WPI Density Dodger

CS4518-A20 Group Project
Final Report

Manjusha Chava, Ivan Eroshenko, Irakli Grigolia, and Chris Mercer

Professor Guo

10/16/2020

A herd of WPI students traversing between classes with no social distancing (before COVID).

Table of Contents

Introduction & Importance

In this hard time of the COVID-19 pandemic, it is important that communities band together to help prevent the spread of the virus. The WPI community had initially felt the impact of several cases on campus in both students and faculty. While everybody who is allowed to be on campus is regularly being tested, it is still important to maintain social distancing and avoid large crowds of people, following CDC guidelines.

The biggest problem currently is that there is no way of knowing how many students are on campus or in a specific location at once. Being able to work and stay connected with each other on campus is very important to the WPI culture of collaboration, and this can only be done by keeping study spaces open while maintaining proper social distancing etiquette. Keeping students away from near or at-capacity buildings while allowing them to find a place to work together is the challenge that our team aims to solve, and is our way of showing care and giving back to the awesome WPI community.

Features and Screens

This application has a host of key features used to keep members of the WPI community safe on campus. There are three primary attributes that make this application the ideal tool for density tracking. The features include displaying the real-tim density of each building on campus on a map, providing both safe and fast navigation for the user from one location to another, and providing alternative routes and buildings to divert the user and guide them away from densely populated areas.

Density Map

The MainActivity is a map of the various buildings on campus, each with their own color coded marker to indicate the location and density level. The user can explore the map by moving it around and clicking on any marker. The map utilizes the Mapbox Library, which provides a customizable map for various applications. In order to implement the map in the application, MapBox Android SDK was used to create a dynamic map with the ability to draw clickable markers (WPI buildings), paths (directions from point A to point B), and support the ability to preload maps for offline use. Green markers indicate a low occupancy level, where yellow means moderate and red is high. The user's current location is also obtained within this activity in order to provide directions from their current location to their provided destination. Figure 1 below shows the map with the various markers and the user's current location.

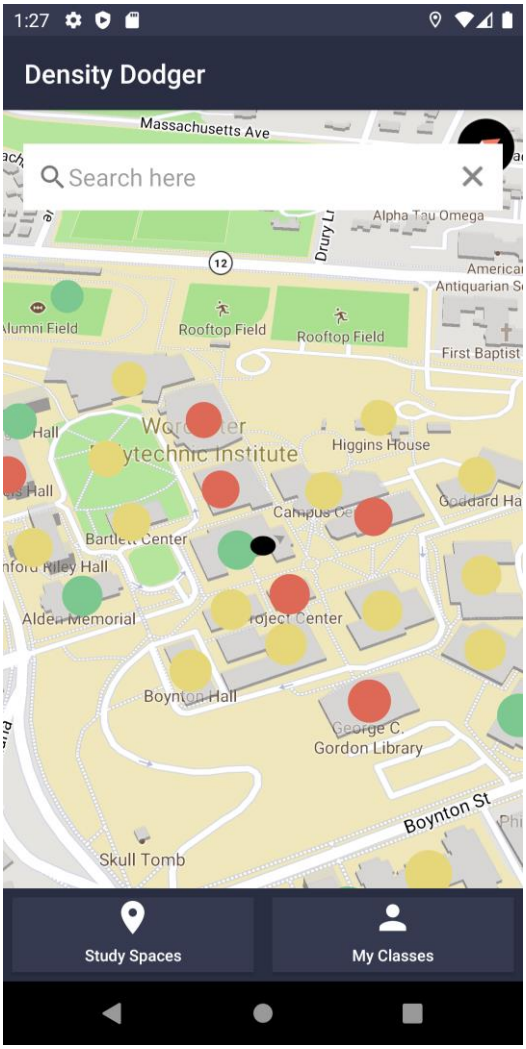

Figure 1. MainActivity with markers per building showing density.

Building Information

This screen provides information for each building based on the data that was provided by the database through the backend API call. Each building contains the number of people, density level, and an image. The user clicks on the marker building of their choice, and it will lead to Figure 2. The image is displayed through the use of the Picasso library to make an API request to the URL to receive the image. The occupancy bar changes based on the density level and percentage of people occupying the building. Figure 3 is what occurs when the "Get Directions" button is pressed. The screen provides a safe route and a fast route from the user's current location to the selected destination, indicated by the blue and pink lines respectively.

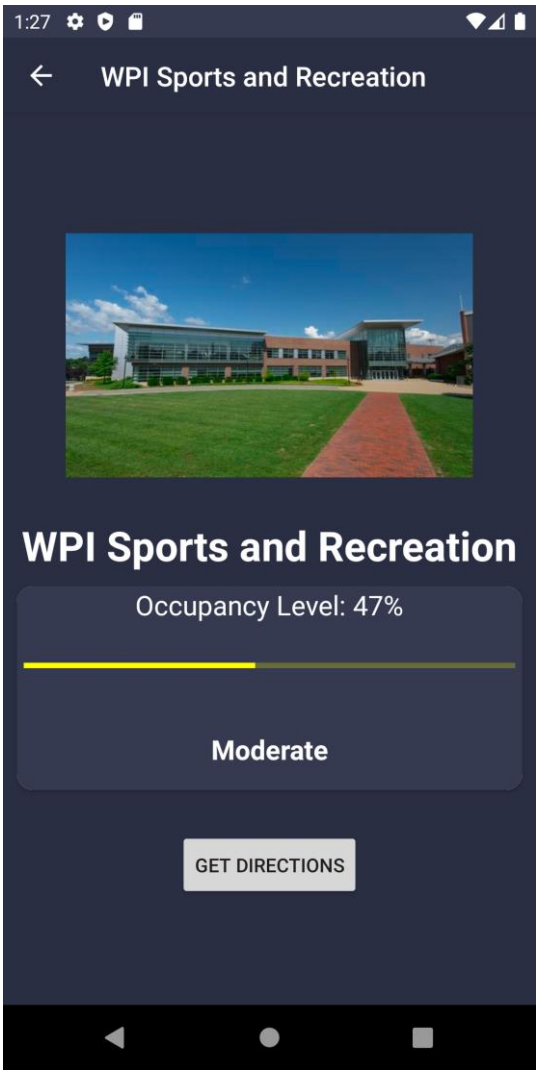

Figure 2. Building Information Screen when building markers are selected.

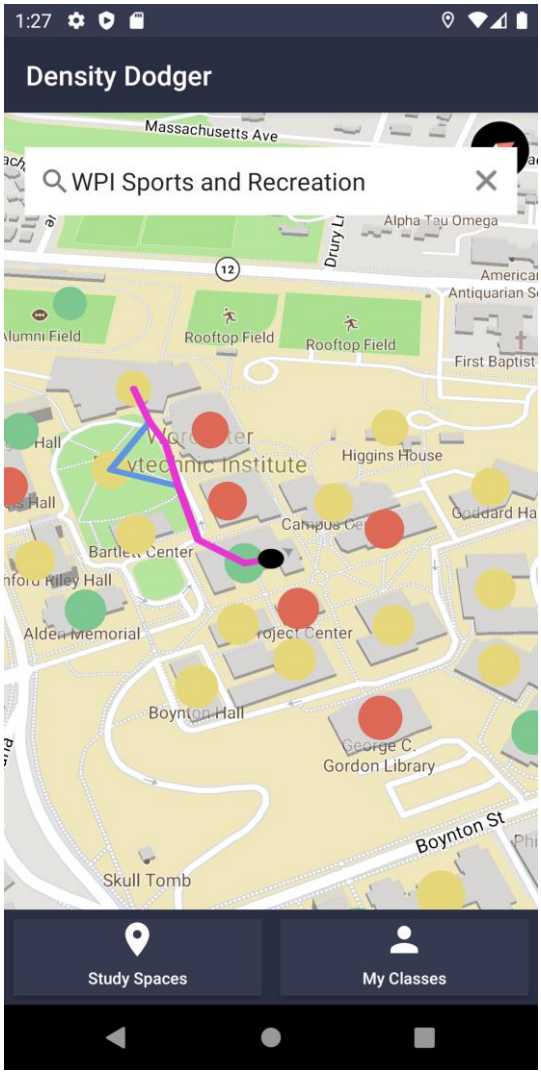

Figure 3. Safe and fast routes provided given a final destination.

Study Spaces

This screen provides a list of study spaces on campus and their respective occupancy levels. This feature is useful for students to search for a low occupancy building with ease. The screen uses a ListView of Card attributes, which is only populated if the building is a study space. If it is not, then it is not shown. Pressing a building card will direct the user to the Building Information activity, which the user can then get directions to their intended destination.

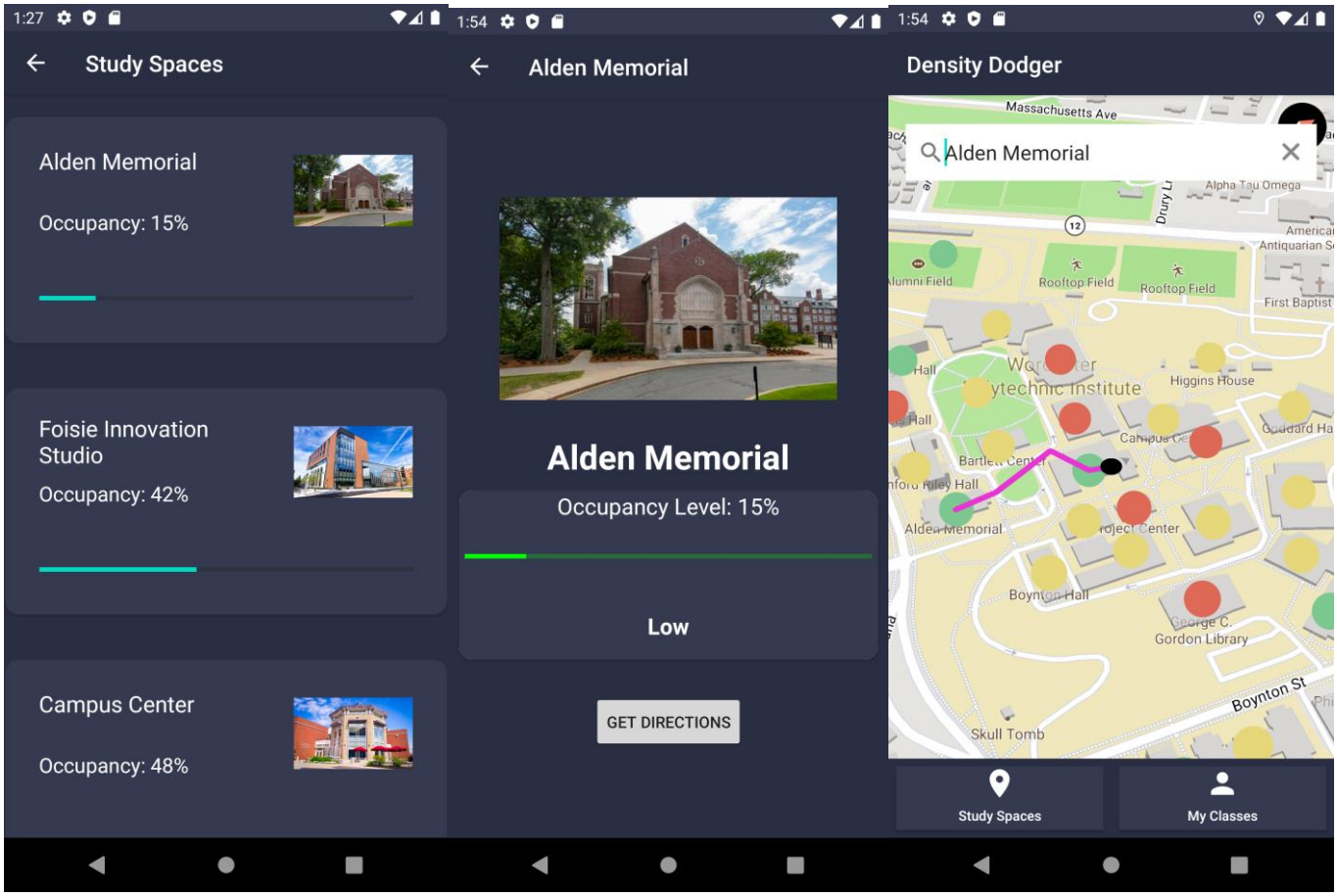

Figure 4 (left). List of buildings that are study spaces showing occupancy level. When clicked, the user is directed to the Building Information page, then to the map for navigation.

My Classes

The final screen displays three editable blocks that the user can use to fill in their current classes and classroom. This feature, shown in Figure 5, acts as a favorites page, where the user can get directions to their classrooms quickly and with great ease.

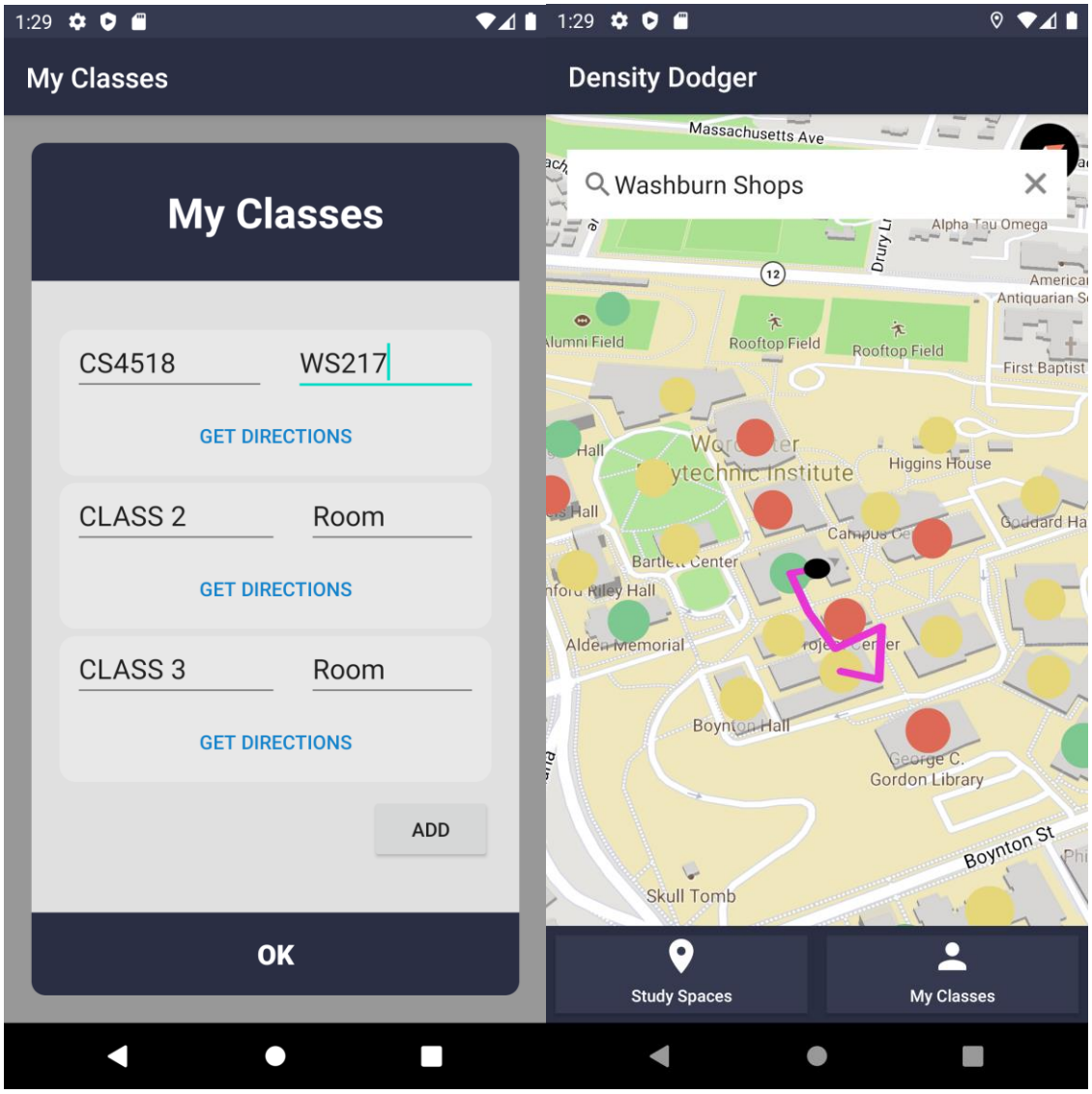

Figure 5 (left). The My Classes screen which holds the user's class information and provides easy navigation to frequently visited buildings.

Metrics and Performance Evaluation

Metrics Used

Since our app uses built-in GPS of a device, as well as provides a dynamic map of WPI campus on the main screen, CPU and memory usage must be measured in the Main screen, as well as when displaying routes from one location to the next. In addition to that, our client-side mobile application makes API requests to the back-end in 2 scenarios: 1) fetch all the building data to display on the map when the app is launched, 2) generate the fastest and safest routes from current user's location to one of the buildings. Thus, network usage and api requests speed must also be measured.

Evaluation Results

Figure 6 indicates the CPU and memory usage of our app while being in the main screen, which requires the most amount of resources. Memory usage on average is about 53 MB and CPU is about 51%.

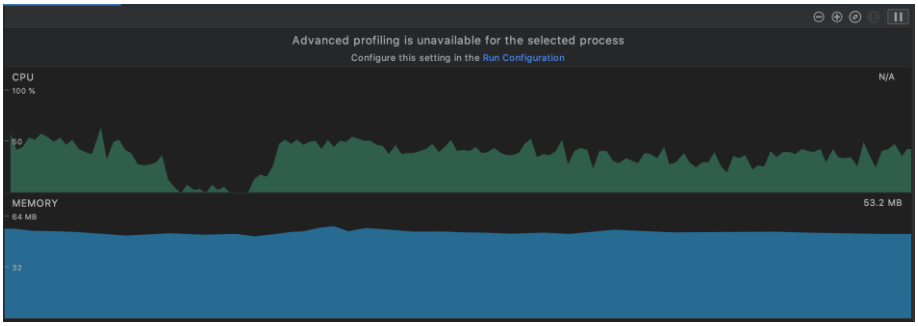

Figure 6. The CPU and Memory usage of the app in the Main screen.

Figure 7 indicates CPU, memory usage and network usage when displaying routes from current user's location to one of the buildings. As we can see CPU and memory do not get affected, which was a bit surprising for us. The time it took to make the API request and receive data back was about 1 second.

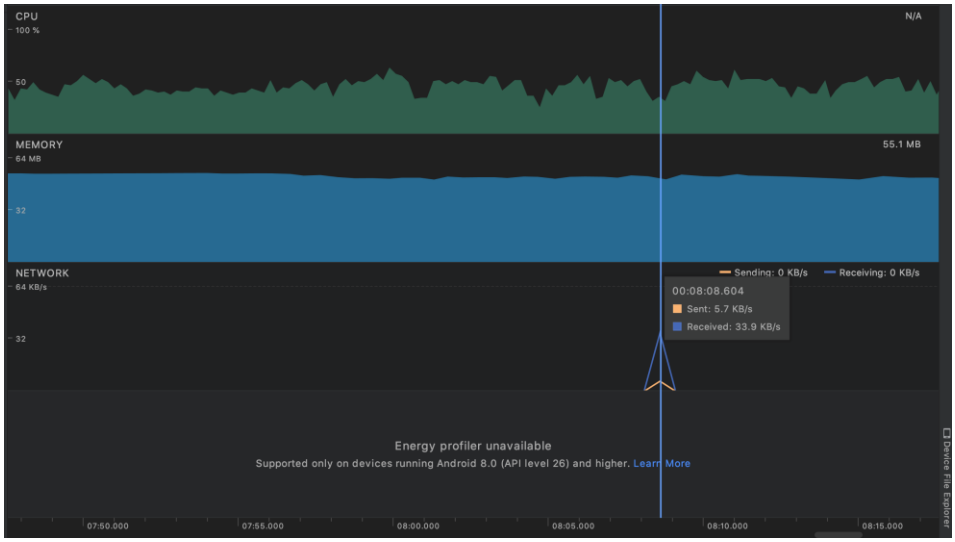

Figure 7. The CPU and Memory usage of the app in the Main screen.

Figure 8 indicates network usage when launching the app and making API requests to the back-end to display all the building nodes. The time it takes to make the request is about 1.2 seconds.

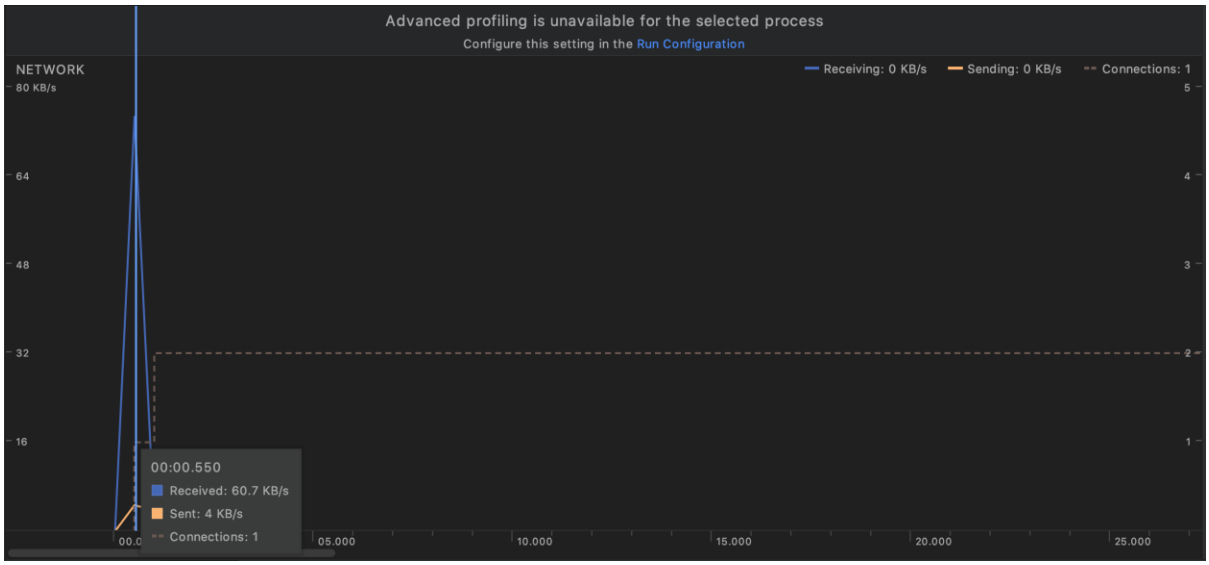

Figure 8. The network usage when making the API request to fetch all the building data.

Development Strategy

Design Strategy

Designing an efficient architecture design for an application is crucial, as it organizes the project in a proper way to run processes such as keeping track of logic, debugging, and testing. It allows for simplicity and easy maintenance, such as adding and removing features. The architecture

that this application will follow is MVC - Model View Controller design. Because this application relies more on our custom designed API to display data and provide paths, this architecture was best because it is lightweight and can provide support and easy flow of data between all layers. The figure below shows a passive model behavior of the MVC design, which was used in this application.

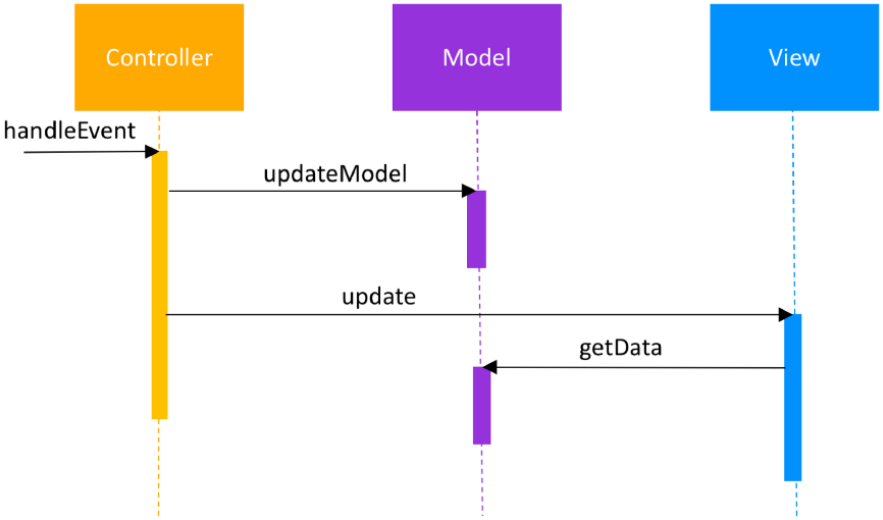

Figure 9. Behavior of the Passive Model case for Model-View-Controller.

The application has one main page, which will always be displaying a map and the building markers. There is navigation between the main activity and other activities when the two main buttons are clicked, which then passes through necessary data back and forth each activity. The MVC model is the most optimal design for this application because there is more data handling and manipulation between the Controller and Model layers, which is lightweight and out of sight until the data needs to be displayed in the View layer.

GPS Sensor

Density Dodger utilizes the mobile device's GPS sensor in order to obtain the current location of the user, which is used to navigate to various destinations across campus buildings.

Long-Term Persistence Strategy

The Density Dodger App is inherently data driven. While maps are nothing new, our app provides a new layer to give insight into where the busiest spots on campus are, down to the minute. This is achieved by a MySQL database with multiple tables. This table is on an isolated server whose credentials are only available to the API server (on a separate heroku instance.) What this means is high security due to the lack of database credentials on the device, and high uptime due to the API server and Database server not being on the same instance. We designed this with scalability in

mind, with more servers being easily added to the network. On the database server are the following tables:

Network Components

The backend for this application will be primarily written in python. While it is possible to execute SQL queries from the android phone to the server, it doesn't scale well and is not wise for security reasons. As such, we will be running a flask python server to be the intermediate logic between the phone and the data store. The MySQL instance will be on the same server as the flask web server. The web server will expose multiple operations through its API:

Table	Columns	Purpose
nodes	nodeID, placeName, latStart, latEnd, longStart, longEnd, type, floors, url, maxCap	Each node represents a building or significant location on campus. The nodeID is the WPI given short code, such as AK, which would have placeName Atwater Kent. The four lat and long points are coordinates for the two corners of each building. The type is either study or non-study. floors is the number of non-zero indexed floors, with maxCap being the max number of people per floor. Lastly, url is the url for the building's image.
route	nodeID, lat, long	Route nodes are points that represent path connections on the map that aren't points of interest. For example, going to the Rec Center from the Campus Center requires turning at multiple points, each of which would be found in the route table. The nodeID is a unique identifier used later on to track connections between nodes, and the lat and long are to mark where the points are on the map.
edges	edgeFrom edgeTo	Edges are the connections between building and route nodes. These represent where users can walk to get to a point. By using an algorithm, we can find paths between two given points.
sightings	epochTimestamp, lat, long	Sightings are instances of a person being logged as somewhere on campus. They are intentionally minimalist to avoid any privacy violations. The epochTimestamp is the time the user was logged as being on campus. The lat and long are the coordinates of where the user was on campus.

Call	Digests	Returns
/api/navigation/route?from_latitude=00.00&from_longitude=0	A source and a destination location in query string format. They may either be coordinates	Returns a JSON response with the route as a set of coordinates

0.00&to=LOCATION&safe=BOOLEAN	or a building ID. The safe parameter specifies if the fastest route should be taken (&safe=false) or the safest route, avoiding large groups of people (&safe=true)	
/api/buildings/all /api/buildings/<id>	Either 'all' to return all the buildings, or a specific <id> of a building to return just that building's data	Returns a JSON response with the buildings percent capacity per floor, along with the number of floors. Both the center points of the buildings, and their lat/long corner tuples are provided to properly display the points. The building data is loaded in on the hour, and the density data is refreshed every minute with a cronjob. This keeps the data up to date while still keeping the server performant. As identified in the database section, the image url and building type are also sent to make sure the campus can be updated in the future. Lastly, a percent average is given to indicate how populated the building is as a whole.
/api/density/add?latitude=00.00&longitude=00.00	A latitude and longitude coordinate for where the user is being logged. This is intentionally lacking any sort of ID to avoid privacy issues.	A 200 OK HTTP response if the entry was added. When it's added, the API server will build an insert into the Database server with the current epoch time. This ensures logged locations will only be considered when they are fresh.

This API is implemented using the Flask framework on top of Python 3.6. This enables the app to be easily understood and modified. Each critical function is in a separate class:

- database.py: A multitude of helper classes for establishing and terminating the DB connection, storing data, and retrieving entries.
- ddUtils.py: Has many debugging functions for testing and logging, as well as helper classes for storing a config file including database connection secrets.
- density.py: Consumes density data from the sightings table and processes it to ensure only the most recent entries are considered. Also places them next to the closest node, be it building or path.
- path.py: Performs a Dijikstra search through the edged graph, including the classes to make both the graph itself and the nodes that lie within it. Handles initiating the graph, nodes, and returning results correctly.
- start.py Combines all these files into one main method that can listen for requests via Flask. This main file is the entry point for the program, and will handle initializing the buildings, nodes, paths, and database connection. Will also terminate open connections gracefully on exit.

Workload Division

Chris - parsed building and route nodes and edges for the “secondary” part of WPI campus into CSV. Created the API for fetching building data and for displaying paths from current user’s location to a building. Provided means of adding and avoiding clusters of people on campus. In addition to that, implemented A* and Dijkstra algorithms to calculate both the fastest and safest routes based on on-campus “people-density” data.

Ivan - added building nodes, route nodes and edges for the Main Campus and saved it all in a CSV. Pair programmed with Manjusha to create a foundation for the app such as integrating MapBox, displaying 3D buildings, displaying the building nodes, and outlined how the app navigation is going to work. Furthermore, created a Search Bar and did the UI part of displaying paths between the buildings by making the API request to the back-end. Created Study Spaces screen.

Manjusha - Designed UI for MainActivity, Building Information, and My Classes screen, proposed frontend architecture approach, pair programmed with Ivan to integrate the Mapbox library and render map information, built functionality for Building Information and My Classes screen, wrote final document, and edited video.

Irakli - Set up MySQL database on Heroku platform. Used python script to connect to Database and create tables. Inserted data of nodes,edges,buildings, from csv file that was given from UI team.