Child Tracker 5000

## Created by:

## Miguel Millan, Justin Isrow, Colin Cambell, Fabian LeFevre, Levi Orlando

# Project vision

## Background

In this day and age, parents have become increasingly concerned over the wellbeing of their children. Yet, between home and school they have no way of verifying where their kids are. One solution to this is monitor kids as they enter and leave the bus. However, while there are many in bus camera systems available for monitoring the passenger activities in the bus, due to the large cost of associated cellular charges, such solutions are cost prohibitive, and they lack the ability to verify specific children. As an alternative, bus systems may implement multi-camera capture system to monitor entering and exiting events for child identification. Utilizing these camera systems to train a facial recognition machine learning algorithm, it is possible to develop an intelligent child tracking system capable of effectively detecting and identifying a parent’s child.

## Socio-economical Impact, Business Objectives, and Gap Analysis

## Security and Ethical concerns

For the project, there would need to be security options for the web application/cloud system, in which there would be registered devices, as well as, admins and parents that can login to the web application. There needed to be ways to authenticate, and the Django Framework had that built in for Administrators and Users. This is done in our web application by utilizing token authentication with JWTs. This also made it so only registered Devices can communicate with the web application.

Other security concerns deal with what information is available to be seen and who can see it. Of course, Admins can see a lot (they can also add, modify or delete almost everything), but wouldn’t want them to see, or rather, use personal information of Parents, like their passwords. For Parents and Devices, they would need a more limited access to the web application. Parent Users can’t be able to see other Users in the system, another Parent’s children, and events that don’t involve their children. Devices are the most limited and only need to send configured information to the web application.

Ethics-wise, considerations had to be made about the options for how to notify Parent Users of events related to their children that are tracked through the system. This is done through web notifications and email notifications. This way notifications can be reliable and easy to access while accessing or not accessing the web application.

Other ethical concerns were on how much information is necessary for the web application to track and maintain. What information would the system need at minimum to track and catalog events of a student? What is necessary information for a Parent User? What information about an event should a Parent User be able to see? How and where should that information be displayed? What is information that would be good to have for registered Devices, like where is it installed and who registered it? Addressing questions like these is very important in guiding how the application and system would be designed and implemented.

## Glossary of Key Terms

**Administrators (Admins)** - Administrative users of the web application that have superuser access and can view, add, modify and delete elements of the server’s database

**Web application** - The website that registered Users can log into and utilize the web features of the project.

**Device(s)** - Physical hardware system that is installed onto a Bus. Utilizes a Raspberry Pi and vision bonnet to detect faces, capture events, and send captured information to the server.

**Parent(s)** - User that can log into the web application and view their registered children’s information and captured events.

**Student(s)** - Indirect actor of the project. Have Parents that are registered to the web application. Are to be captured by a registered Device to have their actions on a Bus cataloged by the Server.

**System** - Comprised of both the Server and registered Devices. Devices will communicate information to the Server, which will then have that processed information cataloged and available, in specific ways, to Users of the web application

**Server** - The Linux server that hosts the web application and the face identification program.

**User(s)** - Members of the web application that can authenticate with the Server and/or login and then view the web application.

**Vision bonnet** - A visual processing unit (VPU) designed by google to run lightweight machine learning inference models.

**Raspberry Pi** - A cheap single-board computer designed for learning programming and electronics.

**Inference Model** – A machine learning model, usually a weights file, that can be used to classify images or other trained events.

# Project Execution and Planning

## Term Information

Our team will build an intelligent child identification system. This system will consist of two subsystems: a local system composed of two cameras and one or more microcomputers that monitors students entering and exiting the bus, and a cloud server which will act as the parent’s portal for monitoring their child’s activity. When a child’s face is captured entering or exiting the bus, a snapshot of their face and status will be sent from the local system to the cloud server for child identification. Once the child is identified using the facial recognition machine learning algorithm, their status in the system will be changed, and their parent will be alerted.

## Tools and Technology

* Github
* Discord
* PyCharm
* Notepad++
* Astah
* Draw.io
* Visual Studio Code
* Django Rest Framework
* Node JS
* React JS

## Project Plan

Last Updated: 1/13/19

Sprint 1: 1/7/2019 - 1/16/2019

* Project Plan
* Requirements
* Tool Chain Setup
* Project Scope Capture

Sprint 2: 1/16/2019 - 1/23/2019

* Basic Diagrams
* User Stories / Detailed Use Cases
* Cloud hosted server setup and configuration
* Camera system research / design

Sprint 3: 1/23/2019 - 2/11/2019

* Web UI skeleton and basic functionality
* Camera system deployment & initial data collection
* Device Registration implemented in backend

Sprint 4: 2/11/2019- 2/25/2019

* Implement general facial recognition on the local system
* Start looking into facial identification
* Start creating admin view for the frontend

Sprint 5: 2/25/2019 - 3/11/2019

* Implement Local system to Cloud server communication
* Finish admin view of the frontend
* Create scripts to automate Microsoft Facial Recognition API calls
* Start working on the parent view of the frontend

Sprint 6: 3/11/2019 - 3/25/2019

* Complete the parent view of the frontend
* UI improvements if necessary/desired
* Start integrating Microsoft Facial Recognition API Calls to identify students.

Sprint 7: 3/25/2019 - 4/9/2019

* Testing and debugging
* Dry Run

Sprint 8: 4/9/2019 - 4/24/2019

* Finish fixing any existing bug
* Complete documentation
* Final Competition

## Best Standards and Practices

To best manage this project, we decided to use the Scrum method and break up the work into two-week long Sprints so that it could be planned in more manageable fashion.

The team member roles and responsibilities are as follows:

* Miguel Millan – Project Lead / Documentation / Local system / Web Application Developer
* Justin Isrow – Backend / Server Administrator / Web Application Developer
* Colin Campbell – Face Identification / Documentation
* Fabian LeFevre – Web Application developer / Documentation
* Levi Orlando – Web Application developer / Documentation

We decided for the whole team to meet at least once a week on Friday to discuss progress for each team member and work through team goals. In addition, we planned on meeting immediately after end of sprint presentations to ensure each member knows what their individual responsibility is for the next sprint.

Our github repository is hosted at <https://github.com/mjmillan/PassengerAndDriverSafety/>.

Our team drive in google drive is located at <https://drive.google.com/drive/folders/0AHXDDMPRTAS-Uk9PVA> (you need permission to access it).

For quick and easy communication, we collaborated over discord.

## Risk Management

# System Requirement Analysis

## Functional Requirements

#### Cloud Application Requirements:

1. The cloud application shall allow one or more local bus systems to be registered for monitoring purposes.
2. The cloud application shall be capable of training the facial detection algorithm using the data in the database.
3. The cloud application shall store a child’s information.
4. The cloud application shall store a parent’s information.
5. The cloud application shall store admin information.
6. The cloud application shall allow admins to register parents children.
7. The cloud application shall allow parents to deregister their child.
8. The cloud application shall allow parents to view the current status of their child (on or off the bus).
9. The cloud application shall display the most recent picture of a parent’s child.
10. The cloud application shall update a child’s status when they enter a bus.
11. The cloud application shall update a child’s status when they exit a bus.
12. The cloud application shall notify a parent when the current status of their child changes.
13. The cloud application shall allow parents to verify whether the local system correctly detected their child.
14. The cloud application shall allow admins to register new users (parents).
15. The cloud application shall allow admins to register new admins.
16. The cloud application shall allow admins to deactivate accounts.
17. The cloud application shall allow admins to view all children on a bus.
18. The cloud application shall allow admins to manually change the status of a child.
19. The cloud application shall use facial recognition to identify a child who enters the bus.
20. The cloud application shall use facial recognition to identify a child who exits the bus.

#### Local Bus System Requirements:

1. The local bus system application shall implement an algorithm to detect a child entering the bus.
2. The local bus system application shall implement an algorithm to detect a child leaving the bus.
3. The local bus system application shall take a picture when a child is detected as having entered the bus.
4. The local bus system application shall take a picture when a child is detected as having left the bus.
5. The local bus system application shall produce an alert when a child is detected as having entered the bus.
6. The local bus system application shall produce an alert when a child is detected as having left the bus.
7. The local bus system application shall upload camera data (pictures) to the cloud for online monitoring when a child is detected as having entered the bus.
8. The local bus system application shall upload camera data (pictures) to the cloud for online monitoring when a child is detected as having left the bus.

## Non-functional Requirements

1. Failure to recognize an event should occur less than 1% of the time when an event occurs.
2. The local system should be available 99% of the time.
3. The local system should be able to re-establish a connection to the cloud system after a power failure occurs.
4. The cameras of the local system should be easily replaceable. (maintainability)
5. The cloud system should be available 99% of the time.
6. The cloud system should keep backups to ensure reliable restoration in the event of a system failure.
7. The cloud system should be secured from data breaches.
8. The cloud system should be scalable to allow for increased traffic as more local system are added.

## On-Screen Appearance of landing and other pages requirements

## Wireframe designs

# Functional Requirements Specification

## Stakeholders

## Actors and Goals

## User stories, scenarios and Use Cases

#### User Stories

1. As a camera, I can detect when someone has entered the bus in order to keep track of activity on the bus.
2. As a camera, I can take a picture of that person’s face when they enter or exit the bus in order to determine if this person is a child within the web application’s database.
3. As a camera, I can detect when someone has exited the bus in order to keep track of activity on the bus.
4. As a camera, I can determine if the person in a taken picture is a child inside the web application’s database in order to determine what to do with the newly gained information (scrap or notify the web application).
5. As a parent, I can register my child to the web application for database entry so that they can be identified by the camera and have their activities documented by the web application.
6. As a parent, I can update my child’s information through the web application and have that update the app’s database entries to ensure all information regarding my child is up to date for the web app and camera device to use.
7. As a parent, I can receive email notifications about my child’s activities on the bus in order to be sure of my child’s safety.
8. As a parent, I can sign into the web application in order to utilize the web app’s features and information regarding my child.
9. As a parent, I can view my child’s activities (got on, got off) while on the bus through the web application in order to confirm and ensure my child’s safety.
10. As a parent, I can use the web application to verify if the device used to identify my child was correct in order to notify the web application and other systems of success/failure.
11. As a parent, any email notification I receive will include a link to the appropriate web application page in order to access and utilize the web application’s relevant web pages and information.
12. As a parent, I can unlist my child from the web application database so that the web app and camera no longer track my child but still be able to hold their information while not having them viewable by anyone else other than admins.
13. As an admin, I can add a parent user to the web application database so that the parent can login and utilize the web application.
14. As an admin, I can enter parent information to the web application database so that the web application can utilize this information appropriately.
15. As an admin, I can query the database for parent user entries by using their identifier information in order to easily search through and parse the web application’s database entries.
16. As an admin, I can deactivate a parent user’s account so that when a parent no longer wants to be a part of the system the system can ignore their entry and all related information while not removing any of it from the web app’s database.
17. As an admin, I can view all children in the web application’s database in order to see all that the database currently has in it’s entries in regards to the registered children.
18. As an admin, I can view all children registered as “Gotten On” in the web application’s database in order to make it easier to search and parse the database.
19. As an admin, I can query the database for children by using their identifier information in order to make it easier to search and parse the database.
20. As an administrator, I can monitor events data from the cloud web application.
21. As an administrator, I can log into the web application.
22. As an administrator, I can be alerted when an alarm is triggered.
23. As an administrator, I can register bus camera systems.
24. As a web application, I can store recorded events in the cloud database.
25. As web application, I can automatically classify events to update and refine the system.
26. As a web application, I can update registered camera systems learning algorithms.
27. As a web application, I can store user information.

#### Use Cases

§ **Name**: Register camera

§ **Pre/Entry-condition**

* Camera must be on and broadcasting.

§ **Trigger**

* The administrator clicks “Register Camera” button.

§ **Post/Exit-condition**

* A pop-up message appears saying registration successful. Camera information is added to the list of registered camera in the system.

§ **Main flow of events – identify all data elements**

* The administrator clicks the “Register Camera” button.
* A window displays the list of broadcast IDs of cameras.
* The administrator selects the camera and clicks the “Add” button.

§ **Exceptions and alternate actions**

§ **Name**: A Person is detected getting on the bus

§ **Pre/Entry-condition**

* The camera must be on and hooked to the system.

§ **Trigger**

* A person gets on the bus

§ **Post/Exit-condition**

* The system adds 1 person to the “bus count”

§ **Main flow of events – identify all data elements**

1. A person gets on the bus.
2. The camera detects the person getting on the bus.
3. The camera sends a signal to the system to add 1 to the bus count.

§ **Exceptions and alternate actions**

§ **Name**: A Face is Recognized

§ **Pre/Entry-condition**

* A person must be registered to the system with a picture of their face.

§ **Trigger**

* A registered person gets on the bus

§ **Post/Exit-condition**

* The system confirms that the person is on the bus

§ **Main flow of events – identify all data elements**

1. A registered person gets on the bus.
2. The bus camera detects the persons face and recognizes it from the picture they registered to the system with.
3. The system displays that the person has safely arrived on the bus.

§ **Exceptions and alternate actions**

* If a person has not previously registered to the system with a picture, the camera will not recognize their face.

§ **Name**: Register Child to webapp

§ **Pre/Entry-condition**

* User must have access to webapp

§ **Trigger**

* User clicks the “register child” button

§ **Post/Exit-condition**

* Child is registered to the system

§ **Main flow of events – identify all data elements**

1. user accesses the system.
2. The user clicks the “register child” button
3. The user fills out all detail asked in the form including a minimum of 1 picture of the childs face.
4. The child is registered to the system

§ **Exceptions and alternate actions**

§ **Name**: Parent is notified

§ **Pre/Entry-condition**

* Parent has registered child to the system

§ **Trigger**

* Child has not been detected getting on the bus.

§ **Post/Exit-condition**

* Parent is notified that their child has not arrived on the bus.

§ **Main flow of events – identify all data elements**

1. The bus arrives at pickup destination
2. The bus departs from the pickup destination
3. The system detects that a child has not arrived on the bus and has not been detected.
4. The system notifies the parent via email that their child has not arrived on the bus

§ **Exceptions and alternate actions**

* If a child is not registered to the system, the parent will not be notified.

§ **Name**: Verify identification of child

§ **Pre/Entry-condition**

* Parent has registered child to the system

§ **Trigger**

* The user selects if the system has correctly identified their child or not.

§ **Post/Exit-condition**

* A prompt that reads “thank you for your feedback” will appear.

§ **Main flow of events – identify all data elements**

1. The system displays the picture taken of identified child to corresponding user based off of registered childs picture.
2. The system asks “has your child been identified correctly? Yes or No”
3. The parent may click either yes or no button.
4. The parent clicks yes.
5. The system updates registered childs pictures with the picture taken when the child was identified getting on the bus.

§ **Exceptions and alternate actions**

* If the parent clicks “No”, then the camera will scrap the picture and prompt the user to add new photo(s) of their child to the system.

#### Scenarios

1. **A child enters the bus**

As a child gets on the bus, the entrance facing camera detects and keeps track of their face. After the child leaves the camera view, the last face detected is cropped and sent from the local system to the cloud system for identification. The cloud system identifies the face of the child and updates the status of that child. The parent of that child is then sent a notification that their child has entered that bus.

1. **A child gets off the bus**

As a child gets off the bus, the exit facing camera detects and keeps track of their face. After the child leaves the camera view, the last face detected is cropped and sent from the local system to the cloud system for identification. The cloud system identifies the face of the child and updates the status of that child. The parent of that child is then sent a notification that their child has exited that bus.

1. **An admin registers a parent’s child**

An admin signs into their account on the cloud server and navigates to the child registration page. To register a parent’s child, they provide the parent’s name, child’s name and upload a picture of the child to the cloud server. The child’s information is stored in the server database, and the facial recognition algorithm is then retrained.

1. **A parent checks on their child**

A parent signs into their account to see a record of if their child got on and off the bus along with what times, along with pictures. If the pictures are wrong, the parent can report that the reporting was false.

1. **An admin deregisters a parent’s child**

A parent alerts an admin they would like their child deregistered from the system. The admin logs into their admin account and navigates to the students page. The admin then edits the child’s monitoring status and sets it to inactive (so they will no longer be tracked/reported).

1. **An admin adds a parent to the system**

An admin logs into their admin account and navigates to the new parent registration page. They then enter the parent’s name, email, phone number, and click a button to register them. The registered parent’s information is added to the database.

1. **An admin removes a parent from the system**

An admin logs into their admin account and navigates to the parent database screen. They will then search for the parent in the database through a search field, confirm that the parent is in the database, select the database entry of that parent and set their status to ‘inactive’ in the database.

1. **An admin can view all parents, students, schools, buses, devices, events, and admins**

An admin logs into their admin account and are taken to the admin page where they can view, create, update, and delete admins. They can then navigate to the students page to view, create, update, and delete students. Or, they can navigate to the parents page to view, create, update, and/or delete parents. Or, they can navigate to the device page to view, create, update, and delete devices. Or, they can navigate to the events page to view, create, update, and delete events. Or, they can navigate to the buses page to view, create, update, and delete buses. Or, they can navigate to the schools page to view, create, update, and/or delete schools.

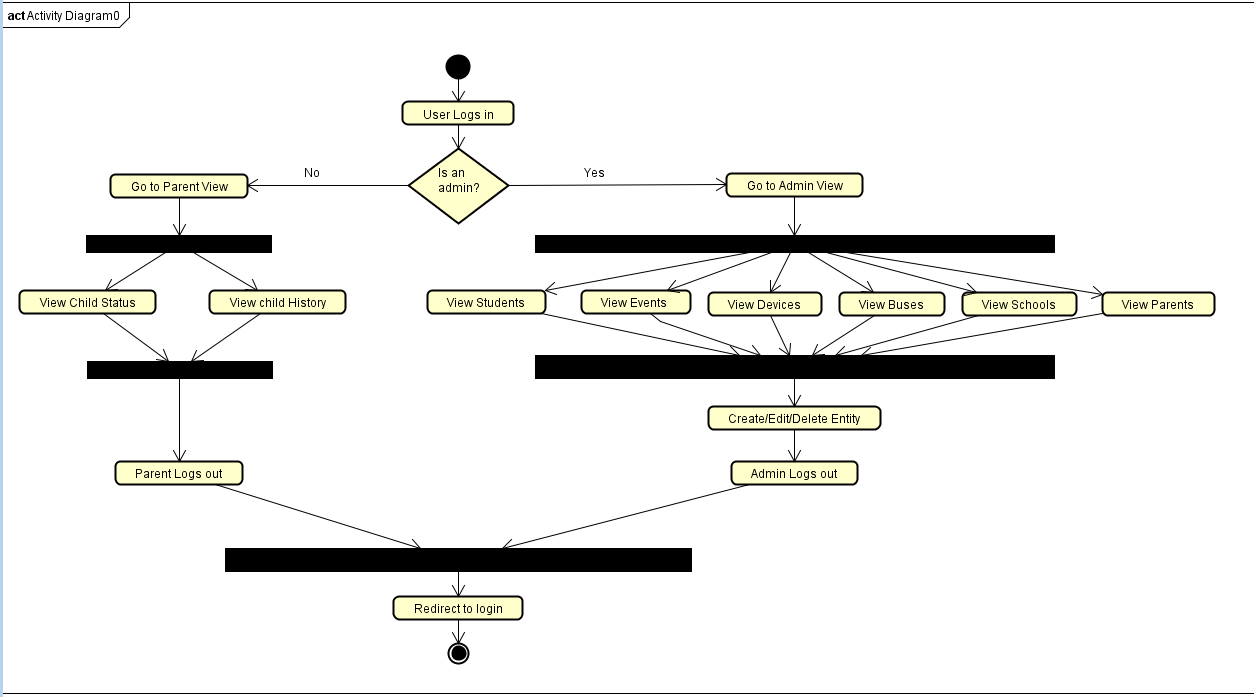
1. **A parent is notified of a child entering/exiting the bus via email**

A parent is sent an email, which contains a picture and a notice that the child is getting on or off the bus. The parent can go to the website through the email, taking them to the check-in page. They can then report any incorrect reporting.

1. **An admin registers a device with the cloud system**

An admin logs into their admin account and navigates to the devices page. They then enter the device name, entering/exiting status, bus number, and who it was registered by. The registered device information is then added to the database.

## System Sequence / Activity Diagrams



# User Interface Specifications

## Preliminary Design

The design for the UI will be that it has different types of views for the admin and the parent users. The views are entirely separate. The admin will be able to navigate through all of the tables using a responsive interface. Different tabs to the side will take the user to different database entities where changes can be made. The parent view will be a landing page with cards featuring the details of their children. Additionally, there will be some recent events from each child along with their status. The parent will also be able to open up the full list for their child.

## User Effort Estimation

The UI has two different views, and both are fairly complex. Doing all the code for each one would probably take over a day’s worth of person hours. The admin view will have some code reusability, but there is much to do for the admin view overall. The parent view has less features but needs to be presented elegantly. Each one has a multitude of different things to work on, not even taking into account the work that will go into building the front end that doesn’t involve designing and coding the UI. The estimate on it would be around 20 hours per view, with plenty a couple hours to implement each major functionality.

# Static Design

## Class Model

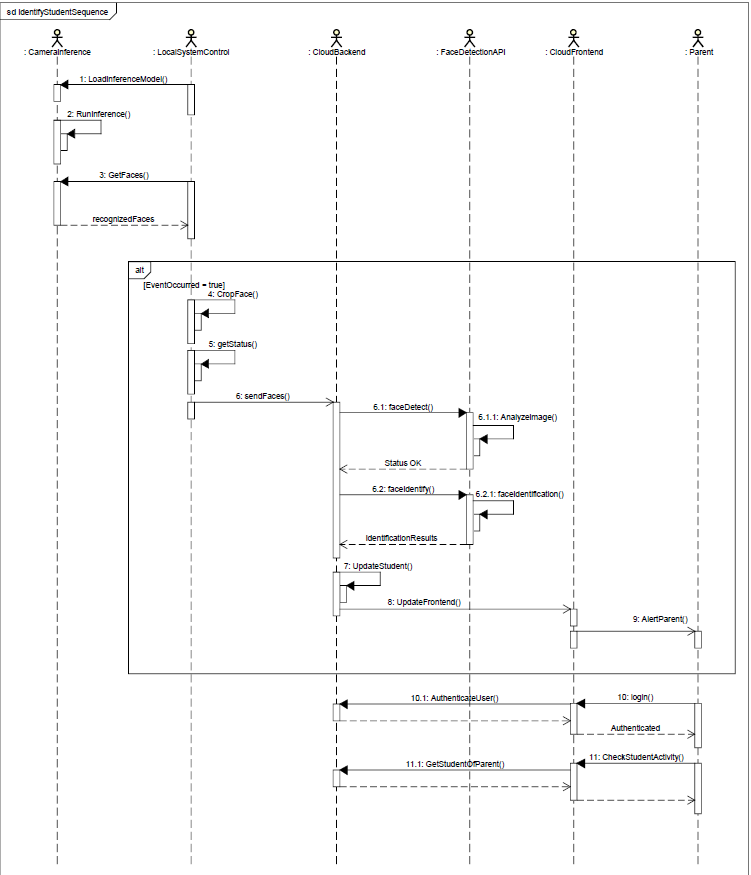
## System Operation Contracts

## Mathematical Model

## Entity Relation

# Dynamic Design

## Sequence Diagrams



## Interface Specifications

## State Diagrams

# System Architecture and System Design

## Subsystems / Component / Design Pattern Identification

There are two subsystems in the Child Tracker 5000. One is the Rest Django Project, the other are the scripts running on the camera devices. The design pattern is that the front end acts as a view for the system backend with a view for admins and a view for parents. There are APIs on the back end which allow the back end to communicate with the camera subsystems, and so the back end has the information from the camera subsystems.

## Mapping Subsystems to Hardware (Deployment Diagram)

This system has essentially two subsystems: a server node, which has the Rest Django Project and its related database deployed to it, and the camera systems, which each one has the facedetection monitor script running on it.

## Persistent Data Storage

All database data is persistent, stored in files on the server, but passwords are hashed. Microsoft Face API data is also persistent between sessions and failures.

## Network Protocol

The server is accessed using HTTP requests, and HTTP is also used to access resources like the Face API. The camera uses HTTP to access the server. The front end and the back end communicate through HTTP.

## Global Control Flow

Control flow is done mostly with restful APIs. So, APIs are given control when they are called through HTTP requests. The cameras are always running and might use the server APIs, allowing the server side back end stuff to run. The front end is given control when the user requests a page. The front end sometimes transfers information to the back end to run stuff there.

## Hardware Requirement

* Simple server for running the web app
* AIY Vision Kit
* Raspberry Pi 3

# Algorithms and Data Structures

## Algorithms

The most algorithmically intense area is the scripts running on the cameras. They use facial recognition software to recognize faces and have an algorithm to track movement. Other machine learning algorithms for identifying students entering or leaving the bus are done using Microsoft’s Face API and were not implemented by the team, but are utilized and understood.

Local System Algorithm:

1. The device must first establish a connection with the cloud server.
   1. Obtain JWT using preconfigured credentials
2. Each device uses a preconfigured region to determine what faces in the field of view are to be counted as event triggers.
   1. So, first calculate the center
   2. Then calculate the corners of the region
3. If streaming is enabled, start the local server (used for debugging purposes).
4. Load the inference model and run facial recognition model in a loop
   1. If 4 minutes have passed since the access token (JWT) was obtain, refresh the token
   2. Loop through the detected faces:
      1. If a face is inside the region, then take a picture and crop out each face in the picture
      2. Store the faces and the associated status in a temporary array
   3. If the number of faces in this iteration is less than the number of faces in the previous iteration, then someone has left the region (entered/exited), so send the previous set of faces and statuses to the cloud server for identification.
   4. Set the current temporary arrays as the set of previous faces (up to three iterations)

## Data Structures

There were no data structures defined for this project, although the team makes heavy use of a database to organize data and often composes HTTP requests which store a lot of different information in JSON objects.

# User Interface Design and Implementation

## User Interface Design

## User Interface Implementation

# Testing

## Unit Test Architecture and Strategy/Framework

## Unit Test definition, test data selection

## System Test Specification

## Test Reports per Sprint

# References

<https://www.python.org/>

<https://realpython.com/>

<https://javascript.info>

<https://material.io/>

<http://astah.net/>

<https://notepad-plus-plus.org/>

<https://discordapp.com/>

<https://github.com/>

<https://www.jetbrains.com/pycharm/>

<https://nodejs.org/>

<https://reactjs.org/>

<https://www.django-rest-framework.org/>

<https://www.django-rest-framework.org/api-guide/authentication/>

<https://code.visualstudio.com/>

<http://rogerdudler.github.io/git-guide/>

<https://gist.github.com/hofmannsven/6814451>

<https://www.tensorflow.org/tutorials/estimators/cnn>

<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/development_environment>

<https://blog.algorithmia.com/hardware-for-machine-learning/>

<https://aiyprojects.withgoogle.com/vision>