Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informática – DAINF
Computer Engineering
Integration workshop 3 (EEX23) – S71 – 2021/2

Technical Report
FareSeer

André Otavio Pedrofeza de Oliveira – andreoliveira.1998@alunos.utfpr.edu.br
Arthur M. S. Oliveira – arthuroliveira@alunos.utfpr.edu.br
Bruno Guillen – brunoguillen@alunos.utfpr.edu.br
Felipe Alves Barboza – felipebarboza@alunos.utfpr.edu.br
Leonardo Muraroto de França Reis – leonardoreis@alunos.utfpr.edu.br
Lucas Felipe Ribeiro – lucasr.1996@alunos.utfpr.edu.br

December, 2021

Abstract

Our main motivation stems from a desire to improve existing public transport systems by integrating innovative technologies into the ticket gates found in bus stations. The traditional bus ticket payment process is prone to security breaches: if a passenger's magnetic card is stolen, or if a special benefits card is being used illegally, the transgressor can still use the card until it is manually blocked by a system administrator - a process that can greatly benefit from facial biometric technologies. Also, cameras present in bus stations are currently used for security reasons only: the camera output could provide valuable information for crowd management and logistics purposes. With this in mind, we have implemented a system which uses facial recognition technologies to facilitate the process of blocking magnetic cards; and also, by analyzing camera feed, we employed the use of crowd counting algorithms to estimate the amount of people inside a bus station at a given moment in time. The present document details the development of the aforementioned project.

1 Introduction

Nowadays, facial recognition technologies are being applied to many different situations for fraud prevention purposes, since they are both reliable and offer better means of identity verification than traditional methods. Recently, some major cities in Brazil, such as São Paulo[1] and Curitiba[2], have already started integrating these technologies into their public transportation systems, but there still is a long path ahead until most cities in Brazil could benefit from these systems. Moreover, the logistics of public transportation as a whole could be revolutionized through the adoption of crowd estimation algorithms: the video feed from security cameras present at bus stations could be analyzed to estimate the amount of people at a given location in time, providing an invaluable

source of information to logistics systems which aim to maximize the efficiency of resource allocation. With the crowd concentration data collected across time, robust logistic systems could predict situations that would increase the flow of passengers, enabling the preemptive dispatchment of vehicles to areas requiring them the most. In this manner, buses could have their itineraries optimized as a function of the concentration of people, potentially sparing resources spent on fuel and minimizing the time passengers spend on their daily commutes.

Having the aforestated scenario in mind, we have conceptualized *FareSeer*, a system that integrates both facial recognition and crowd counting functionalities into a turnstile, and also allows users to pay tickets with smartphone NFC technologies and RFID magnetic cards.

1.1 Project Overview

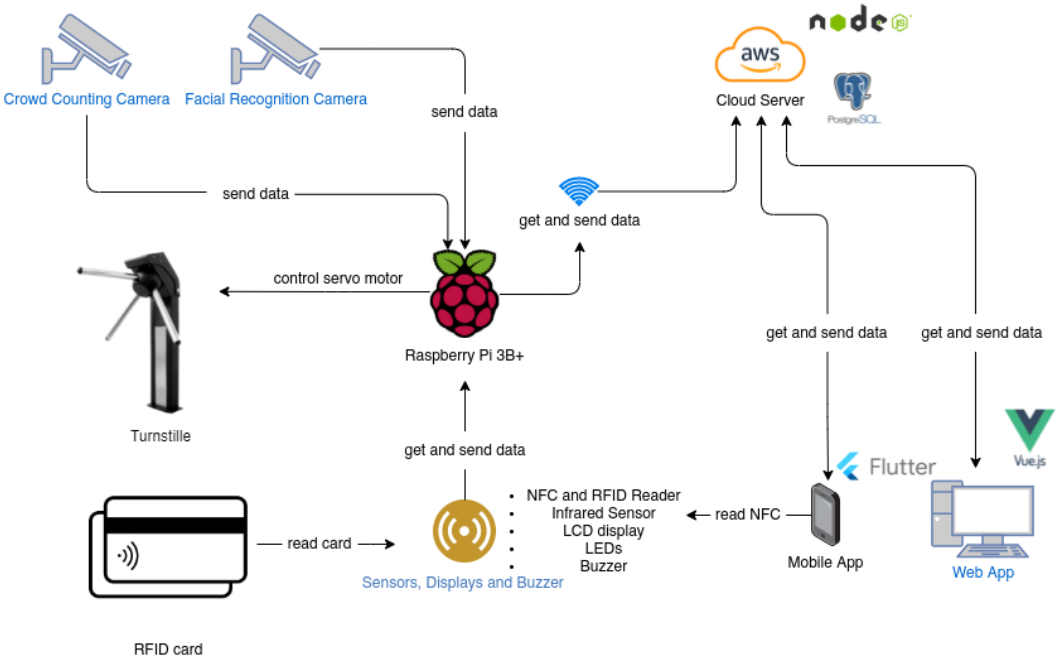

Figure 1: A block diagram of the system.

As depicted on Figure 1, the project is centered around a system embedded into a turnstile, consisting of a Raspberry Pi 3B+ connected to two cameras. A box, placed above the turnstile, houses the micro-controller, the facial recognition camera and an NFC/RFID reader. Also, to provide system status information, the box has an LCD display, LEDs and a buzzer attached to it. The turnstile locking mechanism is composed of a PVC cog placed above an infrared sensor which is used to determine when the turnstile has rotated. A servo-motor un-

locks the mechanism whenever the user is allowed entrance. The crowd counting camera is placed on the bus station's ceiling, and sends its feed to the micro-controller.

A Web application containing two different categories of users was also developed: regular users can register themselves, submit a picture of their faces, login and see their ticket payment history; admins can list the users registered on the platform, add credits to their accounts, and block their cards. Users can also pay the tickets with a mobile application that allows them to login, register and enable NFC communication. The Web application front-end and back-end components as well as are both hosted on the cloud.

Facial biometrics processing take place as soon as the user completes a successful payment transaction. Firstly, the system will attempt to find a face on the video feed provided by the facial recognition camera. If a face is found, the last picture taken will be sent to the cloud along with the payer's ID, where it will then be processed. The cloud API compares the image received with the one registered on the user's profile and returns the similarity, as well as the confidence levels of the comparison, to the micro-controller. Based on the API response, the micro-controller flags the user appropriately.

The pictures captured by the crowd counting camera are periodically uploaded to the cloud, where the counting operation takes place. The results of this process - both the bus station image and the estimated amount of people inside - are stored into the database and displayed to system administrators on the Web application.

2 Project Specifications

The following subsections detail the requirements, materials and tools employed in the development of our project.

2.1 Requirements

After having established the goals of the project, a requirements analysis was conducted. Four main areas of development were identified: Hardware and Mechanical Components, Web Application and Image Processing, Mobile Application, and Firmware. They were divided this way based on the interrelationships between the technologies they encompass. Tables 1 to 4 present the requirements of the project.

Table 1: Hardware Requirements

Hardware Functional Requirements	
H-FR01	The Hardware must receive and process signals from one camera for facial recognition purposes.

H-FR02	The Hardware must receive and process signals from one camera for crowd counting purposes.
H-FR03	The Hardware must connect to the Internet via Wi-Fi.
H-FR04	The Hardware must control the gate locking mechanism.
H-FR05	The Hardware must be able to read NFC and RFID frequencies.
Hardware Non-Functional Requirements	
H-NFR01	The System must be implemented with a Raspberry Pi 3B+.
H-NFR02	The System must have two USB cameras, one for facial recognition and another for crowd counting.
H-NFR03	The System must have an NFC/RFID reader module.
H-NFR04	The System must have an LCD display 16x2 to show operation status.
H-NFR05	The System must have a servomotor to lock and unlock the ticket gate.
H-NFR06	The System must have an infrared sensor to detect when the ticket gate moves.
H-NFR07	The System must have LEDs to show the operation status.
H-NFR08	The System must have a Buzzer that activates whenever the user is allowed entrance into the tube station.
H-NFR09	The mechanical structure must be constructed with wood and PVC pipes.

Table 2: Software (Web Application and Image Processing) Requirements

Web App and Image Processing Functional Requirements	
S-FR01	The web app must allow users to register themselves.
S-FR02	The web app must have two different categories of users: admins and regular users.
S-FR03	The web app must allow users to login and logout.
S-FR04	The web app must allow users to submit pictures of their face.
S-FR05	The web app must allow users to receive credits to pay tickets.
S-FR06	The web app front-end and back-end must be hosted on the Cloud.
S-FR07	The web app must display the user's history of ticket payments.
S-FR08	The web app must have an admin page that displays the last tube station image processed and the amount of people counted on that image.
S-FR09	The system must be able to recognize the user's face.
S-FR10	The system must be able to estimate the amount of people inside the tube station.

S-FR11	The system's database shall periodically be replicated locally, in order to keep the ticket gate working when there is no connection with the cloud systems.
S-FR12	The facial recognition system must flag users who attempt to use another passenger's card.
S-FR13	The admin page shall display the users flagged for suspicious activity.
S-FR14	The admin page shall display the users flagged for failing the facial recognition process.
S-FR15	The admin page shall allow admins to block flagged users.
Web App and Image Processing Non-Functional Requirements	
S-NFR01	The web app front-end must be developed with Vue.js.
S-NFR02	The web app back-end must be developed with Node.js.
S-NFR03	The database must be implemented with PostgreSQL.
S-NFR04	The cloud services must be implemented with AWS.
S-NFR05	The image processing shall be conducted using Cloud Computer Vision Services and IOT Edge technologies.
S-NFR06	The facial image recognition and crowd estimation shall be implemented with Tensorflow CNNs and Cloud Cognitive Services.
S-NFR07	The admin page shall display a warning when the ticket gate is not connected to the cloud server.

Table 3: Software (Mobile Application) Requirements

Mobile App Functional Requirements	
M-FR01	The mobile app must allow users to register themselves.
M-FR02	The mobile app must allow users to login and logout.
M-FR03	The mobile app must allow users to submit pictures of their face.
M-FR04	The mobile app must connect with the system via Wi-Fi.
M-FR05	The mobile app must allow the user to pay tickets via NFC..
M-FR06	The mobile app must have access to the web app content.
Mobile App Non-Functional Requirements	
M-NFR01	The Application must be developed with Flutter SDK.
M-NFR02	The Application must be developed with Dart.

Table 4: Firmware Requirements

Firmware Functional Requirements	
F-FR01	The Firmware must connect to the cloud server.
F-FR02	The Firmware must periodically receive images from the crowd counting camera and send them to the server.
F-FR03	The Firmware must receive images from the facial recognition camera after the user activates the NFC/RFID module.
F-FR04	The Firmware must notify the user whether facial detection succeeds or fails.
F-FR05	The Firmware must notify when the user doesn't have funds.
F-FR06	The Firmware must notify when the user can pass through the ticket gate.
F-FR07	The Firmware must notify the user when he has to look at the camera.
F-FR08	The Firmware must read the infrared sensor to know when the ticket gate moves and lock it again.
F-FR09	The Firmware must be able to lock and unlock the ticket gate.
F-FR10	The Firmware must determine if the image from the facial recognition camera contains a face.
F-FR11	The face detected closest to the camera shall be the one used for the facial recognition purposes.
F-FR12	If the facial detection fails after a certain amount of attempts, the user shall be granted entrance through the ticket gate (provided they have enough funds), flagged for suspicious activity and the last picture taken will be sent to the cloud for the image processing.
Firmware Non-Functional Requirements	
F-NFR01	The Firmware must be developed with Python.
F-NFR02	The Firmware must be implemented in a Raspberry Pi 3B+.
F-NFR03	The Firmware must be able to send JSON data.
F-NFR04	The Firmware must be able to receive JSON data.
F-NFR05	The Firmware must show the notifications through the LCD display.
F-NFR06	The Firmware must control a servomotor to lock and unlock the ticket gate.

2.2 Materials and Tools

The subsections below describe the materials and tools employed in the construction of our project.

2.2.1 Mechanical Structure

The mechanical structure was assembled with 6mm thick MDF boards. Wooden broom handles where used to create the turnstile's rotating tripod component, and the locking mechanism was created with a PVC cog and MDF locking pins.

A ready-made PVC box was acquired to house the electronic components and the facial recognition camera. It's dimensions are: $20x16x8cm$.

2.2.2 Electronic Components

The embedded systems consists of a Raspberry Pi micro-computer that receives and processes signals from the sensors and cameras. A PN532 NFC reader module is used for the smartphone and RFID card recognition system. The locking mechanism is composed of a Servomotor MG995, and an LM393 infrared sensor is used to detect when the mechanism‘s cog has rotated. A $16x2$ LCD display, red and green LEDs and a buzzer are used to display the system‘s status operation. Two USB webcams are used for image capturing.

2.2.3 Cloud Services

The Web App‘s front-end and back-end, along with our database, are hosted on the cloud using Amazon Web Services (AWS) [3], s3 buckets were used for storing registered images for facial comparison. Amazon Rekognition API was used to conduct the image processing. The following were the API modules used: *DetectFaces* was used for comparing the registered image on the s3 bucket with the user's ID and *DetectLabels* was used for crowd counting, by selecting the Person tags on the image analysis response.

2.2.4 Web Application

The Web Application's front-end was developed with *Vue.js*, an open-source JavaScript framework that enables the creation of sophisticated Single-Page applcations [4]. To create the user interface, the Materials Design components from *Vuetify*, a UI framework for *Vue.js*, were utilized [5]. The back-end was developed with *Node.js*, also an open-source framework for JavaScript.

2.2.5 Mobile Application

Flutter and *Dart* were used to develop the Mobile Application. *Flutter* is a cross-platform, open-source UI toolkit created by Google.[6]. *Dart* is an object-oriented programming language designed for the development of mobile and web apps.[7]

3 Development

This section describes the development process of our project. The exact functioning and flow of operation of all the elements are further detailed.

3.1 Project Management

Project management was based on the agile method *Scrumban* using *Flying-Dount.io* as our Dashboard. The team had daily meetings, where each member reported how their assigned tasks were going on Sprint and Deliverable. Our schedule worked as the backlog for the project and our Professors feedbacks were considered for the validation of our tasks. It means our *Epic* was our project and each *Project Iteration* were the Deliverable's assigned by them.

3.2 Mechanical Structure

Figure 2: The assembled mechanical structure.

The mechanical structure was first sketched after a real sized turnstile. After some calculations, the ideal dimensions were established. It was then modeled using the software *SketchUp* for a visualization and a proof of concept. Afterwards, the individual pieces were ordered and crafted by a woodworker. Finally (depicted on Figure 2), the structure was assembled with glue and nails.

The locking mechanism was also created based on similar ones found in existing turnstiles [8]. It consists of a turnplate (cog) locked in place by moving pins placed on its sides.

3.3 Hardware and Firmware

First, we tested all of our electronic components individually to ensure that they were all working as expected and were not defective. That done, we made the electronic project.

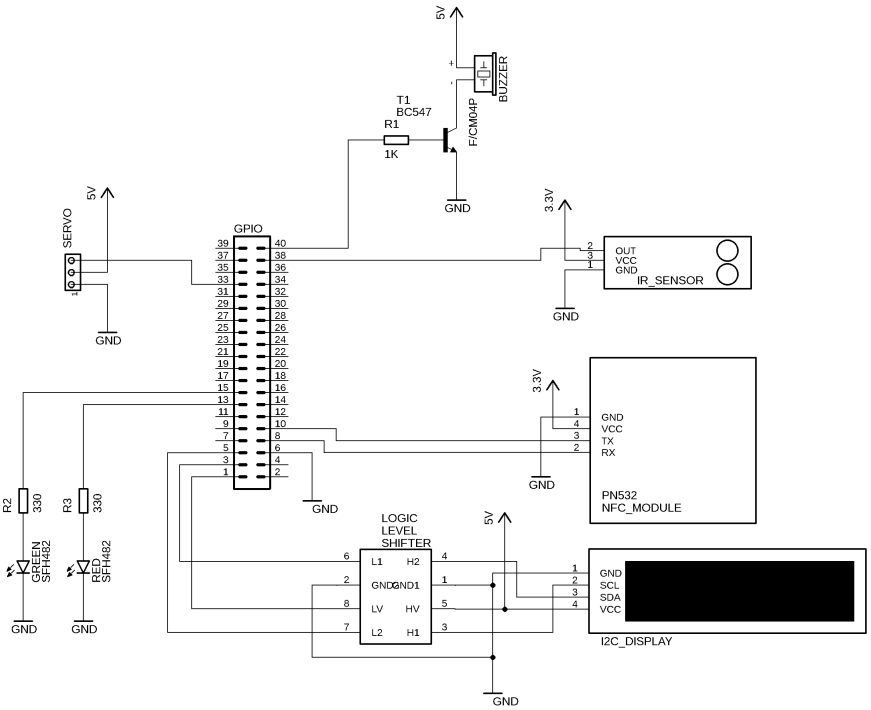

Figure 3: A schematic of the electronics

With the electronics project in hand, we assembled it on a breadboard and tested all the components working together. Finally, we performed the final assembly on a standard board. The NFC reader module is powered by an external 3.3v source and communicates with the Raspberry Pi via UART. The 16x2 LCD display communicates via I2C. As the display operates on a voltage of 5v, we used a logic level converter from 5v to 3.3v to allow connection to the Raspberry.

The buzzer used for the sound signal is of the active type, and it is powered by an external 5v source. We used an NPN transistor to drive it through the GPIO. LEDs are connected directly to the GPIO using resistors. The servo motor that controls the turnstile lock is powered by an external 5v source and controlled by the Raspberry's dedicated PWM generator hardware. The infrared sensor used to detect when the engine has rotated is powered by an external 3.3v source and is directly connected to the GPIO. In addition, we have two USB cameras connected to the Raspberry, one is used for facial recognition and the other for crowd counting.

Facial detection was implemented with *Histogram of Oriented Gradients* (HOG) [9], an object detector algorithm. Experimentation with different algorithms, such as Haar Cascade [10], has shown that HOG was the best option for our needs, since it can effectively detect faces on the images and runs well on a Raspberry.

As soon as users approximate their cards or smartphones to the NFC reader, the algorithm will attempt to find a face on images captured from the facial recognition camera. In attempting to find a face, a total of 10 pictures will be taken, each at $50ms$ intervals; HOG spends around $60ms$ to process the image, so each iteration takes $110ms$. If no face is found after all these attempts, the last image taken will be the one sent to the cloud for facial recognition purposes (As depicted on Figure 4). Amazon's Rekognition API will then compare the image received with the one registered on the user's profile and return the similarity and confidence levels of the comparison. After receiving the cloud's response, the firmware will determine if any suspicious activity took place. If the similarity values were below 50% with a confidence level of over 90%, the user will be tagged for fraudulent activity. If no face was found on the image, or the similarity values were between 50-70% with a confidence level of over 90%, the user will be tagged for suspicious activity.

Every minute, a picture of the passengers inside the bus station is taken by the crowd counting camera and also sent to the cloud (Figure 4), where the image will be processed and the estimated number of people in it will be registered on the database. The pictures are stored in a AWS S3 Bucket, a service that allows you to upload a file and it returns a link that can be accessed later.

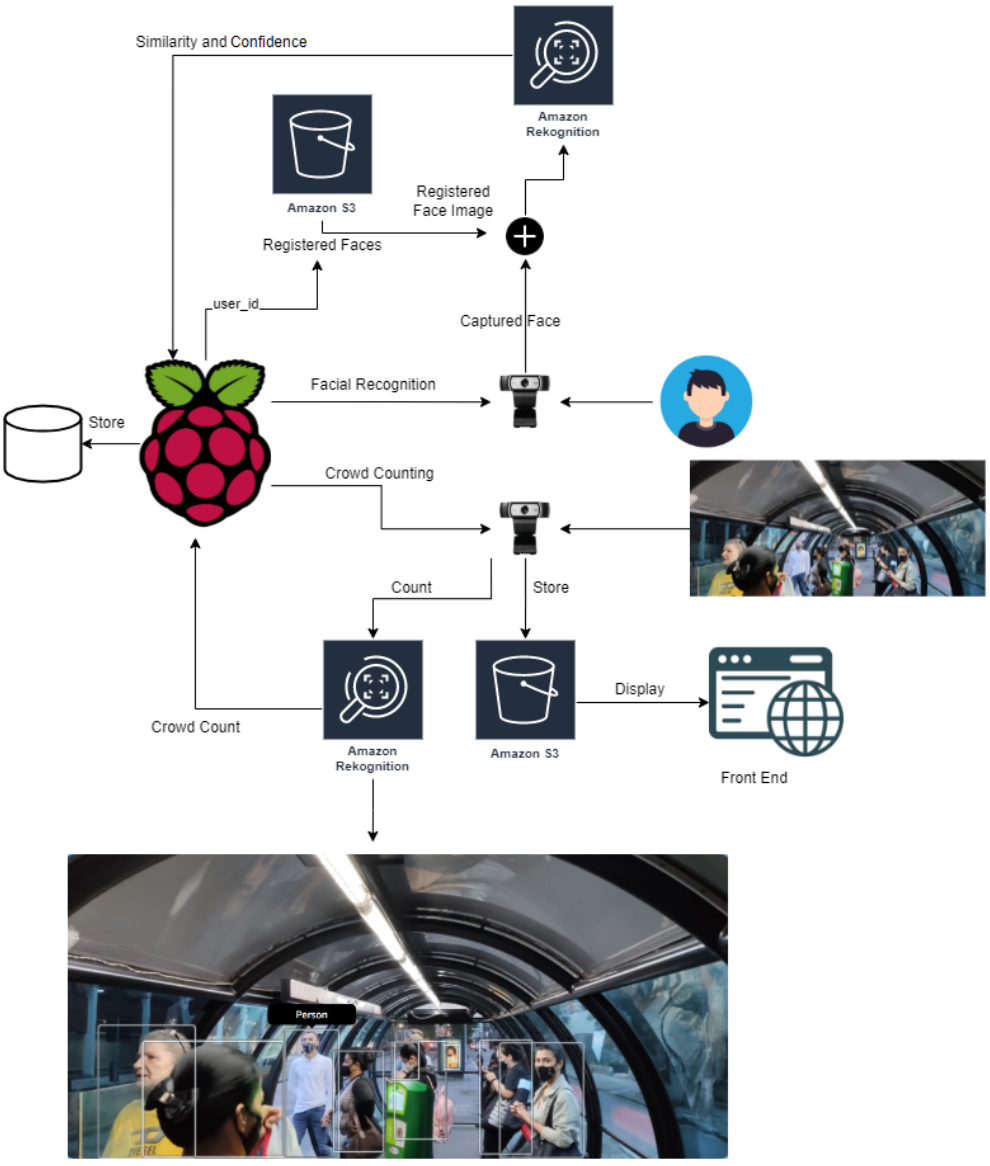

Figure 4: An overview of the communication between the Raspberry and Cloud Systems

In order to allow the payment functionality to work when the turnstile has no connection to the Internet, a database replication system was implemented. Our setup uses a PostgreSQL database server and we have it locally cached on the Raspberry Pi using SQLite [11]. The only special configuration made was to increase the amount of memory SQLite can use for caching. Furthermore, it is in WAL (Write-Ahead Logging) mode, so that the database is never blocked, allowing the turnstile to continue operating normally while the database is updated.

Our local database users table has four columns: the user id, the NFC card

id, a flag that indicates if the user is blocked and the amount of his balance. We have another table that stores the history of updates and a table that stores payments made if there is no internet to send them to the database server. If our local database was never updated, we download the entire table from the server. However, if it already has some data, we download only the lines that underwent some update.

3.4 Web Application

This sections details the development and functionalities of the Web App's front and back end components. All diagrams used to design the software can be found on this link.

3.4.1 Front End

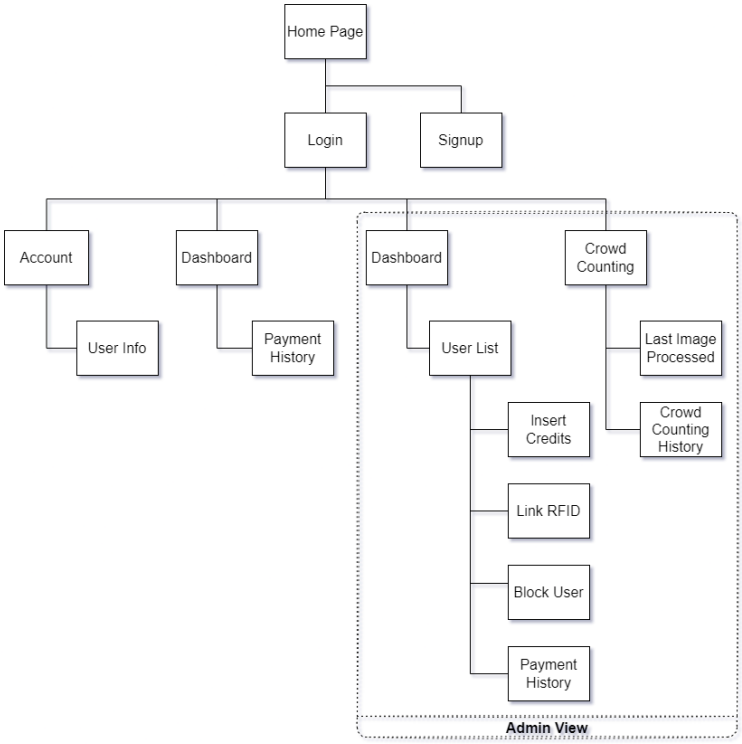

Figure 5: A sitemap of the Web Application

The Web App interface distinguishes between two different classes of users: Admins and Regular Users. Depicted on Figure 5 is a sitemap of the application. Admins have access to a dashboard (Figure 6) that displays the users registered on the system; there, they can check a user's details, see their payment history, add credits to their balance, assign an RFID to their account and also block them. A real money payment transaction system was not implemented

since doing so was out the scope of this project. We assume that a company willing to use our system is either going to provide such a service themselves or seek a third party to do so. Therefore, the current credits system was only implemented for simulation purposes.

RFIDs assigned to users are to be the same ones stored in the magnetic cards they'll use. Whenever a passenger approximates his card or smartphone to the turnstile's NFC module, the RFID assigned to the user will be utilized in the comparison. When a user is blocked, the turnstile will reject accesses coming from the RFID associated to them.

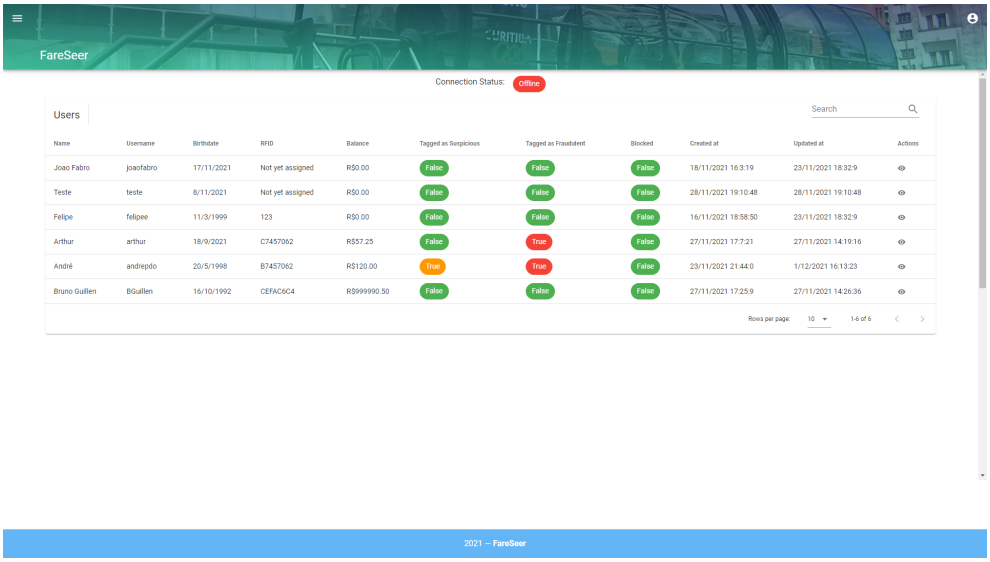

Figure 6: The users list displayed to system administrators

Users who fail the facial recognition process are tagged accordingly. Whenever a passenger is tagged for suspicious or fraudulent activity, the picture taken at the moment the payment occurred is stored as evidence. Admins can then see these pictures when they open up a user's payment history and use this to inform their decision when blocking a user. System administrators also have access to the Crowd Counting page (Figure 7), where a history of the pictures taken from the crowd counting camera along with the estimated amount of people in the image are shown.

Regular users have access to a dashboard that displays their payment history along with a graph of the total amount of credits spent by month. All users have access to the Login, Account and Sign Up pages. During the sign-up process, users are required to upload a profile picture that will be used for facial recognition purposes. On the Account page, users can see their card's status and the information they entered when registering on the system. There, they also have the option of editing their profile picture.

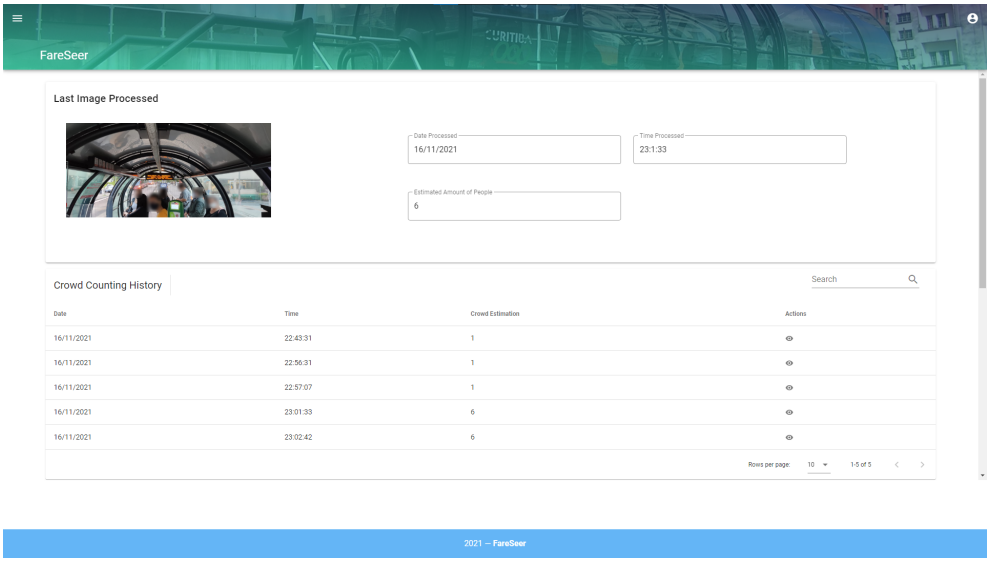

Figure 7: The Crowd Counting page

3.4.2 Back End

The Back End is responsible for managing the request from the Web Page, Mobile and Firmware. It is hosted with AWS on a Ubuntu Server 18.04. It runs a *Node.js* server on port 3333 and a reverse proxy is used on the port 80, so that the connection received in this port is redirected to the server. Also, the domain is configured using CertBot [12], which enables the use of an SSL Certificate and HTTPS protocol on the routing.

3.5 Mobile Application

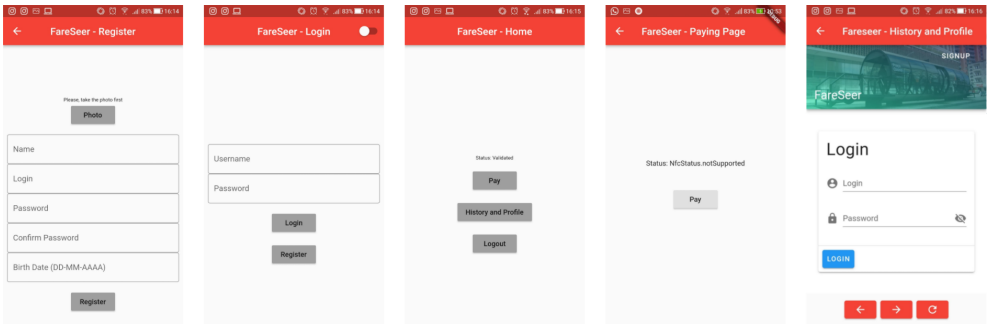

Figure 8: Mobile Application Pages

The Mobile Application's main purpose is to provide NFC payment functionality to users. By using Android's Host-based card emulation technology, the applica-

tion can emulate a magnetic card and communicate with an NFC reader [13]. As depicted on Figure 8, the App's UI has five different pages: Login, Sign Up, Main Menu, NFC Payment and also a Webview that displays the Web App's content. By using the app, users can register themselves on the platform, login, enable NFC to pay by approximation and access contents from the Web App.

3.6 Budget

Figure 9 lists all materials used in the project along with their costs. We were required to spend R$150 per member, totaling R$900. In developing the project, a total of R$1030.34 were spent (14% more than the required amount).

Item	Qnt.	Unit Cost	Total Cost
Raspberry Pi 3B+	1	R$ 414,96	R$ 414,96
Webcam Full HD	2	R$ 72,35	R$ 144,70
NFC reader module PN532	1	R$ 48,63	R$ 48,63
Wood, PVC pipes and other materials for mechanical structure	1	R$ 250,00	R$ 250,00
Servomotor MG995	1	R$ 51,90	R$ 51,90
Infrared sensor LM393	1	R$ 12,00	R$ 12,00
LCD Display 16x2	1	R$ 27,99	R$ 27,99
PVC Box 20x16x8 cm	1	R$ 27,26	R$ 27,26
Power supply - 5v 2A	1	R$ 22,90	R$ 22,90
Electronic components such as leds, buzzer, wires, etc.	1	R$ 30,00	R$ 30,00
Total	-	-	R$ 1.030,34

Figure 9: Materials used and their costs

3.7 Schedule

The project was developed over the course of seven weeks. Table 5 below presents an overview of the total time spent on each deliverable. The complete schedule can be found on this link.

Table 5: Overview of the Scheduled Hours

Deliverable	**Estimated Hours**	**Hours Worked**
Blog and Project Plan	61	60.5
Mechanical Project	42	44
Electronic Project	49	36
Software Project	317	256.5
Hardware and Software Integration	27	45.5
Final Tests and Documentation	127	79
Total	**623**	**521.5**

4 Results and Conclusions

Overall, despite time constraints and difficulties that arose from the COVID-19 pandemic, our team was able to successfully accomplish what was initially proposed. The system we've developed not only performs well, but could, in theory, be applied to a city like Curitiba.

Both the crowd counting and facial recognition systems yielded satisfactory results. Faces can be reliably recognized even if the person is using different clothing items such as face masks, glasses and hats (Figure 10), and two different people are readily recognized as not being the same (Figure 11).

Figure 10: Comparing pictures of the same person wearing different clothing items. Confidence: 99.99813, Similarity: 89.62433

Figure 11: Comparing pictures of two different people.
Confidence: 99.99728, Similarity: 0.09030

For testing purposes, pictures of passengers inside bus stations were taken. Ideally, this would be done by security cameras positioned on the ceiling, yet, in our case, they were captured from a standing position. The estimated number of people was, at all times, very close to the actual amount (Figures 12 and 13).

Figure 12: Crowd counting in a bus station.
API Count Response: 12, Expected: 13

Figure 13: Crowd counting in a bus station.
API Count Response: 9, Expected: 9

Whenever the waiting people move around and from behind others the number of detections may vary. The estimation could be further improved by averaging out the number of people detected on multiple samples.

According to statistics provided by URBS [14], for the year of 2020, the number of active transport cards is 2,035,711, and the average number of passengers transported per day is 710,589. For the year before the pandemic (2019) the the average number of passengers transported per day was 1,331,610. With regards to the database replication system, our benchmark tests demonstrate that it can update 10 million rows in a table with 10 million users in around 3.3 minutes. Considering that our local cache can undergo several small updates a day (every minute, for example), performing the operation only on rows that have had some modification, and with a number of affected lines per day of around 1.3 million (according to the URBS statistics in 2019), we show that even for a real-world situation, with a high volume of data, our system could be applicable.

Moreover, in terms of economical viability, Amazon charges $0.001 per image processed by their *Rekognition* API [15]. As of 2020 the number of tube stations in Curitiba is 333 [14]. If the crowd counting system were to take one picture a minute for 24 hours in all stations, a total of 479,520 pictures would be taken in a day, totaling $479.52 in fees. Given that the daily average amount of paying passengers in 2020 was 340,199 and the bus ticket price was R$4.50, URBS earned around R$1,530,895.5/day in tickets on that year. Therefore, if we convert the daily crowd counting processing cost from USD to BRL (1 USD = 5.68 BRL), this value would represent approximately 0.18% of the total URBS earns in tickets in a day. Similarly, the costs spent on facial recognition processing could also be afforded, since every picture requires R$0.00568 to be processed; a value that could be covered by increasing the ticket price by less than R$0.01.

References

[1] Folha de S. Paulo. Reconhecimento facial bloqueia 331 mil bilhetes Únicos em sp, 2019. https://agora.folha.uol.com.br/amp/sao-paulo/2019/06/reconhecimento-facial-bloqueia-331-mil-bilhetes-unicos-em-sp.shtml.

[2] Gazeta do Povo. Ônibus de curitiba vão ter reconhecimento facial e pagamento com celular e cartão, 2020. https://www.gazetadopovo.com.br/curitiba/onibus-de-curitiba-vao-ter-reconhecimento-facial-e-pagamento-com-celular-e-cartao/.

[3] Amazon. What is aws?, 2021. https://aws.amazon.com/what-is-aws/?nc1=f_ccl.

[4] Vuejs. What is vue.js?, 2021. https://vuejs.org/v2/guide/index.html.

[5] Vuetify. What is vuetify.js?, 2021. https://vuetifyjs.com/en/introduction/why-vuetify/#getting-started.

[6] Flutter. Flutter architectural overview, 2021. https://docs.flutter.dev/resources/architectural-overviewl.

[7] Dart. Dart overview, 2021. https://dart.dev/overview.

[8] Anson. Tripod turnstile user manual, 2021. https://pfccontrols.com/files/Tripod-Turnstile-Manual-EN.pdf.

[9] Carlo Tomasi. Histograms of oriented gradients, 2021. https://courses.cs.duke.edu/fall15/compsci527/notes/hog.pdfl.

[10] OpenCV. Cascade classifier, 2021. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.

[11] SQLite. About sqlite, 2021. https://sqlite.org/about.html.

[12] Certbot. Whats certbot?, 2021. https://certbot.eff.org/pages/about.

[13] Android. Host-based card emulation overview, 2021. https://developer.android.com/guide/topics/connectivity/nfc/hce.

[14] URBS. Urbs em números, 2020. https://www.urbs.curitiba.pr.gov.br/institucional/urbs-em-numeros.

[15] Amazon. Preço do amazon rekognition, 2021. https://aws.amazon.com/pt/rekognition/pricing/.