# GrizzPark

Created By:
Vlad Franco, Jon Mukes, Ryan Dodge, Nick Furlo, Earl Maudrie, Nick Papatheodore

# 1.Project vision

## 1.1.Backgrounds

Nick Furlo: Experience with developing android applications, UI/UX design, java, Raspberry Pi setup and use.

Ryan Dodge: Experience with networking, Java, Linux, Raspberry Pi setup and use, UI design, website design.

Vlad Franco: Experience with mobile app development, machine learning/bot training, data analysis, Raspberry Pi use, server setup/maintenance, Python scripting

Jon Mukes:Expirence with neural nets, machine earning, Web development database management, api building, computer vision, linux, raspberry pi programming, python, javascript, general programming, network/server mantinence

Nick Papatheodore: Experience with website development, Java, Linux, Raspberry Pi maintenance, networking

Earl Maudrie: Experience with system administrator, network experience, setup and maintenance of Raspberry Pi's

## 1.2.Socio-economicImpact, Business Objectives, and Gap Analysis

Socio-economic Impact: The impact for our project will help to lower students' stress levels by providing them with an easy parking experience. The overall lower stress will help in boosting the students education by lowering the amount of time spent driving, which will make them less likely to be late to their classes. The quality of life will increase because the less time spent speeding around a parking lot will both reduce the wear and tear on their vehicles and possibly save lives because people tend to speed down isles when looking for parking. This will also cut down the average time spent driving at school which can help cut down on vehicle emissions and save both the environment and money for the students.

Business Objectives: This project can be used in a variety of different business applications, mostly places that have large number of employees or customers parking. Places like Sporting arenas, malls, and college campuses can use this technology to help reduce the overall traffic in their areas. By reducing the traffic in a business area, it will allow for more traffic to flow through the area and create a better business environment.

Gap Analysis: The potential for this project is up to the owner of the app. The more money spent on the app (servers, upgraded cameras, number of cameras, the hardware) they greater potential. Upgraded cameras and the placement of the cameras are the biggest factors in making this app work to the best of its ability.

## 1.3.Security and ethical concerns

**Security Issues**: The security concerns with this project is we have a camera taking images which can be used to monitor others, but with the correct security, it will get rid of this issue.

**Ethical concerns**: This is a mobile application that is going to be used while the student is driving. Of course, we recommend that users have the app preloaded before they get in the car, but there is a possibility that users will try to use this app while driving.

## 1.4.Glossary of Key Terms

**Raspberry Pi:** is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.

**YoloV3:** is the latest variant of a popular object detection algorithm YOLO – You Only Look Once. The published model recognizes 80 different objects in images and videos, but most importantly it is super fast and nearly as accurate as Single Shot MultiBox

**Android Studio:** is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems.

**APK:** is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware.

**API:** is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software.

**UI**: User Interface

**UX**: User Experience

**Putty**: is a free and open-source terminal emulator, serial console and network file transfer application.

**FileZilla**: is a free software, cross-platform FTP application, consisting of FileZilla Client and FileZilla Server.

# 2.Project Execution and Planning

## 2.1.Team Information

Team member roles:
Android App Development: Nick F. and Vlad.
Android App UI Development: Nick F. and Vlad.
Raspberry Pi Setup and research: Everyone.
Raspberry Pi Maintenance: Vlad, John, Earl, and Ryan.
Website Development: Ryan, Vlad
Bot Development: John and Vlad.
AWS Server Setup and Maintenance: John, Vlad.
Powerpoints: Everyone.
Documentation: Everyone.

## 2.2.Tools and Technology

1. Raspberry Pi 3 and cables(3)
2. Arducam M12 Camera Module, Focus and Angle Enhancement(x3)
3. Amazon Web Server
4. Putty
5. FileZilla
6. OpenCV
7. YOLOv3

## 2.3.Project Plan

Our plan is to develop this project through 6 sprints throughout the semester at intervals of around 2 weeks per sprint. We will set goals for each sprint and attempt to meet those goals every time. If there is a problem with the project, it will be brought up during our bi-weekly meetings with the team.

## 2.4.Best standards and Practices

The best standards and practices for this project is going to be communication. We will have a standard meeting time of 9am every tuesday and thursday throughout the semester, until we are at a point where we feel comfortable with our project. We will utilize apps like Discord, Google Drive, and GitHub to communicate and upload our progress.

# 3.SystemRequirementAnalysis

## 3.1.Functional Requirements

F01: Take pictures of the parking lot(s).
F02: Send the pictures to the web server.
F03: Run detection algorithm on the sent pictures on the server and output results through API.
F04: Pull output from web server into Android mobile app and display.
F05: Inform user of parking lot status

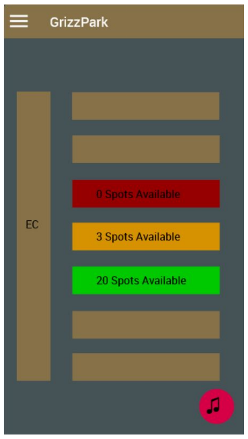
## 3.2.Non-functionalRequirements

NF01: Live feed of camera(s) on site.
NF02: Google auth for admins on site.
NF03: Increase coverage area.
NF03: Text-to-Speech implementation in mobile app.
NF04: Implementation of geolocation into mobile app.

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

Our landing page will require users to log into their school gmail account before landing on the app main page. Once logged in, the user will land in the default parking lot of EC where they will be able to see the available rows as well as change the lot that they are parking in. There is also a function to allow the users to log off when they want to.

### 3.4.Wireframe designs



## 4.Functional Requirements Specification

### 4.1.Stakeholders

Development team, Oakland University Students, Oakland University Administration

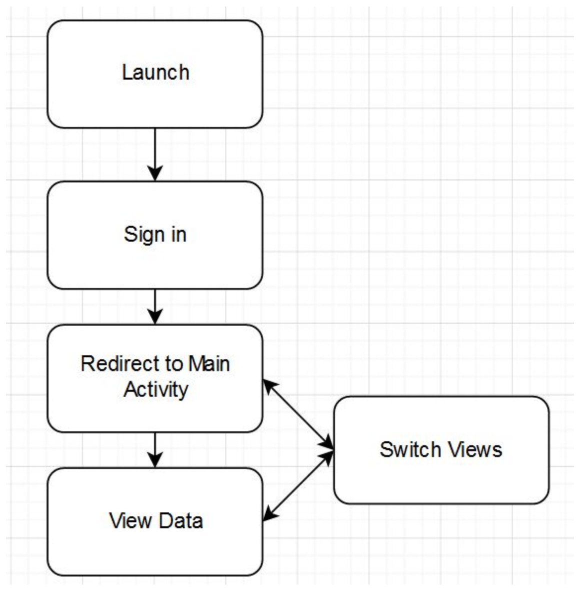
### 4.2.Actors and Goals

End User: Wants to know number of empty spaces in destination parking lot(s).
Admin User:Wants to be able to access website to view live feed.

### 4.3.User stories, scenarios and Use Cases

US01: A user will be able to see the number of available spots through the mobile app.
US02: An admin will be able to view the live feed to ensure service uptime.
US03: A user will be able to hear lot information from the app upon approach

## 4.4.System Sequence / Activity Diagrams



# 5.User Interface Specifications

## 5.1.Preliminary Design

The preliminary design of our project is centered around our android mobile application. It is designed to announce to the user how many parking spots are available when he/she gets closer to our geolocation boundaries. Once the user is close enough, the app says there are X available spots in Row X.

Our app is designed to eliminate the stress of finding a parking spot if you're late to class, or whatever the case may be.

## 5.2.User Effort Estimation

Ryan Dodge: 120 hours

Vlad Franco: 130 hours

Nick Papatheodore: 50 hours

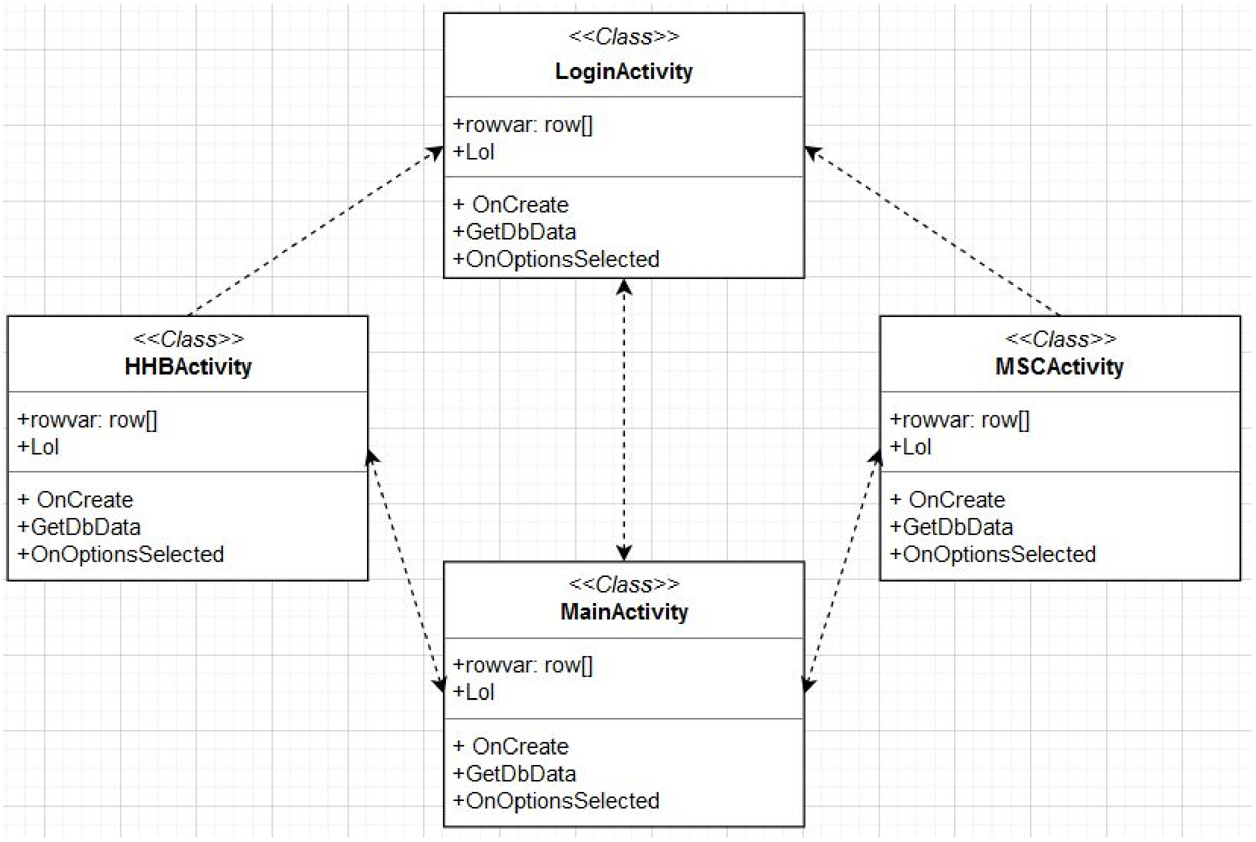
Nick Furlo: 85 Hours

Jonathan Mukes:125

Earl Maudrie: 90

## 6.Static Design

### 6.1.ClassModel



### 6.2.System Operation Contracts

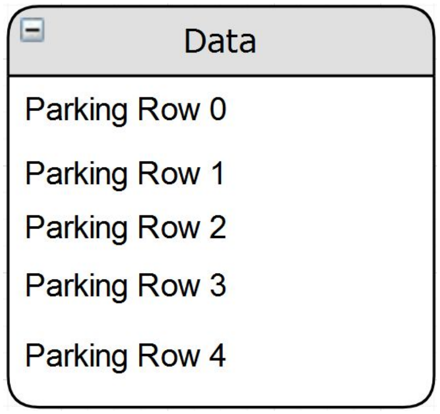
- CT01: Sign In
  - LogIn(UserG, Pass)
  - Upon launch, user is directed to Login activity.
  - System will load Google accounts in phone cache or ask the user to input a new one.
  - Tap on account to load account.
- CT02: View
  - MainActivity
  - Upon sign-in, user is directed to main activity, and can view parking data.

- System waits on geo trigger for text to speech.

## 6.3.MathematicalModel

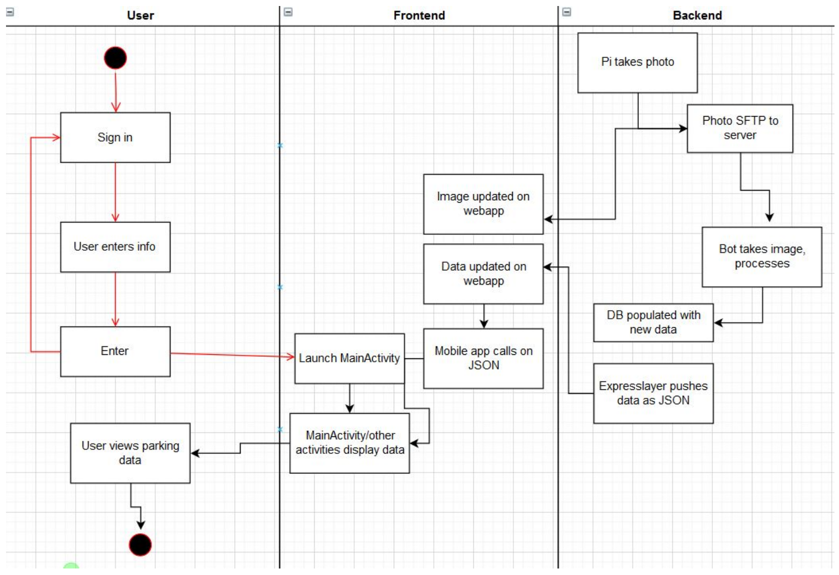
Linear flow, mathematical model unnecessary.

## 6.4.Entity Relation



# 7.Dynamic Design

## 7.1.SequenceDiagrams



### 7.2.Interface Specification

Interface must be extremely simple to allow for easy use, even while driving. Big, blocky cards dominate the view showing available spots.

Hamburger menu holds all the options, allowing change of view and sign out.

Designed to be streamlined and keep the user from having to do too much.

### 7.3.State Diagrams

Due to the nature of our system, a working state is the only state. Our Pi is never in a state of rest, and constantly taking photos and sending them. Our bot, too, constantly processes images and returns data, and our app always shows our data.

# 8.System Architecture and System Design

### 8.1.Subsystems/ Component / Design Pattern Identification

- Raspberry Pi
- Ubuntu server
- Android App

### 8.2.Mapping Subsystems to Hardware (Deployment Diagram)

- Android App deployed through .apk file, compiled through Android Studio.
- Ubuntu server built on an AWS cloud instance.

### 8.3.Persistent Data Storage

Our data is stored in our MySQL database following image recognition, and temporarily stored as JSON on our webpage through ExpressJS

### 8.4.Network Protocol

HTTP Get requests from our mobile app to get JSON data.
SFTP pushes our images to our server from our Pis.

### 8.5.GlobalControlFlow

1. Raspberry Pis run Cronjob, take pictures of lots

2. Pictures pushed to our server through SFTP, using AWS key
3. Pictures stored on server
4. Webapp constantly refreshing pictures as they come in, displaying all 3 lots in real time
5. Bot processes images by cropping and adjusting images
6. Runs detection on each cropped image, returns integer for each row representing status
7. ExpressJS layer pushes integers to webapp as JSON
8. Android app makes HTTP request to JSON page every second
9. Data on android app displayed in lots' respective activities
10. App uses Google Maps api for location data, checks and runs voice function if near lot

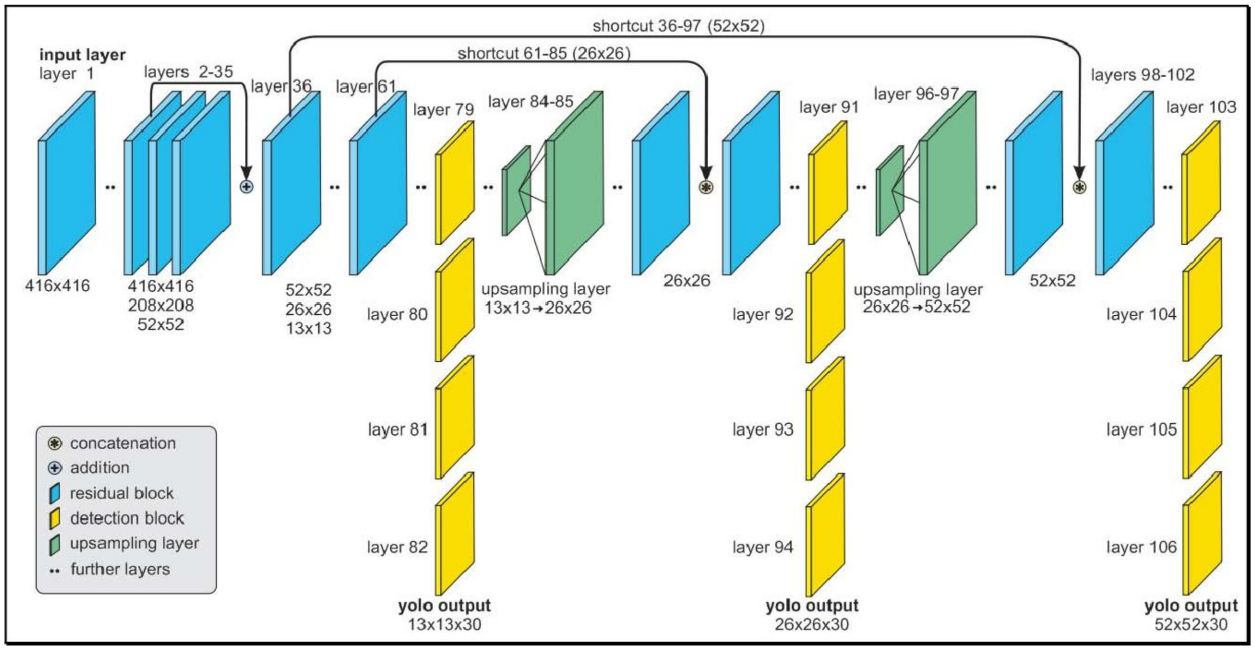
### 8.6.HardwareRequirement

- Running the mobile app requires Android 6 and above.
- Running the webapp requires any web browser.
- Hosting our server required an AWS server.
- Feed required three Raspberry Pi 3s, each with its own Arducam camera.

# 9. Algorithms and Data Structures

## 9.1.Algorithms

YOLOv3 uses a convolutional neural network to detect objects within an image. It processes an entire image by breaking it up into smaller and smaller boxes, looking for evidence of a target object within these boxes.



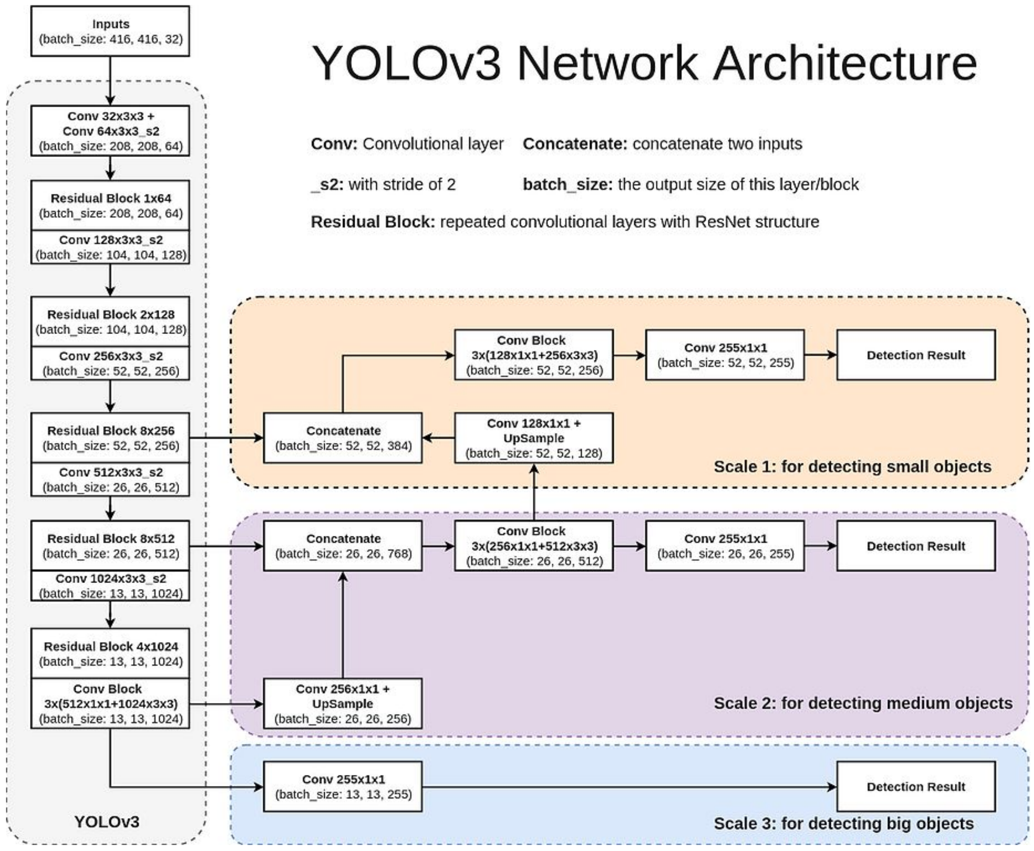
The above image illustrates how the algorithm breaks the image up into layers, creating layer branches for objects, followed by upsampling to magnify the input image, and finally dividing the input image into smaller and smaller boxes.

While processing the whole image, it generates a bounding box around target objects, tagged with a confidence value to illustrate how sure the bot is that the target object is found within that bounding box.

YOLOv3 uses Darknet as its neural net framework.

## 9.2.Data Structures

YOLO uses the same data structure across all implementations, as illustrated in the diagram below.


YOLOv3 Network Architecture
Conv: Convolutional layer
Concatenate: concatenate two inputs
_s2: with stride of 2
batch_size: the output size of this layer/block
Residual Block: repeated convolutional layers with ResNet structure
Inputs (batch_size: 416, 416, 32)
Conv 32x3x3 + Conv 64x3x3_s2 (batch_size: 208, 208, 64)
Residual Block 1x64 (batch_size: 208, 208, 64)
Conv 128x3x3_s2 (batch_size: 104, 104, 128)
Residual Block 2x128 (batch_size: 104, 104, 128)
Conv 256x3x3_s2 (batch_size: 52, 52, 256)
Residual Block 8x256 (batch_size: 52, 52, 256)
Conv 512x3x3_s2 (batch_size: 26, 26, 512)
Residual Block 8x512 (batch_size: 26, 26, 512)
Conv 1024x3x3_s2 (batch_size: 13, 13, 1024)
Residual Block 4x1024 (batch_size: 13, 13, 1024)
Conv Block 3x(512x1x1+1024x3x3) (batch_size: 13, 13, 1024)
YOLOv3
Concatenate (batch_size: 52, 52, 384)
Conv Block 3x(128x1x1+256x3x3) (batch_size: 52, 52, 256)
Conv 255x1x1 (batch_size: 52, 52, 255)
Detection Result
Conv 128x1x1 + UpSample (batch_size: 52, 52, 128)
Scale 1: for detecting small objects
Concatenate (batch_size: 26, 26, 768)
Conv Block 3x(256x1x1+512x3x3) (batch_size: 26, 26, 512)
Conv 255x1x1 (batch_size: 26, 26, 255)
Detection Result
Conv 256x1x1 + UpSample (batch_size: 26, 26, 256)
Scale 2: for detecting medium objects
Conv 255x1x1 (batch_size: 13, 13, 255)
Detection Result
Scale 3: for detecting big objects

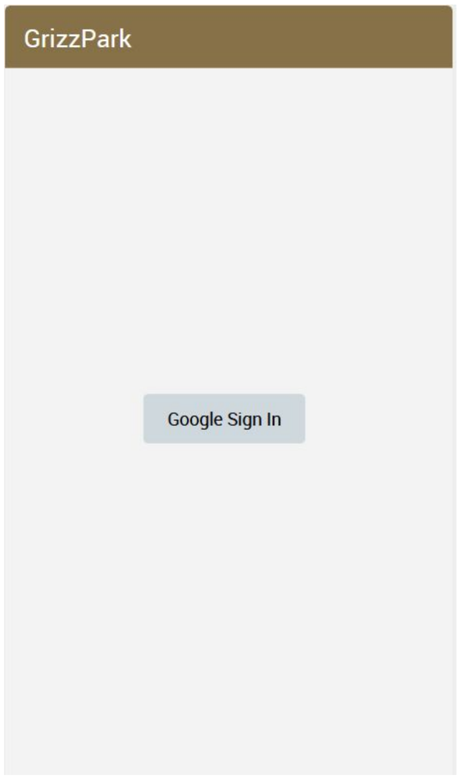
Blocks are processed by breaking up the image, parsing them, reassembling them, and scaling them to detect varying sizes of objects within the image. The residual block list exists only to pass on data to the next concatenation event, and all detection results are outputted to our Express.JS layer.

## 10.User Interface Design and Implementation

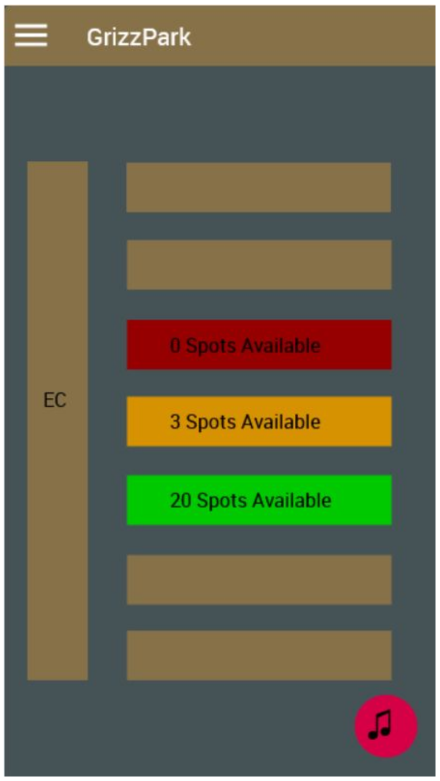
### 10.1.User Interface Design

Our final UI designs are as follows:

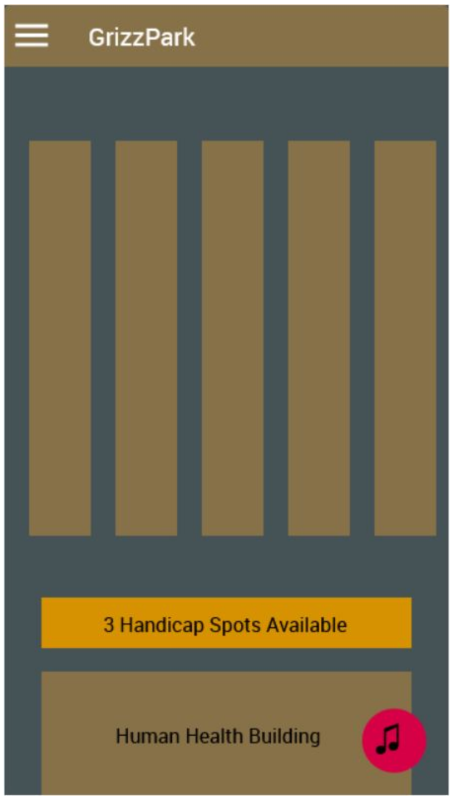
Sign In



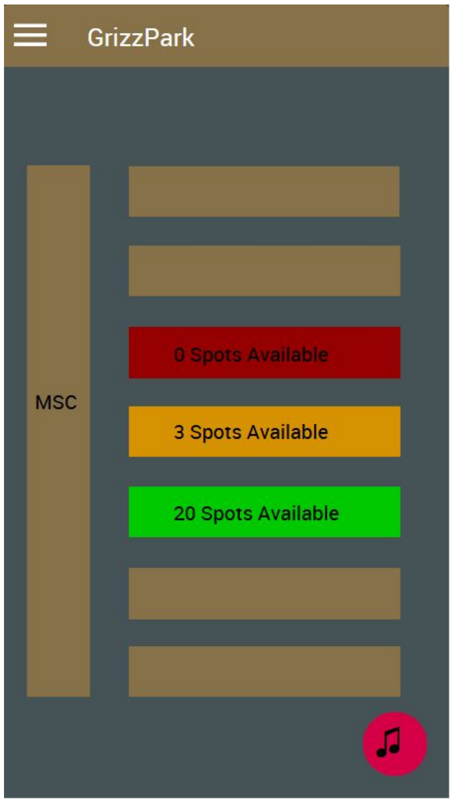
Engineering Center



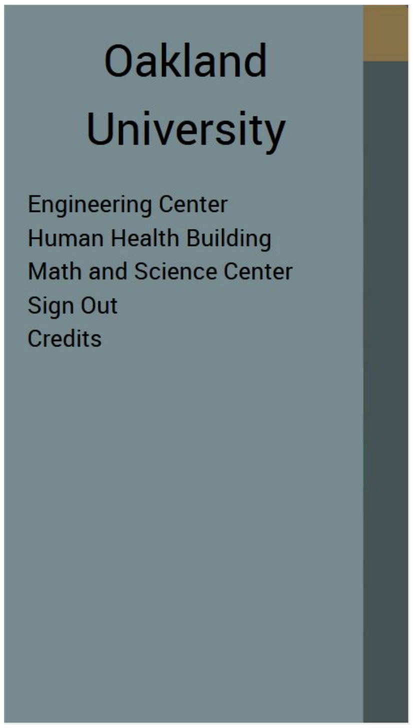
Human Health Building



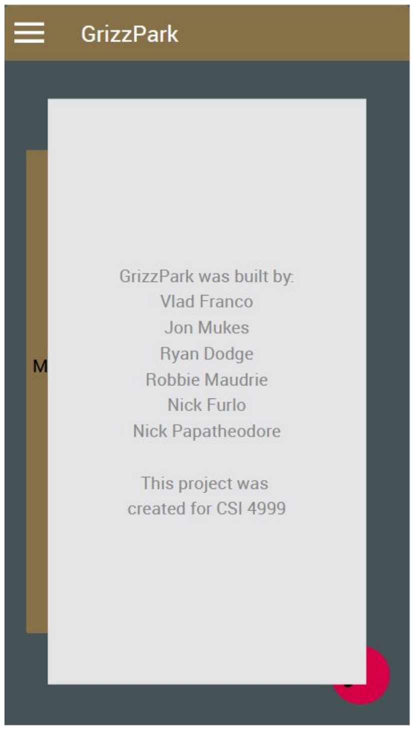
Math and Science Center



Menu



Credits



GrizzPark was built by:
Vlad Franco
Jon Mukes
Ryan Dodge
Robbie Maudrie
Nick Furlo
Nick Papatheodore

This project was
created for CSI 4999

After opening the app, the user clicks the google sign in button and picks their OU google account to sign in. The user then lands at the Engineering Center page. They can then use the menu by clicking the menu button in the top left corner. The menu will allow the user to navigate to the Human Health Building page, the Math and Science Center page, the Credits page, or Sign Out of GrizzPark.

The rows that are being analyzed by the bot will change color depending on how many spaces are free. Red for 0 spots, yellow for 1-5 spots, and green for 5+ spots. There is also text on each row being analyzed by the bot that displays the number of spots available.

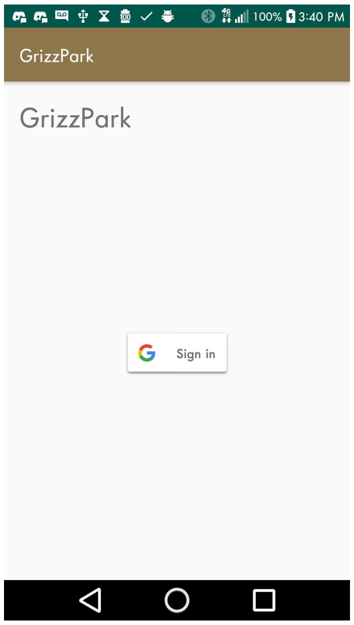
The button in the lower right hand corner of the parking pages will trigger a voice message to play that will list the number of spots available in each row.

The mockups were created with FluidUI.com.

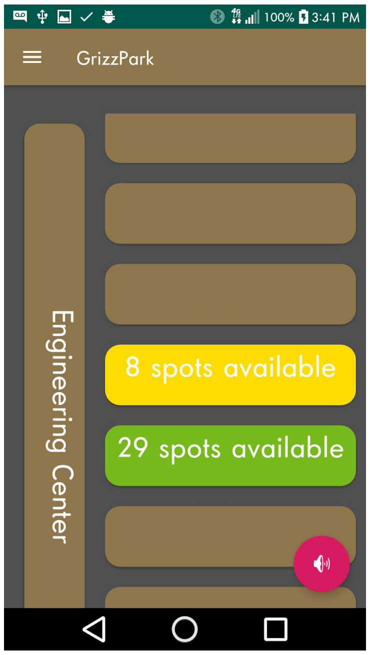
## 10.2.User Interface Implementation

The final version of our UI is shown below.

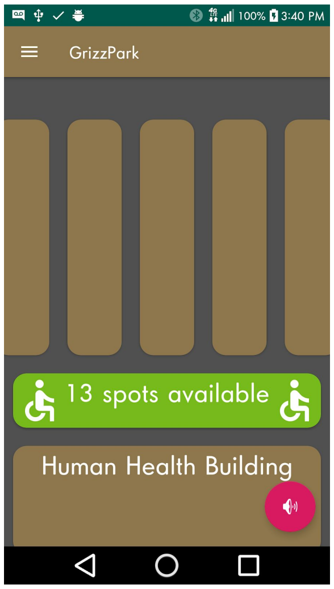
Sign in



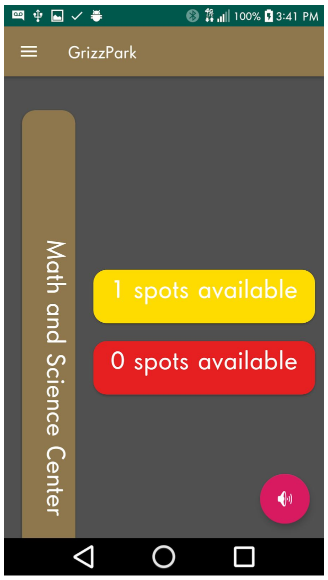
Engineering Center



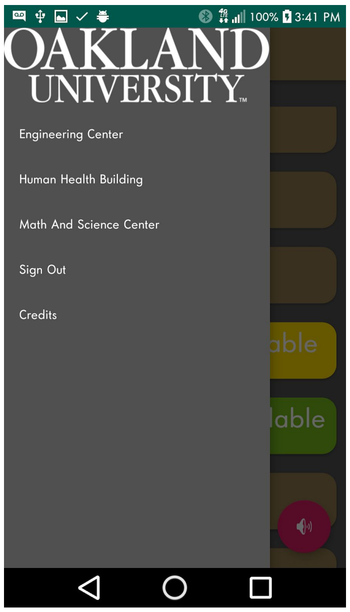
Human Health Building



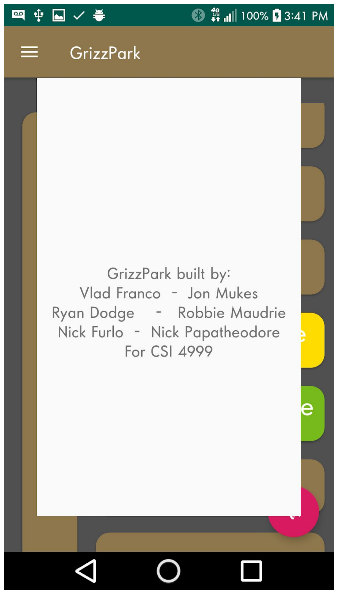
Math and Science Center



Menu



Credits



GrizzPark built by:
Vlad Franco - Jon Mukes
Ryan Dodge - Robbie Maudrie
Nick Furlo - Nick Papatheodore
For CSI 4999

After opening the app, the user clicks the google sign in button and picks their OU google account to sign in. The user then lands at the Engineering Center page. They can then use the menu by clicking the menu button in the top left corner. The menu will allow the user to navigate to the Human Health Building page, the Math and Science Center page, the Credits page, or Sign Out of GrizzPark.

The rows that are being analyzed by the bot will change color depending on how many spaces are free. Red for 0 spots, yellow for 1-9 spots, and green for 9+ spots. There is also text on each row being analyzed by the bot that displays the number of spots available.

The button in the lower right hand corner of the parking pages will trigger a voice message to play that will list the number of spots available in each row.

We created each of the layouts using Android Studio's constraint and linear layouts. Most of the layout creation was done using android studios UI creator, the rest was done with XML. Our layout follows Google UI design standards for material design.

# 11.Testing

## 11.1.Unit Test Architecture and Strategy/Framework

Each android app activity held test condition blocks, where our app would return certain values depending on whether or not functionality was where we wanted it to be.

Each raspberry pi would also be subject to testing, having set up conditions where our running job would append successful image transfers, failed transfers, and camera status to our log with each picture taken.

We often ran commands on our server to test certain functions of our bot, like pushing data and displaying bounding boxes.

## 11.2.Unit test definition, test data selection

We defined our unit tests according to the piece of our system we were performing tests on. For the Android app, each JSON request had a defined test block. In our Pi, it was in each push to the server. In our bot, much testing had to be done by hand.

## 11.3.System Test Specification

Our system test was as simple as looking out the window of the building nearest to the lot. We'd have our components' terminals pulled up, and all tests running successfully. If it really did work, we'd see the numbers on our app matching, or at least close to, the real numbers in the lot.

### 11.4.Test Reports per Sprint

We made an effort to clear any testing issues we encountered before each sprint. Any issues we encountered in sprints were not a result of lack of functionality, but usually had to do with configuration and microadjustments to our system.

# 12.ProjectManagement

### 12.1 Project Plan

1. Develop our backend framework, which included standing up our web and Amazon servers, programming our bot, setting up our Pi's.
2. Develop our admin website and begin getting permission from OUPD
3. Get Raspberry Pi and camera taking and sending images to our amazon server. Record 15k images to begin to train our bot.
4. Use the images to begin training our bot.
5. Once the bot is trained, begin feeding images to our bot to test out the functionality of it.
6. Create a database for our bot to feed the data into.
7. Develop an Android application that pulls the data from our database and outputs it to the users with the app.
8. Test for any bugs in the application.
9. Release app for public use.

### 12.2 Risk management

In order to minimize the risk while working on this project, we created a private GitHub which we would take the individual code that we wrote and upload for the others in the group. GitHub also kept previous pushes of our code incase something broke. Along with GitHub, we also made sure that our AWS server had more power and memory than would be needed. If our server crashed, then everything we had hosted on the server would have been lost. To preserve security for the AWS, we made sure to handle all transfers of the pem files through SSH.

# 13.Reference

https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/6

https://www.raspberrypi.org/documentation/configuration/camera.md

https://iot4beginners.com/how-to-capture-image-and-video-in-raspberry-pi/

https://www.jeffgeerling.com/blog/2017/fixing-blurry-focus-on-some-raspberry-pi-camera-v2-models

https://www.jeffgeerling.com/blog/2017/fixing-blurry-focus-on-some-raspberry-pi-camera-v2-models

https://pjreddie.com/darknet/yolo/

https://pjreddie.com/media/files/papers/YOLOv3.pdf

https://www.w3schools.com/

**Students and Team Members**
Ryan Dodge
Vlad Franco
Nick Furlo
Earl Maudrie
Jonathan Mukes
Nick Papatheodore