a reduction in productivity that is luckily offset with the removal of task [T7.5] and the increase in available time for work. This is all reflected in the updated Gantt chart.

Unfortunately this risk was not considered at the start of the project and that’s why it caused an important change in the time planning of the project.
Inexperience about computer vision techniques

In section 2.4 we saw some of the risks that could hamper the project’s due process, and luckily we considered inexperience in computer vision techniques as one of them. The main cause of this was an extension of task [T4.3] and a reduction of task [T5.3]. This is also reflected in the Gantt chart.

Going in detail about what caused the delay, we must first briefly explain some of the history of computer vision. In the past decade Convolutional Neural Networks (CNNs) have risen in popularity for their uses in computer vision, and have proven to be very useful in detection and segmentation tasks which were previously harder or even impossible. Some older techniques from the decade of the 2000s, while not as effective as the new techniques at detection tasks, can be very effective at tackling problems of feature detection and matching. At the start of the project I identified a couple of research papers that used CNNs to identify and segment PCBs (43, 37), but using their tools meant learning how to use CNNs first. As I started learning these modern techniques I soon realized that using them in the project would be impossible without extending its duration. And that’s why I decided to revert back to using more older and basic techniques I was more familiar with.
8. Budget

In this section we will analyze all the costs related to the project, including those related to human work, other general costs and we will discuss added costs due to contingencies and risks. Since this project will be developed at Boulder, Colorado, it is only fitting that its costs adhere to the real values here in the US.

8.1 Identification of costs

8.1.1 Human resources

In Table 8.1 we can see the identified costs for each role in the project. We use the service Payscale to get the salaries here in Boulder. All salaries include income taxes.

<table>
<thead>
<tr>
<th>Role</th>
<th>Cost/hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project manager</td>
<td>$31.25</td>
</tr>
<tr>
<td>Software Architect</td>
<td>$44.00</td>
</tr>
<tr>
<td>Software Tester</td>
<td>$30.60</td>
</tr>
<tr>
<td>Software Developer</td>
<td>$40.00</td>
</tr>
<tr>
<td>Software Analyst</td>
<td>$27.99</td>
</tr>
<tr>
<td>Computer Vision Engineer</td>
<td>$40.36</td>
</tr>
</tbody>
</table>

Table 8.1: Table of roles and their salaries

After having identified all the roles, we go on to distribute the estimated hours for each of the roles. We distribute each task to one or more roles, depending on the requirements of the task. Meetings, for example, apply to more than one of the roles. We can observe the final distribution in Table 8.2.

Having completed the hour distribution we can now estimate the total cost for each role, as we can see in Table 8.3. Note how the total number of hours paid for is higher than the total
Table 8.2: Table of hour distribution for each role

amount of duration hours estimated since meeting tasks account for more than one person at the time.
8.1. Identification of costs

<table>
<thead>
<tr>
<th>Role</th>
<th>Cost/hour</th>
<th>Hours</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project manager</td>
<td>$31.25</td>
<td>92</td>
<td>$2,750.00</td>
</tr>
<tr>
<td>Software Architect</td>
<td>$44.00</td>
<td>48</td>
<td>$2,112.00</td>
</tr>
<tr>
<td>Software Tester</td>
<td>$30.60</td>
<td>38</td>
<td>$1,162.80</td>
</tr>
<tr>
<td>Software Developer</td>
<td>$40.00</td>
<td>100</td>
<td>$4,000.00</td>
</tr>
<tr>
<td>Software Analyst</td>
<td>$27.99</td>
<td>185</td>
<td>$5,207.03</td>
</tr>
<tr>
<td>Computer Vision Engineer</td>
<td>$40.36</td>
<td>125</td>
<td>$5,045.57</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>588</td>
<td><strong>$20,277.40</strong></td>
</tr>
</tbody>
</table>

Table 8.3: Table of roles with their total hours and cost

8.1.2 General costs

Hardware

We assume each member of the team already owns a computer from where to work from. We add 3 monitors for increased productivity. As for the server costs, Amazon Web Services’ calculator was used to estimate the running costs for two servers during the project’s duration. Costs for maintaining the service running after the end of the project will be estimated at a later time. We also introduce here the concept of amortization. We can calculate it with the formula in Figure 8.1, where \( \text{projectMonths} \) is the previously established 5 months of project duration.

\[
\text{Amortization} = \text{productPrice} \times \frac{\text{projectMonths}}{\text{productLifetime}}
\]

Figure 8.1: Amortization formula

In Table 8.4 below we can see the total hardware costs, with their associated amortization cost.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Price</th>
<th>Product lifetime (months)</th>
<th>Amortization</th>
</tr>
</thead>
<tbody>
<tr>
<td>3x Dell UltraSharp 24 Monitor</td>
<td>$669.72</td>
<td>120</td>
<td>$27.91</td>
</tr>
<tr>
<td>Kensington Keyboard</td>
<td>$19.99</td>
<td>36</td>
<td>$2.78</td>
</tr>
<tr>
<td>Frontend server costs</td>
<td>$100.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Backend server costs</td>
<td>$40.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$829.71</strong></td>
<td></td>
<td><strong>$30.68</strong></td>
</tr>
</tbody>
</table>

Table 8.4: Table of hardware costs with amortization
Section 8. Budget

Software

The only cost in terms of software will be that of a 1 year MATLAB License. This software will be used for developing the image algorithm. All other software used will be free of use.

<table>
<thead>
<tr>
<th>Software</th>
<th>Price</th>
<th>Product lifetime (months)</th>
<th>Amortization</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB License (student)</td>
<td>$250.00</td>
<td>12</td>
<td>$104.17</td>
</tr>
<tr>
<td>Total</td>
<td>$250.00</td>
<td></td>
<td>$104.17</td>
</tr>
</tbody>
</table>

Table 8.5: Table of software costs with amortization

Office space & meetings

As for the place where the project will be conducted, we have chosen Office Evolution’s offices here at Boulder [13]. A rent of one of their private offices comes with enough room for the 6 roles, includes 60$ meeting room credit per month as well as all electricity and internet costs. We can see the total cost for that in Table 8.6 below.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Price/month</th>
<th>Project duration (months)</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Office Space</td>
<td>$349.00</td>
<td>5</td>
<td>$1,745.00</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>$1,745.00</td>
</tr>
</tbody>
</table>

Table 8.6: Table of office costs

We now move on to calculate some of the costs attached to the meetings. Firstly, we have a total of 11 meeting hours over 5 months. Renting a meeting room for an hour is 60$, and with the 60$ credit per month that gives us 6h that we have to pay for. Secondly, for the travel for the meetings with law enforcement we need to calculate the gasoline cost. We assume an average of 3 gallons per hour and 2 hours of driving, with a cost of around 1.80$ per gallon [71]. We can find below, in Table 8.7 the total costs.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Price/hour</th>
<th>Hours</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meeting room</td>
<td>$60.00</td>
<td>6</td>
<td>$360.00</td>
</tr>
<tr>
<td>Car travel for law enforcement meetings</td>
<td>$5.40</td>
<td>1</td>
<td>$5.40</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>$365.40</td>
</tr>
</tbody>
</table>

Table 8.7: Table of meeting costs
8.1. Identification of costs

8.1.3 Incidentals and contingencies

Incidentals

We now move on to estimate the cost attached to some of the risks we considered in section 7.5. As we can see in Table 8.8 we added the cost of a possible computer repair.

<table>
<thead>
<tr>
<th>Incidentals</th>
<th>Cost</th>
<th>Risk</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop breaks, repair cost</td>
<td>$650.00</td>
<td>10.00%</td>
<td>$65.00</td>
</tr>
<tr>
<td>More meetings with law enf.</td>
<td>$5.40</td>
<td>20.00%</td>
<td>$1.08</td>
</tr>
<tr>
<td>Increased computer vision time (+20h)</td>
<td>$807.29</td>
<td>20.00%</td>
<td>$161.46</td>
</tr>
<tr>
<td>Total</td>
<td>$227.54</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.8: Table of incidental costs

Contingencies

Considering the possible risks is not enough when building a budget. That is why we add a safety net around each of the costs of the project. In our case we choose a 10% contingency rate, as we can see below in Table 8.9.

<table>
<thead>
<tr>
<th>Section</th>
<th>Cost</th>
<th>Contingency</th>
<th>Cost with added contingency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human cost</td>
<td>$20,374.41</td>
<td>10.00%</td>
<td>$22,305.14</td>
</tr>
<tr>
<td>Hardware</td>
<td>$860.39</td>
<td>10.00%</td>
<td>$946.43</td>
</tr>
<tr>
<td>Software</td>
<td>$354.17</td>
<td>10.00%</td>
<td>$389.58</td>
</tr>
<tr>
<td>Office space</td>
<td>$1,745.00</td>
<td>10.00%</td>
<td>$1,919.50</td>
</tr>
<tr>
<td>Meeting costs</td>
<td>$370.80</td>
<td>10.00%</td>
<td>$401.94</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>$25,962.60</td>
</tr>
</tbody>
</table>

Table 8.9: Table of contingencies

8.1.4 Total costs

Now that we have seen all the costs in our budget we can put them together to get the complete budget for the project.
### Table 8.11: Table of total costs

<table>
<thead>
<tr>
<th>Subject</th>
<th>Cost</th>
<th>Contingency</th>
<th>Final cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human cost</td>
<td>$20,277.40</td>
<td>10.00%</td>
<td>$22,305.14</td>
</tr>
<tr>
<td>Project manager</td>
<td>$31.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Architect</td>
<td>$44.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Tester</td>
<td>$30.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Developer</td>
<td>$40.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Analyst</td>
<td>$27.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer Vision Engineer</td>
<td>$40.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td>$860.39</td>
<td>10.00%</td>
<td>$946.43</td>
</tr>
<tr>
<td>3x Dell UltraSharp 24 Monitor - U2419H</td>
<td>$697.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kensington Keyboard</td>
<td>$22.77</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frontend server costs</td>
<td>$100.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backend server costs</td>
<td>$40.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>$354.17</td>
<td>10.00%</td>
<td>$389.58</td>
</tr>
<tr>
<td>MATLAB License (student)</td>
<td>$354.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Office space</td>
<td>$1,745.00</td>
<td>10.00%</td>
<td>$1,919.50</td>
</tr>
<tr>
<td>Shared Office Space</td>
<td>$1,745.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meeting costs</td>
<td>$370.80</td>
<td>10.00%</td>
<td>$401.94</td>
</tr>
<tr>
<td>Meeting room</td>
<td>$360.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Car travel for law enforcement meetings</td>
<td>$5.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incidentals</td>
<td>$227.54</td>
<td>-</td>
<td>$227.54</td>
</tr>
<tr>
<td>Laptop breaks, repair cost</td>
<td>$65.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>More meetings with law enf.</td>
<td>$1.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increased computer vision time (+20h)</td>
<td>$161.46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>$26,190.14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 8.2 Management control

Having drawn up the budget we now move on to describe how we will control the economic progression of the project. Even though we took into account potential risks in the development of the project and added some contingencies, it is still necessary to perform regular checks on the real costs of the project to account for any unexpected obstacle that might appear. That’s why a regular evaluation of the budget will be performed to see how the actual costs square up to the budgeted costs. We will use the following indicators to see how big is the offset from the planned costs with the actual costs:

- Labour deviation = (estimated cost − real cost) * total hours
• Task deviation = (estimated cost − real cost) * total hours

• Total deviation from task realization = total estimated task cost − total real task cost

• Total deviation of resources = total estimated resource cost − total real resource cost
9. Sustainability report

In this final section we will study the environmental, economic and social impact of the project. The analysis will follow the matrix of sustainability in figure 9.1.

<table>
<thead>
<tr>
<th>PPP</th>
<th>Exploitation</th>
<th>Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environmental</td>
<td>Consumption of the design</td>
<td>Ecological footprint</td>
</tr>
<tr>
<td>Economic</td>
<td>Invoice</td>
<td>Viability plan</td>
</tr>
<tr>
<td>Social</td>
<td>Personal impact</td>
<td>Social impact</td>
</tr>
</tbody>
</table>

Table 9.1: Matrix of sustainability

9.1 Environmental Dimension

Product put into production: Consumption of the design

In the budget we calculated a total of 588 hours of project work, spread across 6 people, each working on their own laptop.

We first calculate the energy cost associated to working on the project on a computer. We will take the 13 inch MacBook Pro from 2016 as a reference for all roles. It consumes 0.16W when off, 0.5W when idle and 2.97W under normal use [1]. With around 6h of work per day, 2h of idle work and 16h with the computer turned off the resulting cost per day is the following:

\[
6h \times 2.97W + 2h \times 0.5W + 16h \times 0.16W = 21.38Wh
\]

With a total of 110 days we have a total of 2352Wh computer consumption.

The other consumption to calculate is the one associated to the 3 monitors. The model we
considered consumes 24W which gives us 14112Wh for the 588h of project time [2]. Adding this to the previous cost we get a total of 16464Wh for the duration of the project.

**Exploitation: Ecological footprint**

Assuming a useful life of at least one year during a pilot program the costs of running said program would mostly depend on the used servers. We previously established we would use AWS servers for that purpose, but they do not disclose their power consumption figures. Still, cloud-based servers consume up to 84% less power than physical servers so we can say we are choosing the better option out of the available [16]. The only other consumption created during a pilot year would be from a single developer occasionally working on the system. As such, we will consider it to be a marginal and negligible cost.

**Risks**

The biggest risk in augmenting the environmental footprint of the system would be it needing more resources than originally planned. Also, in case its computer vision application were to shift from feature matching to the use of CNNs that would suppose a huge cost increment due to the power required to use the latter.

### 9.2 Economic Dimension

**Product put into production: Invoice**

In section 8 we studied the estimated budget for this project. We took into account human related costs, general costs (such as hardware, software and office related costs) with amortization included and finally incidentals and contingencies. Including these last two items gives us security in knowing that we have a safety net in case any of the risks were to happen. From the final results we can see how human related costs take the majority of the pie (almost 80%), which is very much expected in a Software Project. The workforce costs may seem exorbitant, but we assumed the project is being developed in Boulder, CO, where IT-related salaries are much higher than in Spain. The salaries include income taxes.
Exploitation: Viability plan

The running costs of the system would be those of the AWS server. We already established a running cost of the servers of $140 during the 5 months project duration, $28 per month. This would mean $336 in total for a year.

One thing we know about Graffiti Tracker is how they perform analysis on the Graffiti entered on the system; they do so manually with human analysts. Our proposed solution would do this automatically with computer vision techniques, and as we saw in our budget, human costs are way more expensive than machine costs. This means a significant reduction in costs by our proposed solution, especially during the useful life of the product; although not during the development phase of the project, where a computer vision engineer is needed in our case.

Risks

The economic risks for our system are very much tied to the environmental costs. An increase in resources needed would also mean an increase of running costs from the basis of $336.

9.3 Social Dimension

Product put into production: Personal impact

With this project I have set out to achieve the objectives described in section 4.2.4 and I can say I have fulfilled them. Furthermore, since the project is centered around credit card fraud and other similar crimes I have become more conscious about it and its ramifications and I have developed a better sense of justice.

Exploitation: Social impact

There is no solution at the present time that mirrors ours, which makes it very difficult to say whether the system will negatively impact any fragment of society. What we can say is that no existing job will be removed from the implementation of our solution; except, of course, that of the skimmer criminals.

It would be pretentious to claim that there is a clear need for our solution. But what’s clear is that there needs to be innovative solutions to counterattack the new techniques applied by
criminals in matters of card fraud with skimmers. With this project we hope to explore a possible solution to fight fraud, as well as to motivate others to help in this honorable fight.

Risks

As mentioned in the previous section, having no existing solution of reference with which to compare ours means, possible social risks caused by it would be bold speculation.
10. Conclusions

We have now reached the conclusion of this report, where we went over all aspects of the Skimmer Tracker project. Its future remains a question as to whether it ever gets used in the field. Not all the objectives we set for the project could be fulfilled due to various obstacles, such as obtaining a 75% accuracy in relating skimmer cases. But working on it was worth it nonetheless.

To finalize it, we will go over possible future work and analyze the completion of technical competences.

10.1 Future work

*Frontend*

- Currently the system for checking the status of a skimmers analysis tasks works by making recurrent AJAX calls to a controller. An alternative way of performing this check would be using Action Cable, Rails’ integration of WebSockets, with which getting status updates would be far more efficient.

- Some fringe cases in terms of moderation remain to be solved, mainly for ensuring skimmer images are up to standards and as skimmer archival (in case its case gets solved) and handling user removal.

- In section 6.4 we mentioned how not all parts of the code were being tested, and in improving the *frontend* they should be covered by tests. These included, firstly, interface testing, which can be implemented with libraries such as capybara, and secondly, integration testing with the *backend* API.
10.1. Future work

Backend API

The backend API could help with two main improvements:

- Make the execution of the MATLAB scripts be concurrent for the same image to reduce execution times.
- Increase testing capabilities by adding integration testing.

Computer vision

It is no secret that the strategy taken in analyzing and matching skimmer PCB images is rather basic. To improve matching of skimmers we propose a more modern approach that would greatly increase the fidelity of the analysis, while improving the workflow for users.

The first step in the process would consist of the already developed PCB detector, with the necessary improvements, using Detectron [23]. With the PCB segmented we would then apply the PCB component detection algorithm from Kuo et. al [37]. The algorithm detects components such as integrated circuits, resistors, capacitors or inductors. This information could be easily leveraged to compare skimmer based on these distinct features.

This approach would be based on Convolutional Neural Networks (CNNs), which need to be run on highly powerful computers and require specialized training. Using this improved approach would then incur an increase in development costs.

Showcase and pilot

The initial intention for the Skimmer Tracker system was for it to be showcased to the same law enforcement agents we initially talked to and gather feedback on the system. That was not possible due to the COVID-19 pandemic and means the system would need to be showcased by a third party in the future since I, the writer, will not be in the US once normal life resumes.
10.2 Completion of technical competences

CES1.1: Develop, maintain and evaluate complex and/or critical software systems and services

It could be said that this technical competence defines the project in itself. We developed and evaluated a software system that could prove to be critical if it ever were to be used by law enforcement.

CES1.3: Identify, evaluate and manage the potential risks that may arise associated with the construction of software

In sections 2.4 and 7.5 we outline and describe how we would manage risks that could appear during development.

CES1.5: Specify, design, implement and evaluate databases

In both main parts of the system we worked with a database, indirectly in the frontend and directly in the backend.

CES1.7: Control quality and design tests in software production

The project has been tested and evaluated for its quality during its entire development. Furthermore, we have implemented design patterns that improve code quality and structure and make it more expandable for potential future developers.

CES2.1: Define and manage the requirements of a software system

At the start of the project we interviewed two law enforcement agents and gathered requirements for the system. These were then turned into the specification we can see in section 4.
CES2.2: Design appropriate solutions in one or more application domains, using software engineering methods that integrate ethic, social, legal and economic aspects

Being a system intended for use by law enforcement the implications of storing evidence in the site had to be considered.
List of Figures

1.1 Some of the types of skimmers .................................................. 4
1.2 Gas pump with an included payment station. Source: Parker PD ........... 4
2.1 Proposed system architecture .................................................... 8
4.1 Use cases .................................................................................. 17
4.2 Conceptual data schema ............................................................ 32
5.1 System architecture ................................................................. 35
5.2 Homepage with skimmer feed .................................................... 37
5.3 Skimmer page ............................................................................ 38
5.4 Skimmer page on a narrow screen .............................................. 39
5.5 Backend architecture ............................................................... 40
5.6 Segmentation results from Detectron. Source: [27] ......................... 42
5.7 Images used for benchmarking each feature matching method. Source: [27] . . 44
6.1 Frontend physical schema, automatically generated with Rails ERD [55] .... 51
8.1 Amortization formula ............................................................... 75
List of Tables

5.1 Benchmark results ............................................................... 44
5.2 Benchmark results with added sharpening ............................. 45
7.1 Table of tasks ................................................................. 69
8.1 Table of roles and their salaries ......................................... 73
8.2 Table of hour distribution for each role ............................... 74
8.3 Table of roles with their total hours and cost ....................... 75
8.4 Table of hardware costs with amortization ......................... 75
8.5 Table of software costs with amortization ............................ 76
8.6 Table of office costs .......................................................... 76
8.7 Table of meeting costs ......................................................... 76
8.8 Table of incidental costs ...................................................... 77
8.9 Table of contingencies ......................................................... 77
8.11 Table of total costs .......................................................... 78
9.1 Matrix of sustainability ....................................................... 80
Bibliography


[57] Richardson Maturity Model. URL: https://martinfowler.com/articles/richardsonMaturityModel.html (visited on June 17, 2020).


BIBLIOGRAPHY


