University of Florida Testbed Initiative - Transit Components

Bus Bike Rack System (BDV31-977-113)

October 2020

Final Report

PM: Gabrielle Matthews, Florida Department of Transportation (FDOT)

PI: Yong-Kyu YK Yoon, University of Florida (UF)
Team members:
- Todd Schumann (UF)
- Hyun Ho Cho (UF)
- Austin Kee (UF)
- Xiao Qin (UF)

UFTI Director: Lily Elefteriadou

DISCLAIMER

The opinions, findings, and conclusions expressed in this publication are those of the authors and not necessarily those of the State of Florida Department of Transportation.

METRIC CONVERSION TABLE

SI* (MODERN METRIC) CONVERSION FACTORS

APPROXIMATE CONVERSIONS TO SI UNITS

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
in	inches	25.4	millimeters	mm
ft	feet	0.305	meters	m
yd	yards	0.914	meters	m
mi	miles	1.61	kilometers	km
		AREA		
in^2	square inches	645.2	square millimeters	mm^2
ft^2	square feet	0.093	square meters	m^2
yd^2	square yard	0.836	square meters	m^2
ac	acres	0.405	hectares	ha
mi^2	square miles	2.59	square kilometers	km^2
		VOLUME		
fl oz	fluid ounces	29.57	milliliters	mL
gal	gallons	3.785	liters	L
ft^3	cubic feet	0.028	cubic meters	m^3
yd^3	cubic yards	0.765	cubic meters	m^3
	NOTE: volumes greater than 1000 L shall be shown in m^3			
		MASS		
oz	ounces	28.35	grams	g
lb	pounds	0.454	kilograms	kg
T	short tons (2000 lb)	0.907	megagrams (or "metric ton")	Mg (or "t")
		TEMPERATURE (exact degrees)		
^oF	Fahrenheit	5 (F-32)/9 or (F-32)/1.8	Celsius	^oC
		ILLUMINATION		
fc	foot-candles	10.76	lux	lx
fl	foot-Lamberts	3.426	$candela/m^2$	cd/m^2
		FORCE and PRESSURE or STRESS		
lbf	poundforce	4.45	newtons	N
lbf/in^2	poundforce per square inch	6.89	kilopascals	kPa

APPROXIMATE CONVERSIONS FROM SI UNITS

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
mm	millimeters	0.039	inches	in
m	meters	3.28	feet	ft
m	meters	1.09	yards	yd
km	kilometers	0.621	miles	mi
		AREA		
mm^2	square millimeters	0.0016	square inches	in^2
m^2	square meters	10.764	square feet	ft^2
m^2	square meters	1.195	square yards	yd^2
ha	hectares	2.47	acres	ac
km^2	square kilometers	0.386	square miles	mi^2
		VOLUME		
mL	milliliters	0.034	fluid ounces	fl oz
L	liters	0.264	gallons	gal
m^3	cubic meters	35.314	cubic feet	ft^3
m^3	cubic meters	1.307	cubic yards	yd^3
		MASS		
g	grams	0.035	ounces	oz
kg	kilograms	2.202	pounds	lb
Mg (or "t")	megagrams (or "metric ton")	1.103	short tons (2000 lb)	T
		TEMPERATURE (exact degrees)		
^oC	Celsius	1.8C+32	Fahrenheit	^oF
		ILLUMINATION		
lx	lux	0.0929	foot-candles	fc
cd/m^2	$candela/m^2$	0.2919	foot-Lamberts	fl
		FORCE and PRESSURE or STRESS		
N	newtons	0.225	poundforce	lbf
kPa	kilopascals	0.145	poundforce per square inch	lbf/in^2

https://highways.dot.gov/research/resources/research-library/modern-metric-conversion-factors

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle University of Florida Testbed Initiative - Transit Components: Bus Bike Rack System (BDV31-977-113)		5. Report Date Oct. 2020 6. Performing Organization Code:
7. Author(s) Yong-Kyu Yoon, Todd Schumann, Hyun Ho Cho, Austin Kee, Xiao Qin		8. Performing Organization Report No.
9. Performing Organization Name and Address University of Florida 946 CENTER DR GAINESVILLE, FL 32611		10. Work Unit No. 11. Contract or Grant No. BDV31-977-113
12. Sponsoring Agency Name and Address The State of Florida Department of Transportation, 605 Suwannee St., Tallahassee, FL 32399		13. Type of Report and Period Draft Final Report, 5/20/19 – 10/31/20 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

The Alachua County, FL, has the second highest bicycle mode share in the state. Bicycle riding combined with bus riding, i.e., multimodal commuting, is very popular in the City of Gainesville (COG), FL, while quantified information of usage is very limited. Although some infrastructure could be upgraded, there is no scientific data and ground to make a good decision. For example, for certain bus routes, the standard two-slot bike racks may not be sufficient because of the large number of bus-bike commuters, for which replacing the existing ones with three-slot bike racks are desirable. As COG is a college town, the bus bike rack usage is varying daily, weekly, seasonal, and yearly. Taking that into account, a technical system to access the usage data and help decision is imperative.

A UF team has developed a remote real-time sensing system for the detection of bike presence on the bus bike rack using pressure sensors and readout electronics in this project. For the consideration of future usage by potential bike riders, BikeRide mobile app has been developed. The report details the hardware of sensing system, the developed app, and the bus bike rack usage data in different time scales, e.g., day, week, and season, and in different bus routes.

The purpose of this study is to develop a bus bike rack sensing system that can detect bicycle usage per rack position and perform usage analysis and behavior study of bike users.

The outcomes are expected to help COG increase user satisfaction of bus-bike multimodal commuters, enhance attractiveness of multimodal commuting by enabling better trip planning for bus-bike riders, and maximize cost effectiveness of infrastructure investment with help from UF's advanced information technology (IT).

17. Key Words	18. Distribution Statement
Bike Rack, Sensor, Mobile App, Multimodal Commuting	No restrictions.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	59	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized.

Acknowledgements

This work was completed under FDOT contract number BDV31-977-113. The authors would like to acknowledge the following people for their invaluable guidance and help in the completion of this study: Gabrielle Matthews (FDOT), David Sherman (FDOT), Yong-Kyu Yoon (UF), Nithin Agarwal (UF), Todd Schumann (UF), Hyun Ho Cho (UF), Austin Kee (UF), Xiao Qin (UF), Lily Elefteriadou (UF), and Jesus M. Gomez (COG).

Executive Summary

Alachua County, FL, has the second highest bicycle mode share in the state, and passengers in this city employ bicycle racks on RTS buses approximately 750 times per day. Despite this massive usage in Gainesville, FL, the system frequently faces capacity constraints. Due to safety[1] and maneuverability concerns, RTS buses are equipped with two-bike racks, which mean only two bicycles can be accommodated at the same time. Although there are already three-bike racks which can be installed in front of the bus, there is hesitancy to purchase the higher capacity racks because there is only limited information about how many people use and need them. Moreover, a limited budget also makes this problem unsolved.

Existing bike rack technology has two major shortfalls, which are just providing the deployment of bike racks and no real-time data on bike rack capacity. With this inconvenience, people who would like to use bike rack cannot use the bike rack if there is no availability until the next bus is coming for about 30 minutes or so.

From this idea, our group has installed the bike rack sensing system from Jun. 2017 to Dec. 2018 to get information which can be used to detect when bus bike racks are used. Now, we have data from the bike rack sensing system so that we can perform a usage analysis and behavior study with the data collected. Furthermore, we have made the BikeRide Mobile Application which can be used for checking real-time bike rack availability, and bike rack users can take advantage of this technology for their efficient time management.

We have reached the following conclusions. The bus bike rack usage is distributed during the daytime from 7 am to 6 pm, which is aligned with the commuting and/or school time. Weekday usage exceeds the weekend usage, which indicates most of bus bike rack usage is for commuting during weekdays. It is interest that the usage on Saturdays during fall semester is much more than the other semesters (spring and summer) because of the college football games in town. Usage data for several bus routes has been included. Certain bus routes are very popular for the bus bike multimodal commuters, which would justify for replacing the existing two-slot bike racks with three-slot ones.

[1] The primary safety concern is the obstruction of headlights.

Table of Contents

DISCLAIMER.....II

METRIC CONVERSION TABLE.....III

TECHNICAL REPORT DOCUMENTATION PAGE.....IV

ACKNOWLEDGEMENTS.....V

EXECUTIVE SUMMARY.....VI

LIST OF FIGURES..... XI

LIST OF TABLES..... XIII

PART A: INTRODUCTION AND BACKGROUND INFORMATION.....1

1 Introduction.....1

1.1 Background Statement.....1

1.2 Project Objectives.....2

1.3 Supporting Tasks and Deliverables.....2

1.3.1 Bicycle Rack Capacity Sensing System.....2

2 Bicycle Rack Occupancy Sensor System.....3

2.1 Sensor System Fabrication.....3

2.2 Sensing Concept.....4

2.3 System Design.....5

2.4 Installation.....7

2.5 Bill of Materials.....8

Part B: BIKERIDE MOBILE APPLICATION.....11

3 BikeRide Mobile Application.....11

3.1 Introduction.....11

3.2 Requirements.....11

3.2.1 User Requirements.....11

3.2.2 Functional Requirements.....11
3.2.3 Non-Functional Requirements.....12
3.3 Wireframe.....12
3.4 User Interface & Features.....14
3.4.1. Main Page.....14
3.4.2. Get a User's Current Location.....14
3.4.3 Search Buses by Route Number.....15
3.4.4. Get a Bus's Current Location.....16
3.4.5 Data Source.....17
3.4.6 API Documentation.....18
PART C: USAGE ANALSIS of BUS BIKE RACK.....21
4 Usage Analysis of Bus Bike Rack.....21
4.1 Introduction.....21
4.2 Overall Usage Analysis.....21
4.2.1 Daily Usage Analysis.....22
4.2.2 Weekly Usage Analysis.....23
4.2.3 Monthly Usage Analysis.....24
4.2.4 Seasonal Usage Analysis.....25
4.2.5 Saturday Usage Analysis.....25
4.3 Individual Bus Route-Based Usage Analysis.....27
4.3.1 Bus #2.....27
4.3.2 Bus #4.....28
4.3.3 Bus #8.....29
4.3.4 Bus #16.....30

4.3.5 Bus #13.....31
4.3.6 Bus # 22.....32
4.4 Discussion and Conclusion.....33
Part D: CURRET STATE and FINAL RECOMMENDATIONS.....34
5 Current State of the Project.....34
5.1 General Architecture.....34
5.2 Mobile Application.....34
5.3 Web API.....35
5.4. Moving Forward with Deployment and Long-Term Support.....36
6 Configuring, Building, and Installing the BikeRide Application.....39
6.1 Introduction.....39
6.2 Installing Flutter SDK.....39
6.2.1 MacOS/Linux.....39
6.3 Installing Android Studio.....40
6.3.1 MacOS.....40
6.3.2 Linux.....40
6.4 Configuring Android Studio.....40
6.5 Wrapping up the Installation.....41
6.6 Cloning the Project.....42
6.6.1 Using Android Studio's VCS.....42
6.6.2 Using Git CLI.....42
6.6.3 Manually Adding the Project to Android Studio.....42
6.7 Building the Project.....43
6.7.1 Ensuring the Project is Functional.....43
6.7.2 Generating App Code and Building the App for Target Devices.....43
6.8 Disclaimer.....43

6.9 Android Minimum Requirements OS……44

6.10 iOS Minimum Requirements OS……44

PART E: SUMMARY AND CONCLUSION……45

7 Summary and Conclusion……45

7.1 Summary……45

7.2 Conclusion……46

List of Figures

Figure 2-1. Fabrication steps of the sensor assembly.....3

Figure 2-2. Batch fabrication of sensor assemblies.....4

Figure 2-3. Wiring diagram illustrating the occupancy sensing concept.....5

Figure 2-4. Wiring diagram of the system, excluding power connections.....6

Figure 2-5. Assembled system secured in a plastic box, ready for installation.....6

Figure 2-6. (Left) Side panel next to the driver's feet that contains the port to the battery compartment (outlined). (Right) View from the battery compartment of the same port (outlined).....7

Figure 2-7. (Left) Sensor wire fed through the front bumper of the bus to the front undercarriage. (Right) Port under the front undercarriage where the sensor wires are fed to the battery.....8

Figure 3-1. Wireframe. Main interface (left), user's location(top), function of searching buses (bottom-left), search result (bottom-middle), bus's location (bottom-right).....13

Figure 3-2. User interface and feature.....14

Figure 3-3. User's current location.....15

Figure 3-4. Search Buses by route number.....16

Figure 3-5. Get Bus's current location.....17

Figure 4-1. List of the bus number and system ID of the 18 buses used for the data analysis.....21

Figure 4-2. Daily bike rack usage.....22

Figure 4-3. Weekly bike rack usage.....23

Figure 4-4. Monthly bike rack usage.....24

Figure 4-5. Seasonal bike rack usage.....25

Figure 4-6. Saturday bike rack usage.....26

Figure 4-7. The map (a) and the bike rack usage data (b) of Bus #2.....27

Figure 4-8. The map (a) and the bike rack usage data (b) of Bus #4.....28

Figure 4-9. The map (a) and the bike rack usage data (b) of Bus #8.....29

Figure 4-10. The map (a) and the bike rack usage data (b) of Bus #16……30

Figure 4-11. The map (a) and the bike rack usage data (b) of Bus #13……31

Figure 4-12. The map (a) and the bike rack usage data (b) of Bus #22……32

Figure 5-1. UML activity diagram for BikeRide mobile application……34

Figure 5-2. UML activity diagram for the BikeRide web API……35

LIST OF TABLES

Table 2-1. Bill of materials to build 20 systems 9

Table 2-2. Cost of the cellular plan for 20 systems.....10

Table 3-1. User Requirements.....11

Table 3-2. Functional requirements.....12

Table 3-3. Non-functional requirements.....12

Table 3-4. Data source.....18

PART A: INTRODUCTION AND BACKGROUND INFORMATION

1 Introduction

1.1 Background Statement

The City of Gainesville (COG) Regional Transit System (RTS) provides bus service to the University of Florida (UF), Santa Fe College (SF), portions of unincorporated Alachua County, and the City itself. The partnership between these entities has produced remarkable results over the last two decades. National Transit Database (NTD) statistics show the region has the 14th most trips per capita in the nation[2] and the most trips per revenue mile in Florida.[3]

The COG urbanized area (UA) has been equally successful in encouraging bicycle usage and integrating this usage with transit to expand the reach of both. Data from the American Community Survey shows that Alachua County has the second highest bicycle mode share in the state.[4] Automatic Passenger Counter (APC) data indicates that passengers employ bicycle racks on RTS buses approximately 750 times per day. Based on average daily ridership, at least 2% of all trips involve the use of a bicycle. This is remarkable when you consider the statewide bicycle mode share is barely above 0.7%.

The urbanized area's remarkably high mode shares of both transit[5] and bicycling[6] has not been without its challenges. The system frequently faces capacity constraints. Because of safety[7] and maneuverability concerns, typical bus bicycle racks can only store two bicycles at a time. Though FDOT recently granted transit agencies the ability to transition to larger, three-position bicycle racks, it is still in the incipient stages of being adopted. There is hesitancy to purchase the higher capacity racks because there is only limited intelligence on their need. Anecdotal evidence, though, does suggest a need for these larger capacity bicycle racks – RTS customer service representatives regularly receive complaints from individuals who miss the bus because the bicycle rack is full and they will not or cannot[8] leave their bicycle at the bus stop. However, given limited capital funding expenditures[9], this is not sufficient evidence to justify the change.

As stated above, RTS relies on its APC units to determine bicycle usage. This technology has two major shortfalls:

---

[2] https://fivethirtyeight.com/datalab/how-your-citys-public-transit-stacks-up/

[3] http://www.fdot.gov/transit/Pages/2015TransitHandbook.pdf

[4] https://factfinder.census.gov/faces/tableservices/jsf/pages/productview.xhtml?pid=ACS_15_5YR_B08006&prodType=table

[5] http://www.fdot.gov/planning/trends/special/acs012816.pdf

[6] http://www.cityclock.org/urban-cycling-mode-share/#.WJSIHlMrKUk

[7] The primary safety concern is the obstruction of headlights.

[8] There are many reasons for this. The most common reasons posited are inadequacy of the stop to store a bicycle (both safety and space) and the need for the bike on the other end of the trip. As one example, it is quite possible to have classes in abutting periods on UF's campus that require a bicycle to be traveled between in a timely manner.

[9] The three position bicycle racks vary widely in cost but can be purchased for approximately $600 to $800.

1. It only shows rack deployment, and
2. The data is post-processed, thereby providing no real-time insight on available bike rack capacity.

The first shortfall leads to serious undercounting of bicycle usage.[10] The second shortfall causes users who want to use a bicycle to guess whether it will be possible since there is no live information on availability. Considering several bus stops are only served every 30- to 60-minutes, there is a severe penalty for users that wait to use a bicycle rack only to be denied.

For this initiative, we have performed the initial work on the bus bike rack sensing system from Jun. 2017 – Dec. 2019. We have successfully implemented pressure sensing systems in 18 COG buses. Some data have been acquired between May 2017 and Dec. 2018, which have been reported during the project meeting.

1.2. Project Objectives

The purpose of this study is to develop a bus bike rack sensing system that can detect bicycle usage per rack position and perform usage analysis and behavior study of bike users. This will fulfill the following objectives:

1) Increase use of ITS technology in transit.
2) Improve transit and bicycle mode attractiveness by enabling better trip planning and increasing user satisfaction.
3) Maximize cost effectiveness of infrastructure investments by better understanding travel patterns of bicycle users.

1.3 Supporting Tasks and Deliverables

1.3.1 Bicycle Rack Capacity Sensing System

UF developed the smart bus bike rack system with a pressure sensor integrated in each rack slot. However, for the sensor reliability and later maintenance, a commercially available pressure sensor is adopted for the sensor deployment to the bus bike rack.

[10] Most RTS directional patterns have between 20 to 30 stops. Though likely improbable, a trip with 30 stops where the bike rack is deployed and fully utilized at the first stop could have 58 trips occur with passengers using a bicycle, but database records would only show one usage. Equally, you could have some large, untold number of trip denials because the bicycle rack is fully occupied yet the database's single record would imply limited demand.

2 Bicycle Rack Occupancy Sensor System

2.1 Sensor System Fabrication

To enable the systems built during this project to continue operation after completion, we elected to use commercial off-the-shelf (COTS) sensors with a simple fabrication. This allows replacement of the sensors once the systems are handed off to the responsible agency.

The base sensor is a force sensitive resistor. When pressure is applied to the sensor head, the resistance of the sensor drops sharply. This is read by the system using a potential divider with a large series resistor to monitor occupancy and a small series resistor to monitor sensor failure (more in the sensing concept section).

As the sensor head is fragile to sharp contact, the sensor is protected by aluminum bars. These also serve to concentrate the pressure to the sensor head, allowing a small, 0.25” force sensitive resistor to measure a 12” area along the tire support on the rack. Silicone adhesive is used to assemble the sensor assembly and to seal the sensor from the elements. The fabrication is as follows:

A) Solder wire connections to COTS sensor and crimp opposite ends into RJ11 connector. Leave approximately 10-15 feet of wire.
B) Use silicone to secure the sensor to the aluminum bar with the head approximately at the center. Also secure the end of the wiring to the aluminum bar. Allow silicone to dry.
C) Use silicone to seal the sensor and exposed connections. Allow silicone to dry.
D) Create the plunger over the sensor head using silicone. Add height-matched silicone bumps at each end of the aluminum bar. Allow the silicone to dry.
E) Using a thin layer of silicone, secure the top aluminum bar to the rest of the sensor. Do NOT press into place. Allow silicone to dry.
F) Coat a layer of silicone around each end of the sensor assembly to ensure the sensor stays intact. Allow silicone to dry.

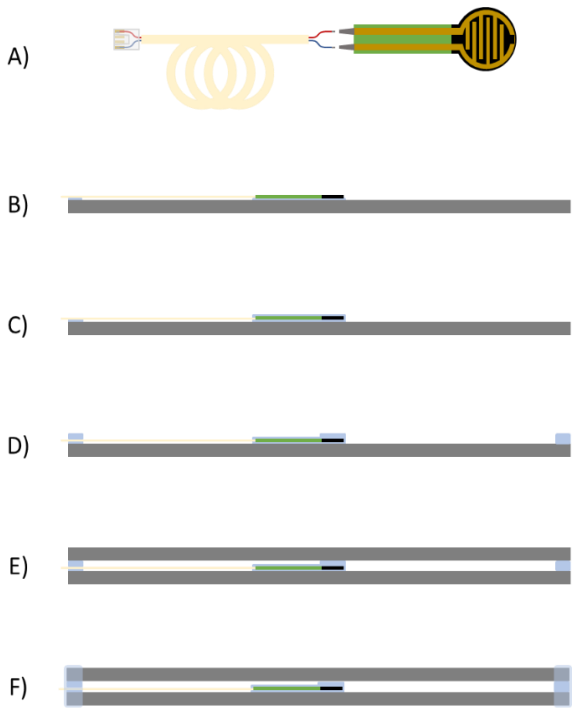


Figure 2-1. Fabrication steps of the sensor assembly.

Although the fabrication is simple and very tolerant especially as the sensors are individually thresholded, the full process is time consuming since the silicone will take about 12-24 hours to fully cure at each step. It is therefore recommended to produce many sensors simultaneously to increase throughput. Allowing the silicone to fully cure is critical during assembly as it will shrink slightly during the curing process, leading to the sensors reading as always triggered. By fully curing the silicone at each step, this effect is severely reduced, allowing a larger measurement range for the sensors.

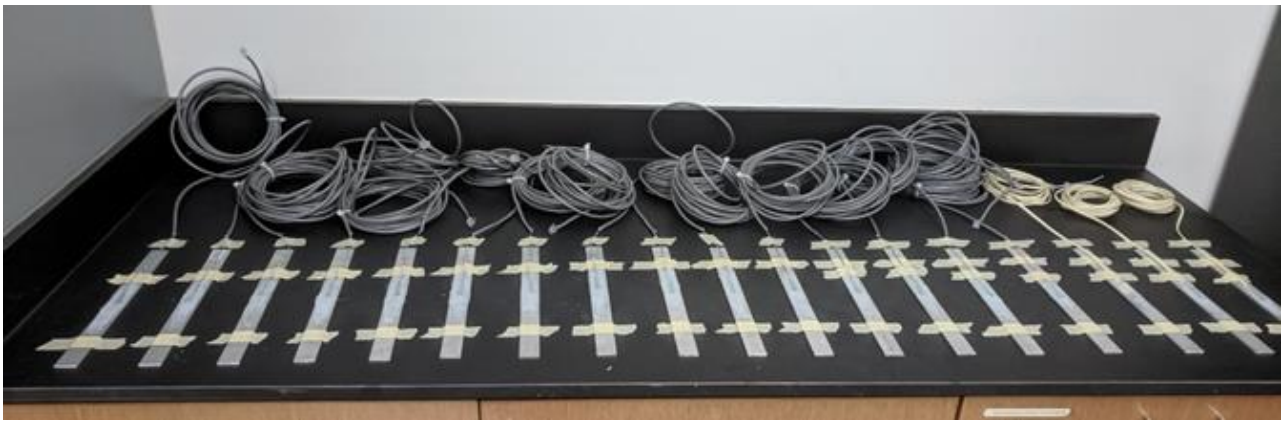


Figure 2-2. Batch fabrication of sensor assemblies.

2.2 Sensing Concept

Due to the large (orders of magnitude) change in resistance of the COTS sensor, extreme accuracy is not necessary when measuring resistance, eliminating the need for a Whetstone bridge configuration. Instead, a large resistor (100 kΩ) is placed in series with the sensor. A constant voltage is placed across the two resistors and the voltage is measured between them. When untriggered, the sensor is highly resistive and thus the measured voltage is close to the applied voltage. When triggered, the sensor's resistance drops much below the static resistor, and the measured voltage becomes much smaller than the applied voltage. By setting a threshold value for the measured voltage, the system determines whether the slot is occupied.

To detect cases where a sensor is shorted (the voltage is bypassing the sensor element), a second resistor is connected in the same manner, but with a resistance comparable to the lowest resistance the sensor can achieve (approximately 2 kΩ for these COTS sensors). When the voltage measured between this resistor and the sensor is measured, it should read a non-zero value. However, if the sensor is shorted, a very low value will be read as the voltage drop across sensor itself is 0 V. Again, this value is thresholded to determine if any given sensor is malfunctioning.

Unfortunately, due to the high resistance of the sensor when not triggered, checking for open circuits (sensor has become disconnected) is not feasible without additional circuitry such as buffering opamps. In this case, the systems will rely on user-reported problems. However, thus far, instances of this happening have been rare.

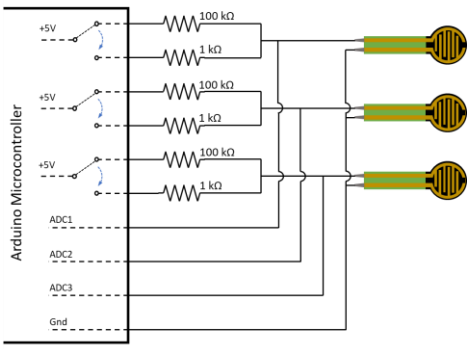


Figure 2-3. Wiring diagram illustrating the occupancy sensing concept.

2.3 System Design

Due to incompatibility with existing systems and reluctance of vendors to allow our sensors to interface with their systems, a stand-alone system was developed to read the bicycle rack occupancy sensors. The base platform was chosen to be an Arduino. This keeps the components easily replaceable, but also offers the option of designing a custom circuit board if the systems will be deployed in a high enough quantity to warrant it.

For communication, a SIM900 module was chosen. The SIM900 is limited to the 2G network, although this is not an issue with the amount of data being communicated. As T-Mobile is the only carrier currently supporting its 2G network, Ting, which uses T-Mobile's network, was chosen as the carrier, due to their flexibility with a high number of low-data devices.

Since the mobile network can have blind spots, a copy of all data and log files is stored on the system in addition to data being sent. This is accomplished with an SD card module for nonvolatile storage. The amount of data being stored is relatively low, so an 8GB or 16GB SD card is sufficient for many months (the server also has the option to remotely erase the SD card). The SD card also allows data to be stored temporarily when out of coverage and transmitted as stale data once coverage resumes.

To alleviate sensor replacement work, two small custom boards were designed to branch the sensor ports away from the system and sensors. The first board, which is placed with the system, contains the necessary measurement resistors, headers for interfacing with the Arduino, and a CAT5 port. A CAT5 cable is then used to route the necessary connections (see installation) to the battery compartment of the bus. The second board, termed routing box, located in the battery compartment, splits the connections into each sensor port. This allows sensors to be replaced with relative ease, as they do not need to be routed through the cab of the bus.

12VDC power is down converted to 5VDC using a COTS USB car charger. These provide all the necessary buffering to handle the power surge and dip when the bus starts up. The power and ground terminals are soldered to wires with spade connectors to connect to the bus's power terminals. Ideally, the car charger would contain two ports, allowing the main system and the

SIM900 to be on different lines. The SIM900 input current during transmission in poor coverage can cause a brownout in the Arduino, forcing a system reset, if it is powered from the Arduino. This can be alleviated by splicing a secondary wire to the USB 5VDC feeding the Arduino if only a single port is available on the car charger.

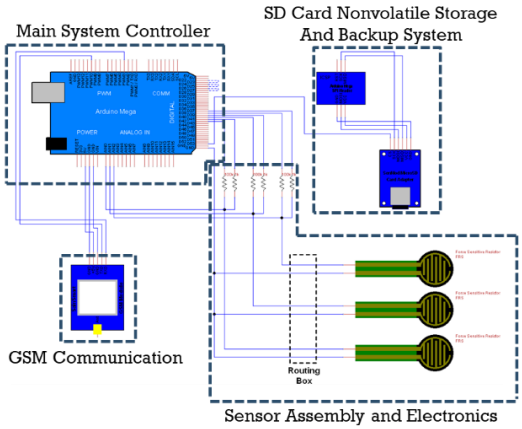


Figure 2-4. Wiring diagram of the system, excluding power connections.

Once the boards are wired together, the system is assembled in a small plastic box with holed cut to allow external wiring. The individual components are secured with silicone or hot glue except for the SIM900 module, which allows replacement of the SIM card if necessary. Additionally, the SD card module should be placed in such a way that it allows the SD card to be removed and replaced, as necessary. Finally, an inline fuse should be placed on the 12VDC power line coming into the system as it is not guaranteed that the bus power is fused.

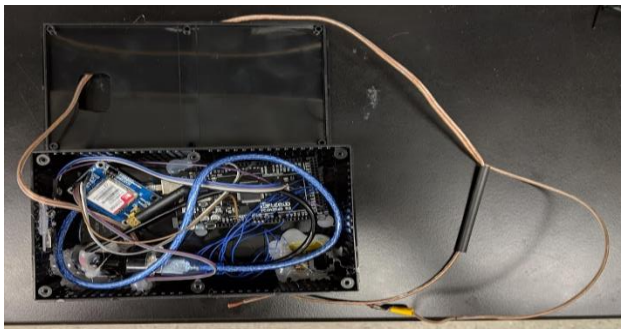


Figure 2-5. Assembled system secured in a plastic box, ready for installation.

2.4 Installation

The installed systems are placed in the electronics cabinet of the bus, usually located directly behind the driver. The power terminals are routed out of the system and attached to the keyed power and ground terminals using spade connectors. The system is then tested to ensure that data is being transmitted.

The CAT5 connection is then routed along the left side of the bus to the cabinet near the gas and brake pedals. These cabinets are accessed by removing the panels along the left side of the bus. The CAT5 connection is then routed into the battery compartment through a port used for power connections for the other bus electronics in this cabinet.

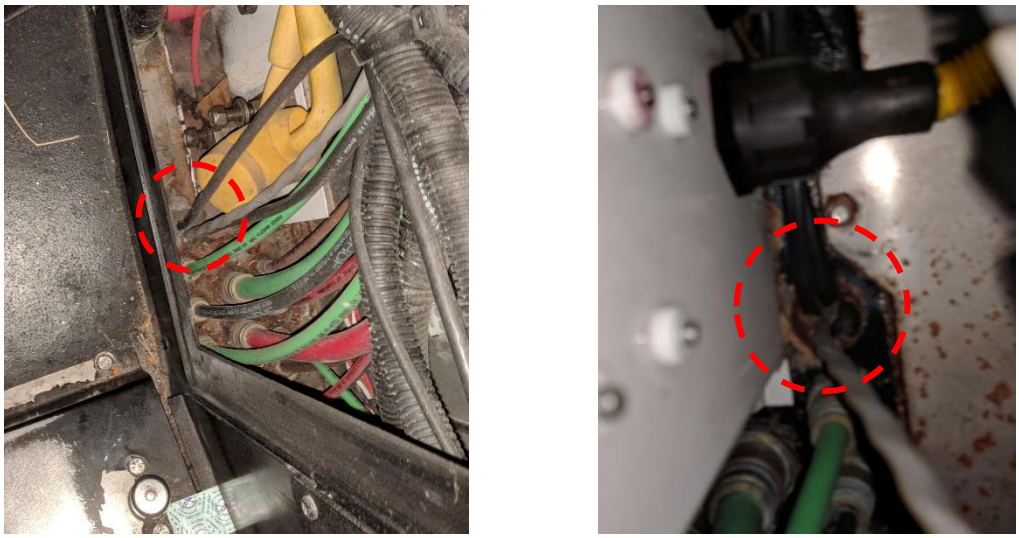


Figure 2-6. (Left) Side panel next to the driver's feet that contains the port to the battery compartment (outlined). (Right) View from the battery compartment of the same port (outlined).

Once in the battery compartment, the routing box is attached to the CAT5 connection. In some cases, the port is not sufficiently large or open to pass a terminated CAT5 cable. In this case, it is necessary to pass the unterminated CAT5 cable, then crimp the connector once it is passed.

The sensors are tested with a spare system directly prior to installation. Once functionality is verified, the sensors are secured to the front or back tire support of the rack using a combination of silicone and tie straps. It should be noted that the tie straps should not be over tightened and should only be placed at the very ends of the sensor assemblies to prevent them from triggering the sensor. The silicone adhesive attaches the sensor to the rack and the tie straps reduce the shear on the sensor assembly. To remove a sensor, first remove the tie straps, then use a shear motion to remove the sensor. Alternatively, using a sharp blade, cut through the silicone adhesive under the sensor.

Once attached, the sensor wire is routed along the bars of the bike rack, using tie straps to secure it at any bend and to keep it taut. This is extremely important as the bus wash will rip the sensor off if there is any slack in the wire. The wire is then routed either under or through the front bumper (depending on the bus model) to the front undercarriage, where it is secured using another tie strap. Care should be taken near the hinge for the bike rack: enough slack should be left so the bike rack can be folded up and down, but not enough slack should be left that the wire will catch when folding up. Once in the undercarriage, the wire is routed through a small port into the battery compartment, where it is plugged into the routing box.

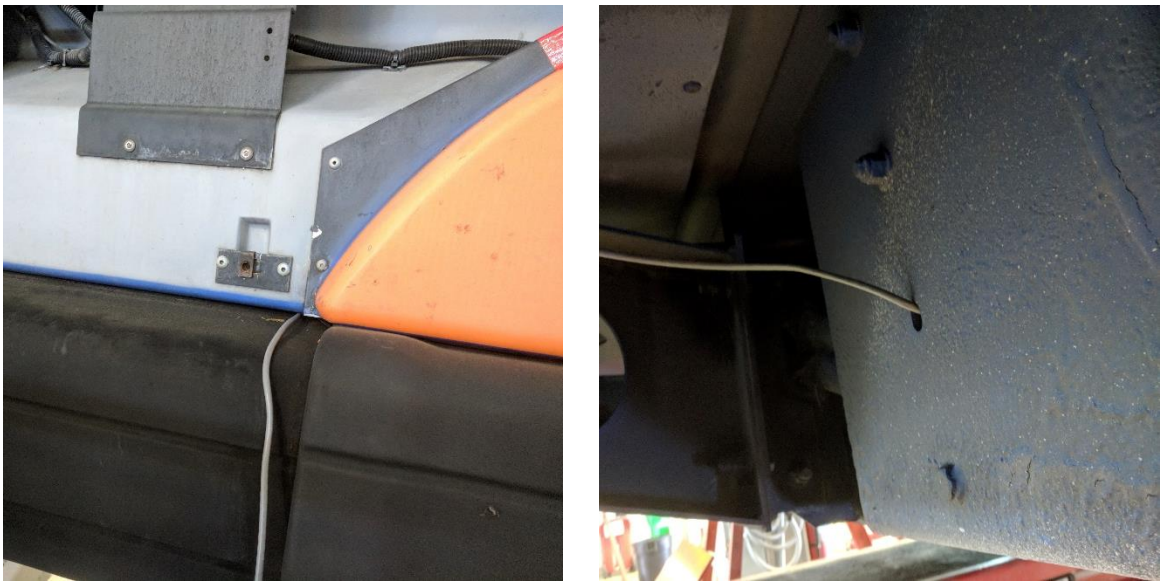


Figure 2-7. (Left) Sensor wire fed through the front bumper of the bus to the front undercarriage. (Right) Port under the front undercarriage where the sensor wires are fed to the battery compartment.

The system is then fully tested again by powering up the bus. Each sensor is tested, followed by every combination of sensors. The server is updated to map the system number to the installed bus and the number of slots is set. If the system is set to auto-threshold the installation is complete. Otherwise, the threshold is determined for each sensor and sent to the system through the server.

2.5 Bill of Materials

As mentioned above, to allow the responsible agency to either maintain or increase the number of systems, every part used was commercially available, except for the small custom boards used to branch out the sensor ports. More of these boards can be ordered with the Gerber files included. Below is a bill of materials for the systems.

Table 2-1. Bill of materials to build 20 systems

Part	**Cost**	**Number**	**Subtotal**
Arduino Mega	$15.00	20	$300.00
SIM900 GSM Module	$33.00	20	$660.00
MicroSD Card Module (5 pack)	$8.29	4	$33.16

SD Card	$6.00	20	$120.00
Ting SIM Card	$0.00	20	$0.00
USB Power Supply (2 pack)	$9.00	10	$90.00
Resistors	$10.00	1	$10.00
Ethernet Cable (500 ft)	$20.00	2	$40.00
Phone Cable (500 ft)	$10.00	3	$30.00
Force Sensitive Resistor	$4.70	60	$282.00
Solder	$20.00	1	$20.00
RJ11 Jacks (20 pack)	$8.19	3	$24.57
RJ 45 Jacks (39 pack)	$12.28	2	$24.56
Routing Box Board	$110.00	1	$110.00
System Board			
Sensor Board			
Silicone	$5.92	9	$53.28
8ft Aluminum Bar	$13.98	15	$209.70
Wire Connections	$7.68	4	$30.72
Packaging	$20.00	20	$400.00
Total Cost:			**$2,437.99**
Cost Per Bus:			**$121.90**

In addition to the building cost, each system requires a cellular plan to communicate on the 2G network. Given the low data rate and quantity of systems, Ting was chosen to be the provider as they charge a small fee per line and group the data (approximately 20MB per month for 20 buses). The following table includes the costs associated with the cellular plan with Ting.

Table 2-2. Cost of the cellular plan for 20 systems

Ting Service	**Cost (Monthly)**	**Number**	**Subtotal (monthly)**
SIM Activation	$6.00	20	$120.00
Data	$10.00	1	$10.00
Taxes	$1.22	20	$24.40
Total Monthly Cost:			**$154.40**
Total Monthly Cost Per Bus:			**$7.72**

Part B: BIKERIDE MOBILE APPLICATION

3 BikeRide Mobile Application

3.1 Introduction

RTS buses are one of the most used transportation mode in the City of Gainesville, and the bike racks provide convenience for those passengers who are taking a bus with their bike. Because the number of bike rack slots is limited, it is good for those passengers to know the bike rack availability before they take a bus. BikeRide is a mobile application used for checking real-time bike rack availability on RTS buses, built using Flutter to support both iOS and Android platforms developed by the UF team. With the app, the users can know if a bus they plan to take has capacity to hold a bike.

3.2 Requirements

3.2.1 User Requirements

This table is for what users need and require for BikeRide Mobile Application.

Table 3-1. User requirements

Req. ID	User Requirements
3.2.1.1	The app shall provide a user's current location
3.2.1.2	The app shall allow the user to search buses by route number
3.2.1.3	The app shall display bike rack availability of each running bus of a certain route
3.2.1.4	The app shall be able to locate a bus
3.2.1.5	The app shall allow the user to browse the map

3.2.2 Functional Requirements

This table is for what BikeRide Mobile Application must meet the requirements for functions.

Table 3-2. Functional requirements

Req. ID	Functional Requirements
3.2.2.1	The app shall provide a user's current location after the user clicking the locating button
3.2.2.2	A list of bus information shall be provided after a valid user input of route number(s), including the bike rack availability of the bus
3.2.2.3	The bike rack availability shall include the status of each slots (empty/occupied)
3.2.2.4	The app shall provide a bus's current location after the user clicking the bus information in the searching results
3.2.2.5	The map shall be displayed as the background of the app

3.2.3 Non-Functional Requirements

This table is for basic requirements which BikeRide Mobile Application must operate.

Table 3-3. Non-functional requirements

Req. ID	Non-Functional Requirements
3.2.3.1	Performance: finish loading app within 5 seconds
3.2.3.2	Scalability: be able to handle data for all the running buses

3.3 Wireframe

The wireframe provides an overview of the layout, user flow and functionality of the app.

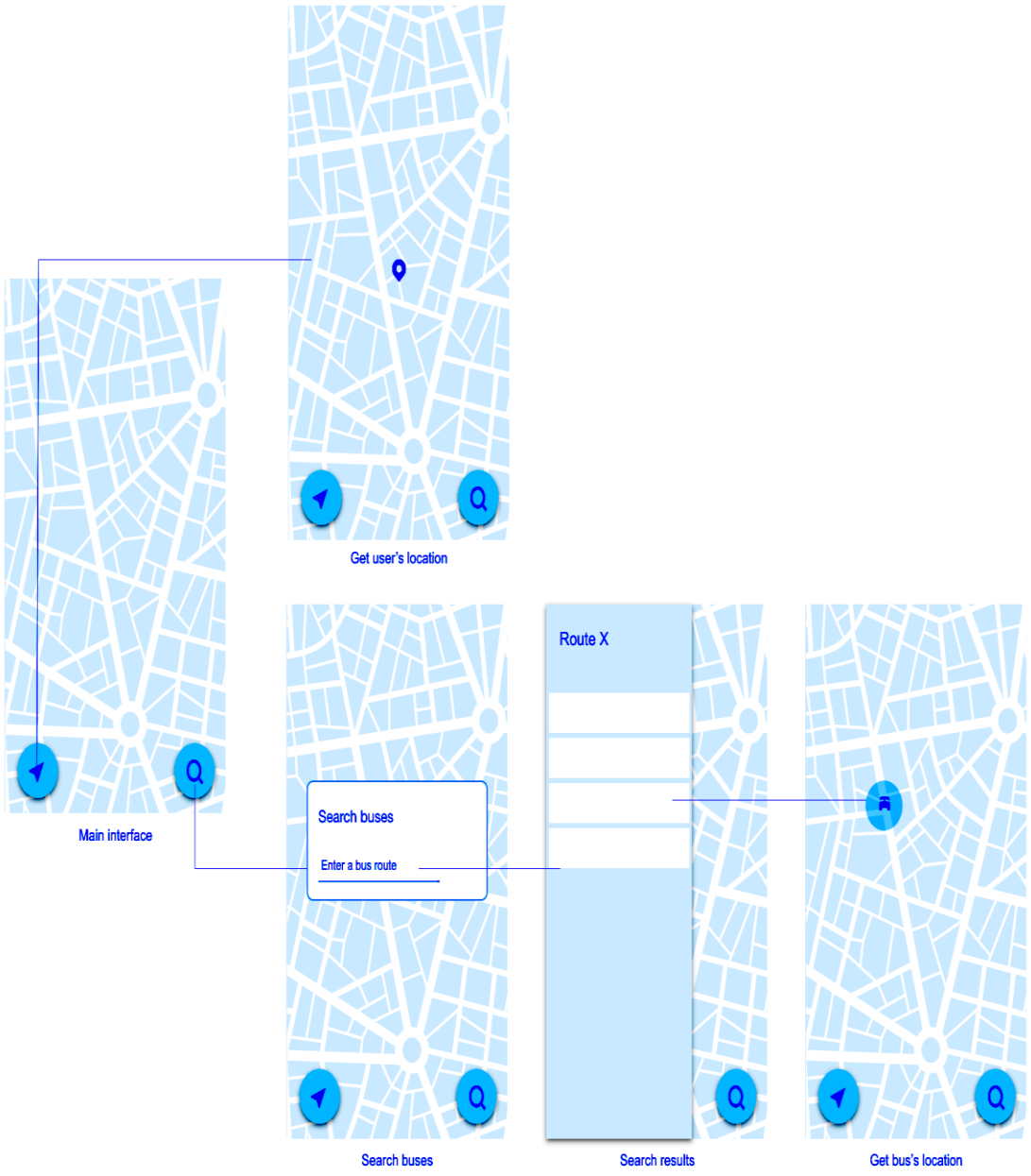


Figure 3-1. Wireframe. Main interface(left), user's location(top), function of searching buses(bottom-left), search result(bottom-middle), bus's location(bottom-right)

3.4 User Interface & Features

3.4.1. Main page

A map of Gainesville is shown, with UF campus as the center by default.

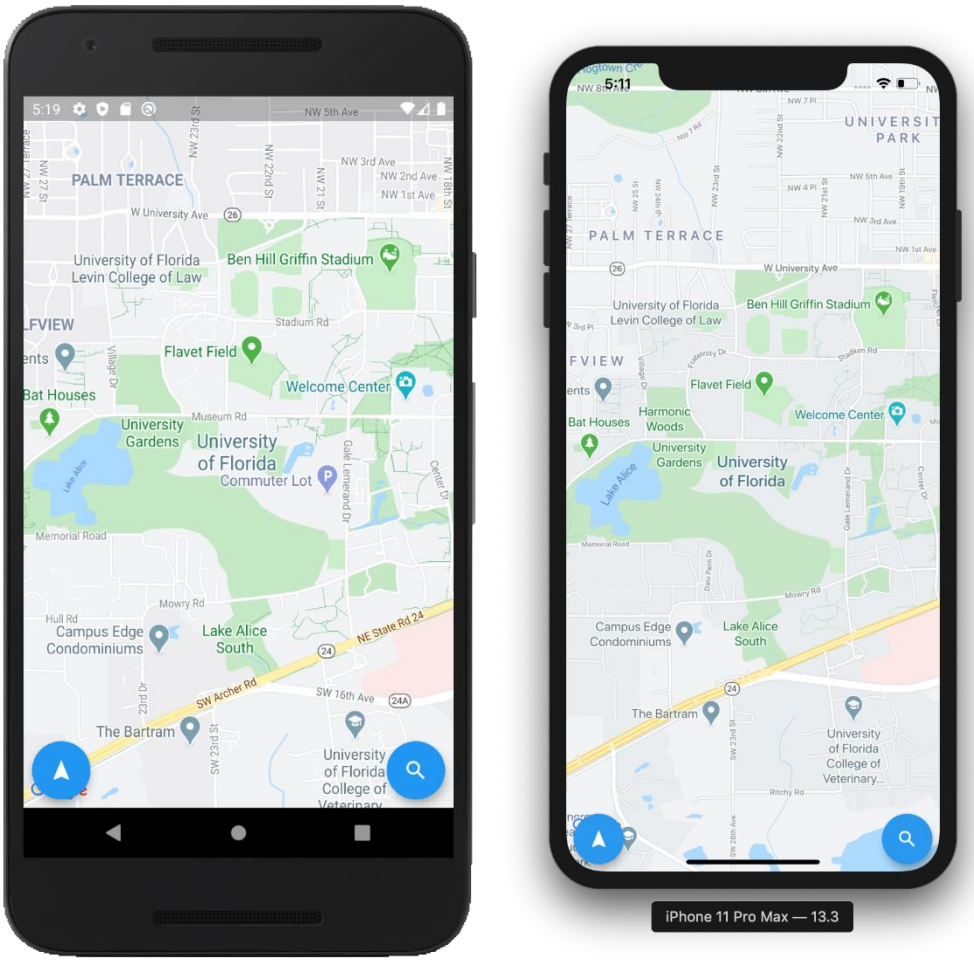


Figure 3-2. User Interface and Feature

3.4.2. Get a user's current location

Tap the button on the bottom-left to get the user's current location.

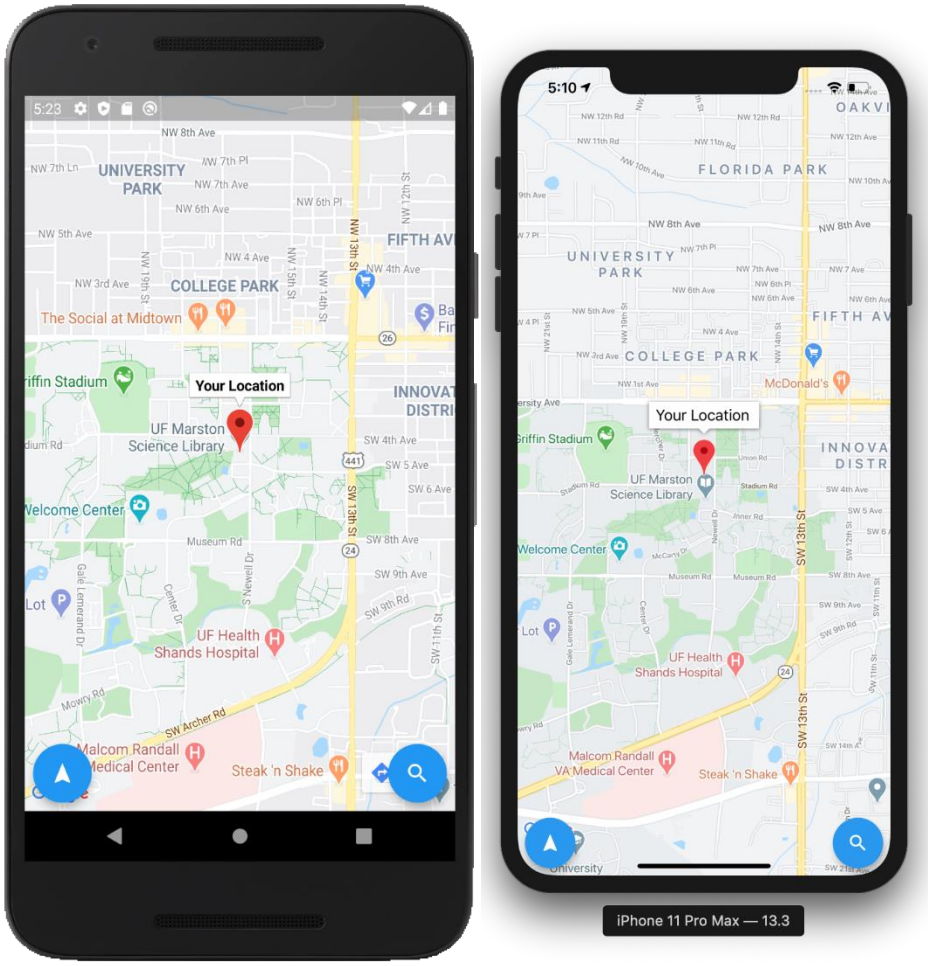


Figure 3-3. User's current location

3.4.3 Search buses by route number

Tap the button on the bottom-right to search buses by route number. Enter a bus route number in the search bar to check all the running buses with sensors of the certain route. The availability of each slot of the bus bike rack is represented by green and red signs, meaning empty and occupied, respectively. Searching multiple routes are also available.

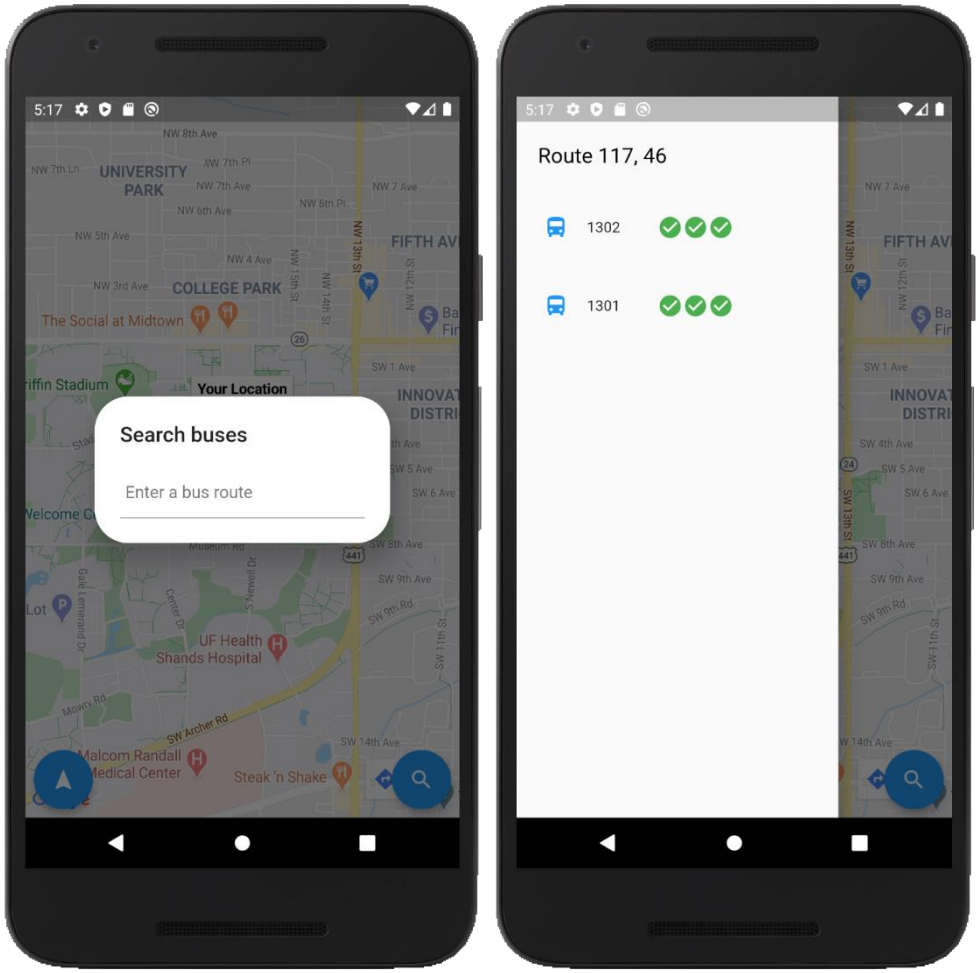


Figure 3-4. Search buses by route number(left), the result of available 3 slots with green signs(right)

3.4.4. Get a bus's current location

Tap a bus shown in the result list to locate it on the map.

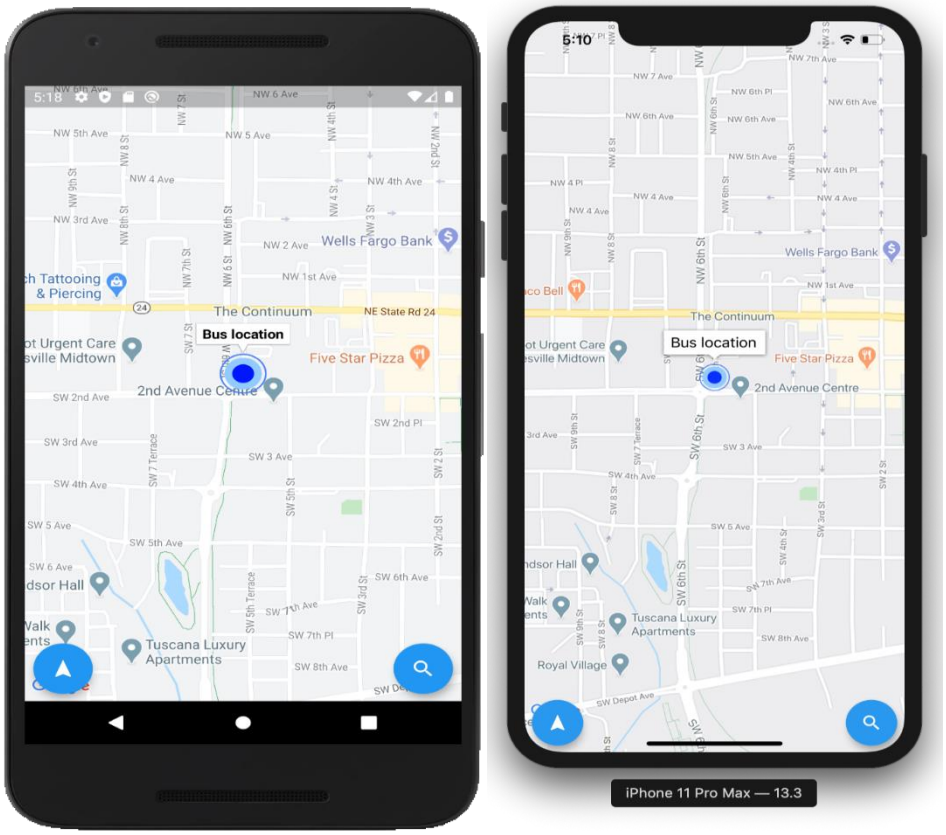


Figure 3-5. Get bus's current location

3.4.5 Data source

Table 3-4. Data source in server

Bus ID	Route Number	System ID
1302	117	2
573	35	4
2572	1	9

568	38	12
2574	2	13
1301	46	21

3.4.6 API Documentation

Front-end

`void _setCustomMapPin()`
//Sets a custom map pin for a bus.

`void _getUserLocation()`
//Gets the user's current location and shows it on the map.

`void _getBusLocation(LagLng position)`
//Gets the bus's location and shows it on the map.
Parameters:
`position` // - the location in the form of latitude and longitude

`void _setBusDataList(String input)`
//Sets the global list of BusData that matches the input string and builds the drawer.
Parameters:
`input` // - the String to search for

`Future _displaySearchDialog(BuildContext context)`
// Displays the dialog for searching buses.
Parameters:
`context` // - the context that contains the information about the location in the tree at which
// this AlertDialog is being built.
Returns:
AlertDialog of searching buses

`Future _displaySubscribeDialog(BuildContext context)`
// Displays the dialog for subscribing to a bus.
Parameters:

`context` // - the context that contains the information about the location in the tree at which
// this AlertDialog is being built.
Returns:
// AlertDialog of subscribing

`List<Widget> _getDrawerList(String input)`
// Creates a bus list of the input route.
Parameters:
`input` // - the input route
Returns:
// the list of widgets including the title and the bus information

`List<GestureDetector> _getBusInfoItems(List<BusData> list)`
// Creates a list of the bus information.
Parameters:
`list` // - the list of BusData
Returns:
// the list of GestureDetectors

`Row _getSlotsStatus(String input, int numSlots, int slotsFilled)`
// Displays the slot status with red for an occupied slot and green for an empty slot.
Parameters:
`input` // - the input route
`numSlots` // - the total number of slots of a bike rack
`slotsFilled` // - the number of occupied slots
Returns:
// the row of the slot status

Back-end Interface

`Future<Position> getAsyncCurrentPos()`
// Returns the current position of the user.
Parameters:
none
Returns:
// the current geolocation of the user packaged as a Position structure

`Future<List<BusData>> getBusesByStr(String str)`
//Creates a list of the bus information by performing a string search on all available data.
Parameters:
`str` // – the user input
Returns:
// the list of BusData with matching fields

Back-end

```
Set<String> parseStringForBusID(String str)
```
// Parses the string for numbers and returns a set with searchable strings.
Parameters:
`str` // – the input string
Returns:
// a set of searchable strings

```
Future<List<BusData>> getJSONs() async
```
// Creates a list of all available bus information by parsing JSON data received via http.
Parameters:
// none
Returns:
// the list of all available BusData

```
List<BusData> parseData(Set<String> str)
```
// Creates a list of bus information from the larger set by finding data with fields containing the
// search strings.
Parameters:
`str` // – the user input
Returns:
// the list of BusData with matching fields

```
Future<Position> getCurrentPosition() async
```
// Gets the current position of the user.
Parameters:
none
Returns:
// the current geolocation of the user packaged as a Position structure

```
Future<DateTime>refreshData() async
```
// Refreshes the data stored in the instance of the backend with webdata.
Parameters:
none
Returns:
// the time of the async response

PART C: USAGE ANALYSIS of BUS BIKE RACK

4 Usage Analysis of Bus Bike Rack

4.1 Introduction

Bus Bike Rack usage analysis is performed using the data acquired from the installed sensing systems of the 18 selected buses in the City of Gainesville Regional Transit System (RTS) between May 1, 2018 and April 30, 2019 (12 months). Twelve buses are equipped with 3 slot racks and the other six with 2 slots racks (total 18 buses).

The bus number and system ID of the 18 buses checked on September 18, 2018 as shown in Figure 4-1.

Bus Number	System ID	Last Checked In	Last Data	Firmware Version
1200	22	Thu, 20 Sep 2018 18:40:39 -0400	Thu, 20 Sep 2018 18:40:42 -0400	2.01
unmapped	24	Tue, 24 Jul 2018 19:51:23 -0400	Tue, 24 Jul 2018 20:01:09 -0400	2.01
1302	2	Thu, 20 Sep 2018 09:07:53 -0400	Thu, 20 Sep 2018 09:37:59 -0400	2.01
1205	23	Mon, 02 Jul 2018 06:46:57 -0400	Mon, 02 Jul 2018 05:53:51 -0400	2.01
unmapped	17	Tue, 24 Jul 2018 19:05:12 -0400	Tue, 24 Jul 2018 19:06:15 -0400	2.01
1301	21	Thu, 20 Sep 2018 07:40:13 -0400	Thu, 20 Sep 2018 18:02:37 -0400	2.01
553	16	Thu, 20 Sep 2018 06:41:07 -0400	Thu, 20 Sep 2018 17:03:12 -0400	2.01
2572	9	Thu, 20 Sep 2018 19:07:49 -0400	Thu, 20 Sep 2018 19:17:53 -0400	2.01
2582	15	Fri, 07 Sep 2018 21:39:31 -0400	Fri, 07 Sep 2018 21:39:33 -0400	2.01
548	11	Mon, 13 Aug 2018 16:39:20 -0400	Mon, 13 Aug 2018 16:39:23 -0400	2.01
1300	7	Tue, 07 Aug 2018 17:53:37 -0400	Tue, 07 Aug 2018 18:03:40 -0400	2.01
568	12	Thu, 20 Sep 2018 06:22:28 -0400	Thu, 20 Sep 2018 21:24:39 -0400	2.01
2583	10	Thu, 20 Sep 2018 18:55:50 -0400	Thu, 20 Sep 2018 21:26:17 -0400	2.01
107	14	Wed, 08 Aug 2018 18:48:17 -0400	Wed, 08 Aug 2018 18:48:28 -0400	2.01
2574	13	Thu, 20 Sep 2018 06:35:14 -0400	Thu, 20 Sep 2018 21:28:25 -0400	2.01
549	6	Thu, 20 Sep 2018 07:23:17 -0400	Thu, 20 Sep 2018 18:15:10 -0400	2.01
573	4	Thu, 20 Sep 2018 21:25:08 -0400	Thu, 20 Sep 2018 21:25:12 -0400	1.00
1201	3	Never	Never	
1202	8	Thu, 20 Sep 2018 21:00:11 -0400	Thu, 20 Sep 2018 21:10:16 -0400	1.00
2575	5	Mon, 13 Aug 2018 16:28:14 -0400	Mon, 13 Aug 2018 16:38:18 -0400	2.01
unmapped	255	Tue, 08 May 2018 00:28:59 -0400	Tue, 08 May 2018 00:29:02 -0400	2.01

Figure 4-1. List of the bus number and system ID of the 18 buses used for the data analysis

4.2 Overall Usage Analysis

Both the total number of hours and users of bike rack usage during the study period (May 1, 2018 – April 30, 2019) are analyzed daily, weekly, monthly, and seasonally. In the plots, the blue

histogram shows the total number of hours of bike rack usage reading in the left y-axis and the orange line shows the total number of users reading in the right y-axis.

4.2.1 Daily usage analysis

Figure 1 shows the total usage data of the bus bike rack sensors plotted in a 24-hour daily span during the entire study period. Starting as early as 4 am, the usage increases and reaches a peak at around 8 am in the morning. The usage is sustained till 4 pm and starts to decrease. After 8 pm, the usage is rare. Note the first class of University of Florida starts 7:25 am and normal administration offices starts 8 am and ends 5 pm. The bike rack usage trend is well matched with the usual working hour schedule. It shows that the sensing system operates well as expected.

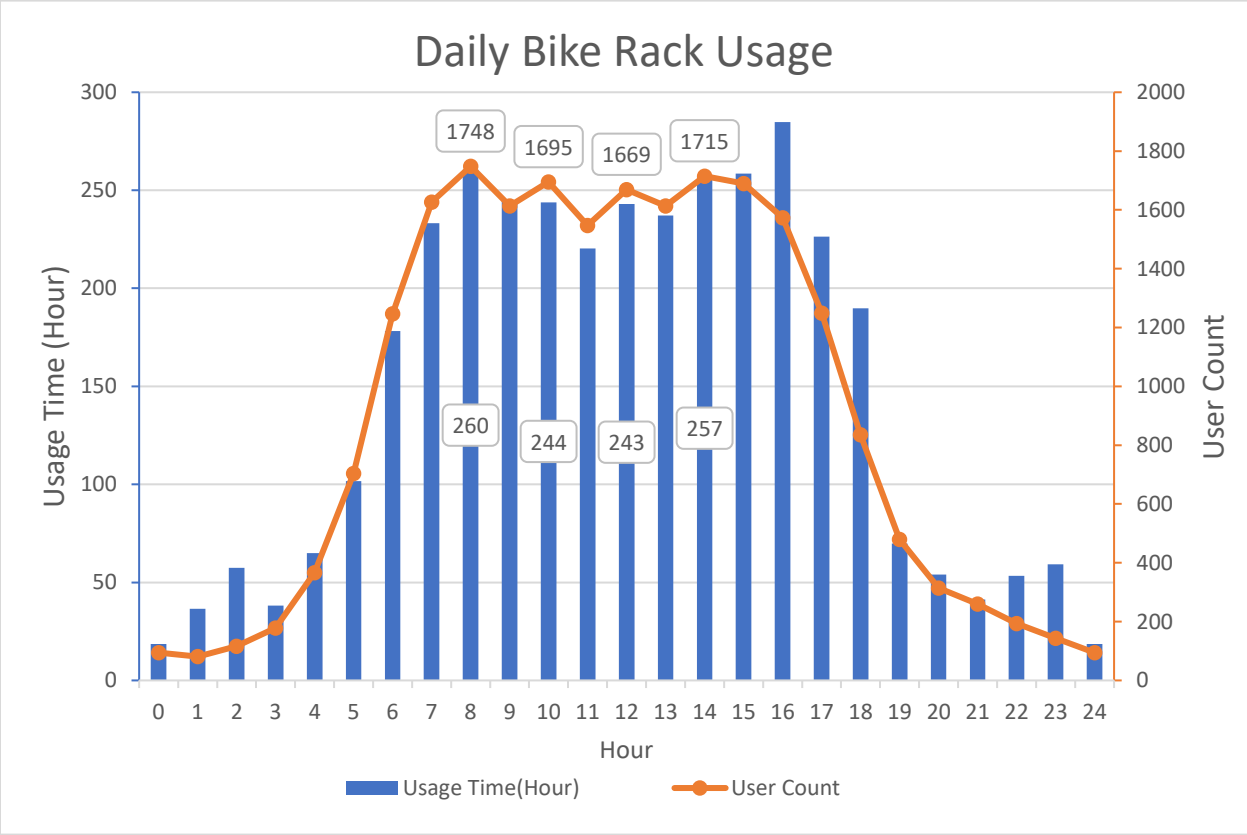


Figure 4-2. Daily bike rack usage

4.2.2 Weekly usage analysis

Figure 2 shows the usage data of the bus bike rack sensors plotted in a 7-day weekly span. During the weekdays between Monday and Friday, the usage is high compared with the weekend days (Saturday and Sunday). We observed that the usage on Monday and Tuesday is marginally higher than that on Wednesday, Thursday, and Friday. Please note that the bus schedules on Saturday and Sunday are the same while the usage of Saturday is much higher than that of Sunday. This trend might be related to Saturday events. This will be revisited later.

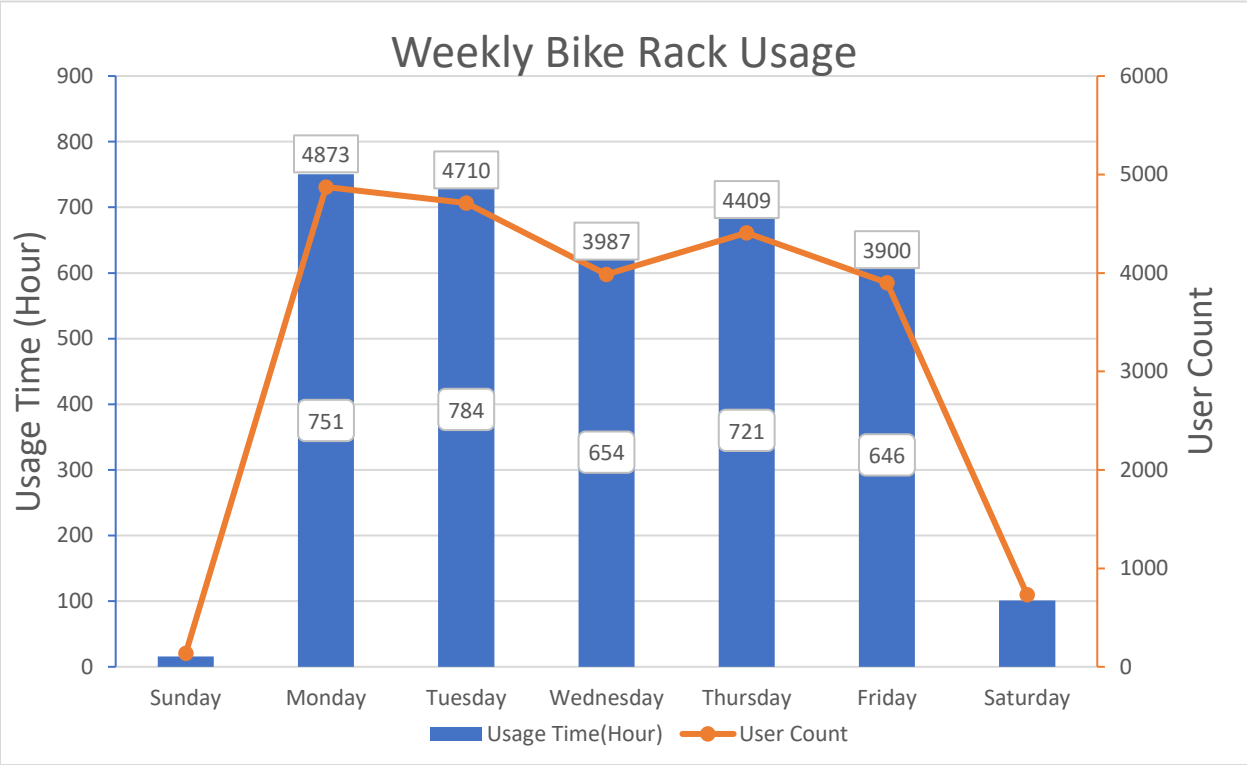


Figure 4-3. Weekly bike rack usage

4.2.3 Monthly usage analysis

Figure 3 shows the usage data of the bus bike rack sensors plotted in a monthly span between May 2018 and April 2019. It shows steady usage data all year around except February and March 2019, where the number is lower than other months. This is attributed to the reduced usage during the Spring break (March 2 – 10, 2019) and the major hardware and software upgrade by the research team during February and March, 2019. In April 2019, the usage is back up again.

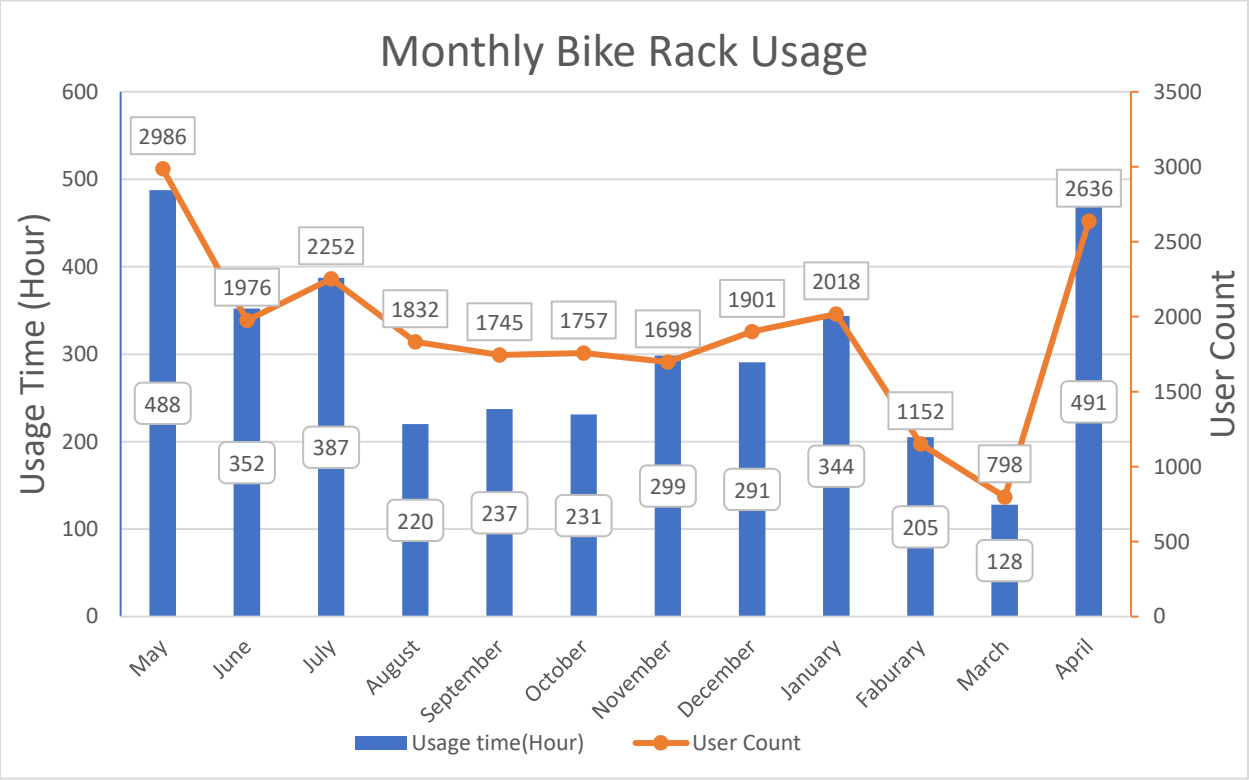


Figure 4-4. Monthly bike rack usage

4.2.4 Seasonal usage analysis

Figure 4-5 shows the usage data of the bus bike rack sensors plotted in a seasonal span. Summer 2018 represents the data summation between May and August 2018. Fall 2018 shows September to December 2018, and Spring 2019 shows January to April 2019. Interestingly, the Summer 2018 usage is approximately 25 to 30 % higher than that of Fall 2018 and Spring 2019. The Summer 2018 usage may be attributed to longer daylight and the reduced number of buses available during Summer semester, which encourages or forces people to use the multimodal (bus and bike) commuting approach. Spring 2019 usage is slightly lower than that of Fall 2018, which is in part due to the reduced usage during the system upgrade in February and March 2019.

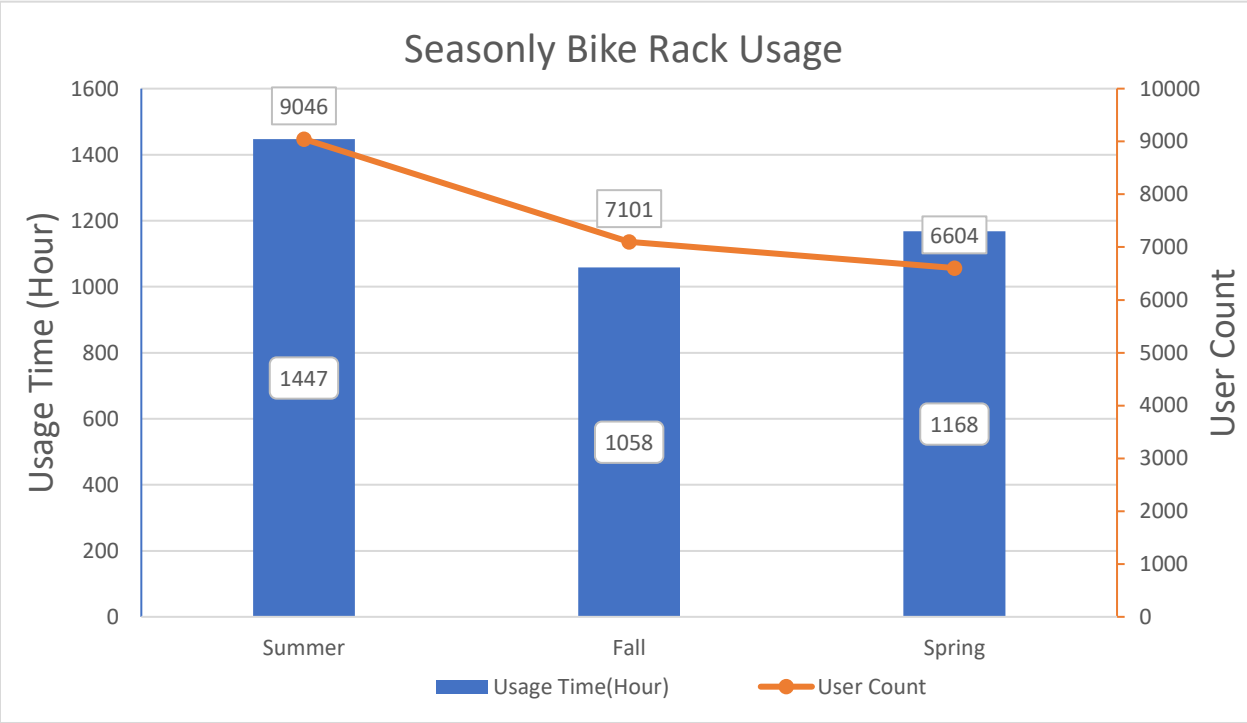


Figure 4-5. Seasonal bike rack usage

4.2.5 Saturday usage analysis

Figure 4-6 shows the usage data of the bus bike rack sensors on Saturdays in a seasonal span. The usage of Fall 2018 is higher than other seasons. The high usage in Fall may be attributed to the home football games when more people gather in town, and the usage of the bike rack increases. Note that seven home games occurred during Fall 2018 as shown in Figure 4-7.

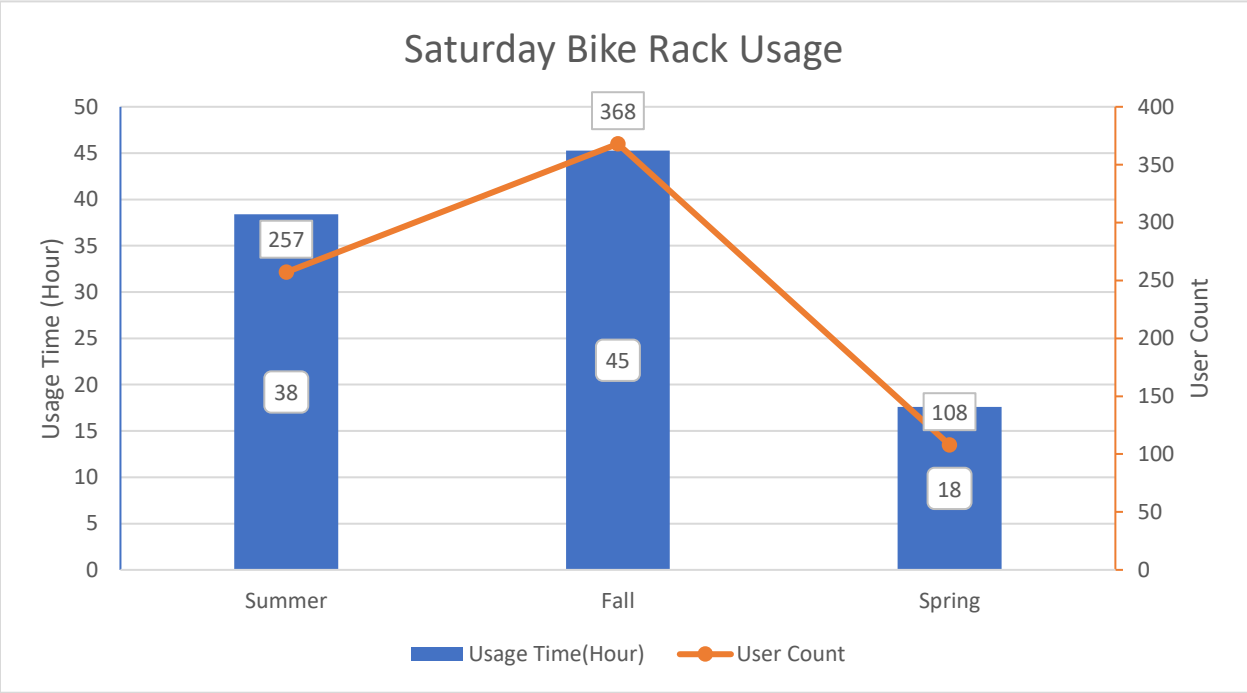


Figure 4-6. Saturday bike rack usage

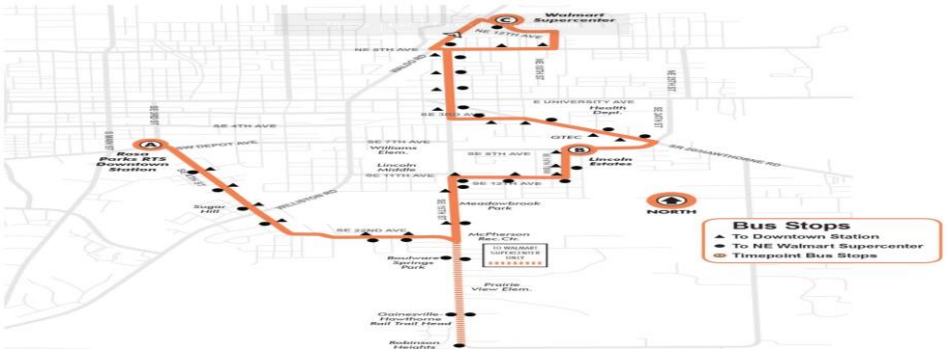
Figure 4-7. Football home game schedule in Fall 2018

4.3 Individual Bus Route-Based Usage Analysis

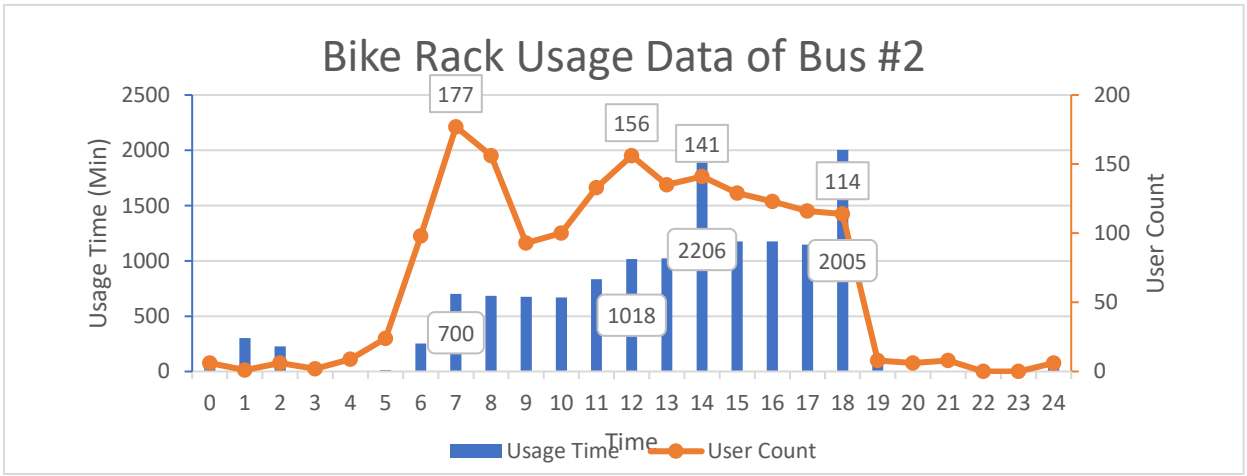
In this analysis, the bike rack usage based on selected bus routes are shown. The bus numbers are Bus 2, Bus 4, Bus 8, Bus 16, Bus 13, Bus 22. To help facilitate the analysis, the map of each bus route is shown together with the usage data.

4.3.1 Bus #2

Bus #2 shuttles between the Rosa Parks RTS Downtown Station and Walmart Supercenter located in Waldo Rd and NE 12th Ave, which covers eastern Gainesville as shown in Figure 6a. The data is one collected for the entire study period. The usage trend follows the typical working hour pattern, i.e. high usage between 7 am and 6pm as shown in Figure 6b. Note that the middle of the bus route leads to the Gainesville-Hawthorne Trail, which offers a nice bike trail reaching the UF campus and Shands Hospital. As this route is approximately 5 miles away from the campus, the usage time starts an hour earlier and ends an hour later than the daily usage time shown in Figure 1.



(a)

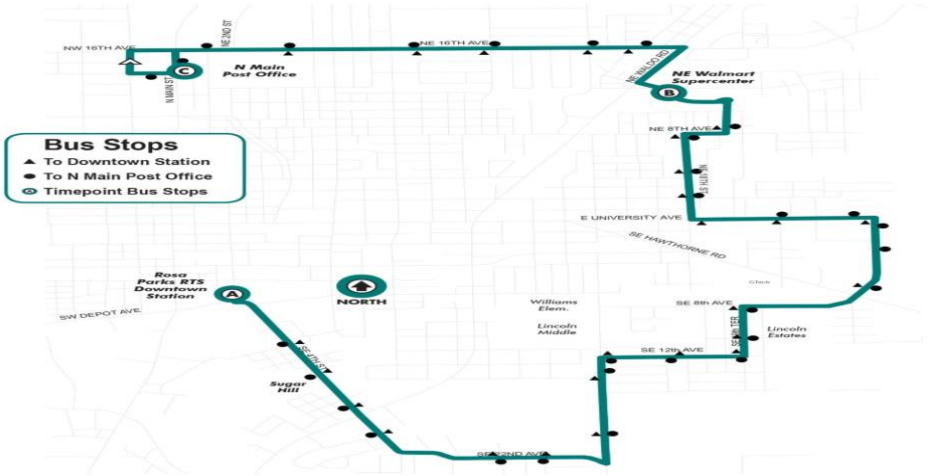


(b)

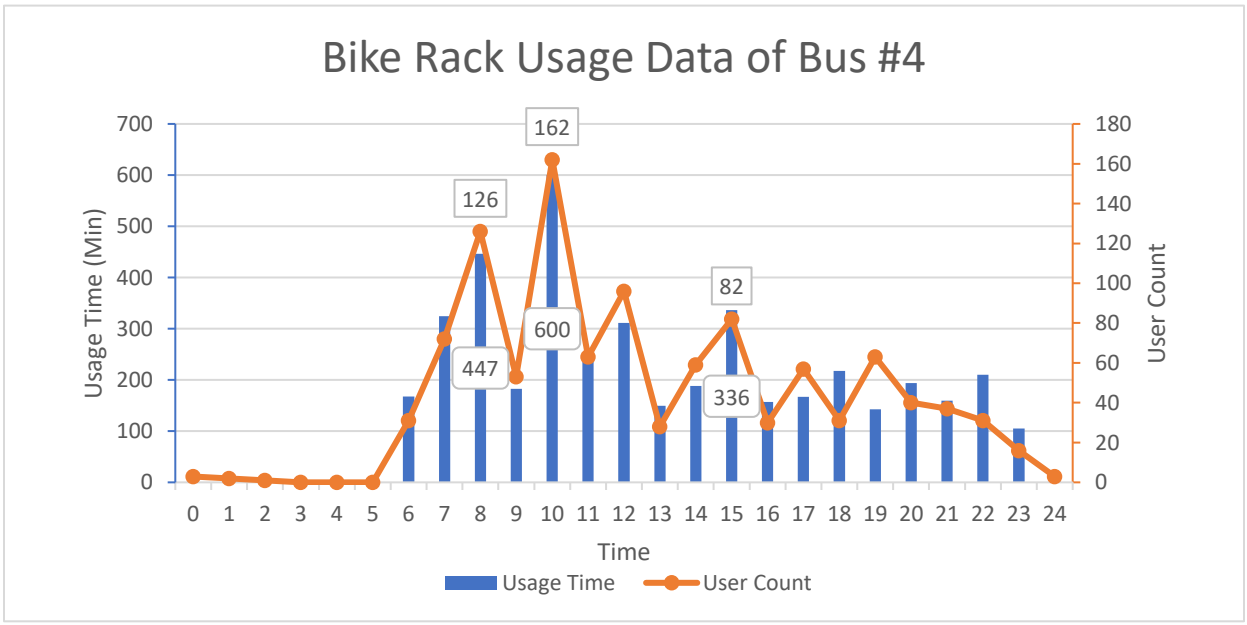
Figure 4-7. The map (a) and the bike rack usage data (b) of Bus #2

4.3.2 Bus #4

Bus #4 shuttles between the Rosa Parks RTS Downtown Station and the Station located in N. Main and NE 16th Ave, which covers eastern Gainesville as shown in Figure 7a. Also, a large portion of the route overlaps with that of Bus #2 e.g. between the Rosa Parks RTS Downtown Station and Walmart Supercenter located in Waldo Rd and NE 12th Ave. The overall usage is lower than that of Bus #2. It shows periodicity in usage. For instance, the usage is high at 8 am, 10 am, and noon while the usage of 9 am, 11 am, and 1 pm is low. The trend could be related to the frequency of the bus operation.



(a)



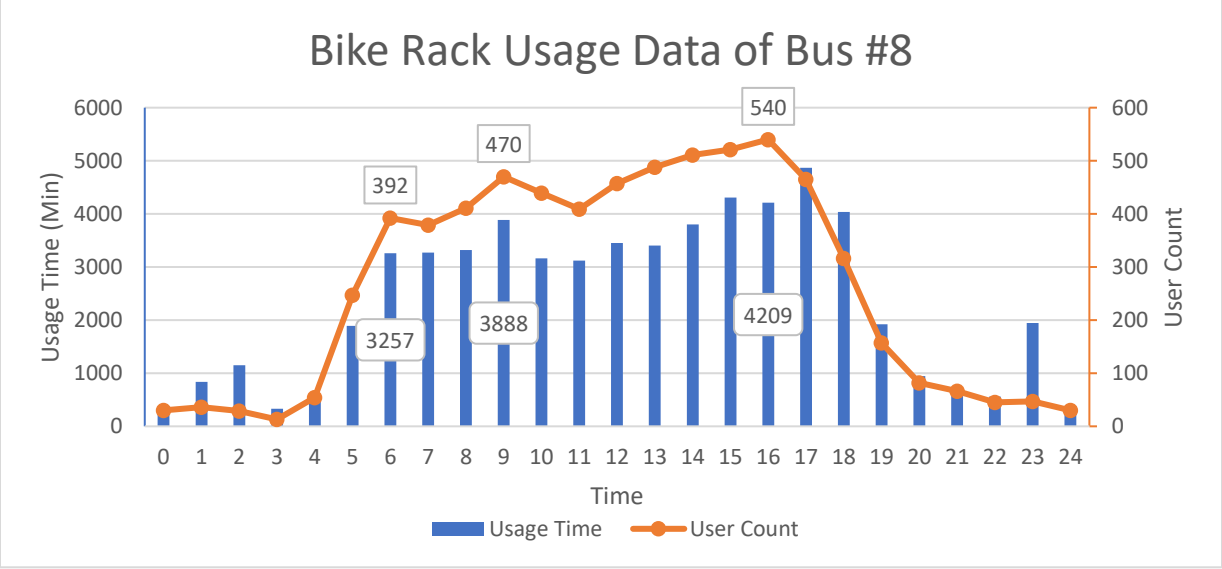
(b)

Figure 4-8. The map (a) and the bike rack usage data (b) of Bus #4

4.3.3 Bus #8

Bus #8 shuttles between the Reitz Union Station (Center of UF Campus) and the Station near Senior Recreation Center located near in SR 121 and NW 53rd Ave, which covers northern Gainesville as shown in Figure 8a. The bike rack usage is significantly higher than the previous two bus routes, e.g. four times more usage than that of Bus #2 as shown in Figure 8b. One end of bus stops is the Reitz Union station, which is the center of the campus; therefore, the route is very popular for commuting students, staff, and faculty, and the bike rack usage is very high. Also, the route passes Shands hospital, which would include multi-modal commuters working for the hospital.

(a)

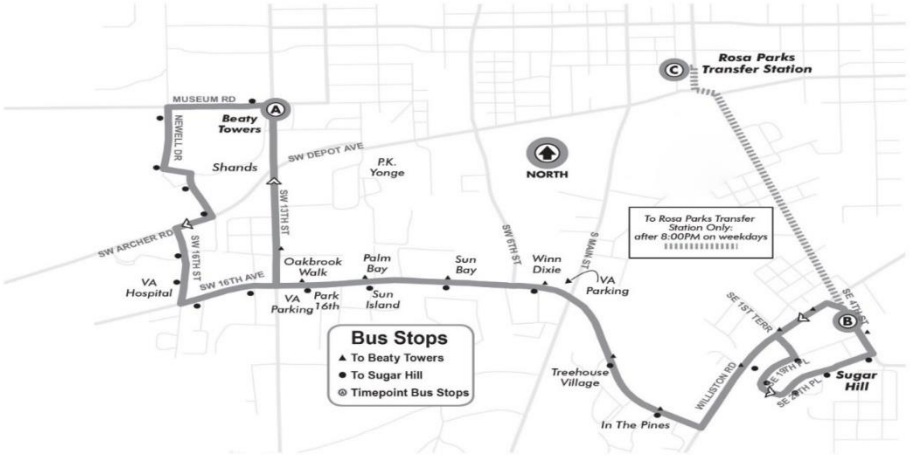


(b)

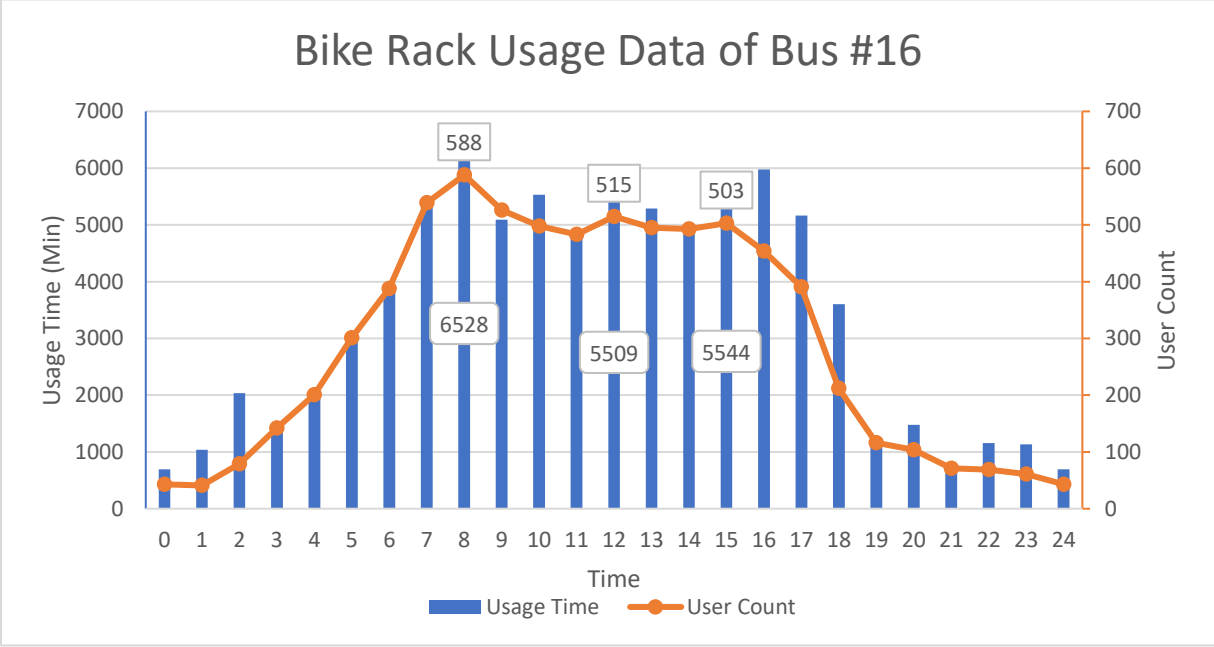
Figure 4-9. The map (a) and the bike rack usage data (b) of Bus #8

4.3.4 Bus #16

Bus #16 shuttles between Shands Hospital Station and the Station near Williston Rd and SE 4th St, which covers southeastern Gainesville as shown in Figure 9a. This route is also very popular. During the daytime usage is steady high as shown in Figure 9b. This indicates that multi-modal commuters between the campus/Shands hospital and the southeastern Gainesville actively use the bus bike racks.



(a)

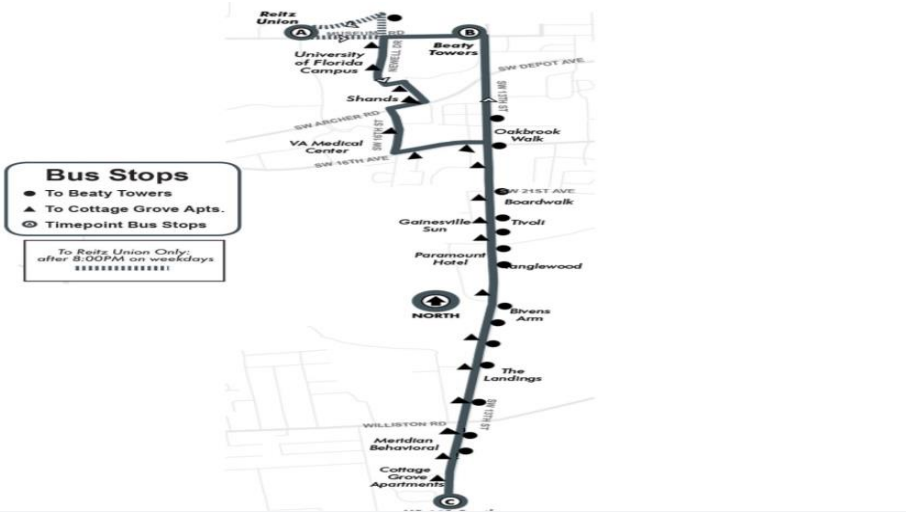


(b)

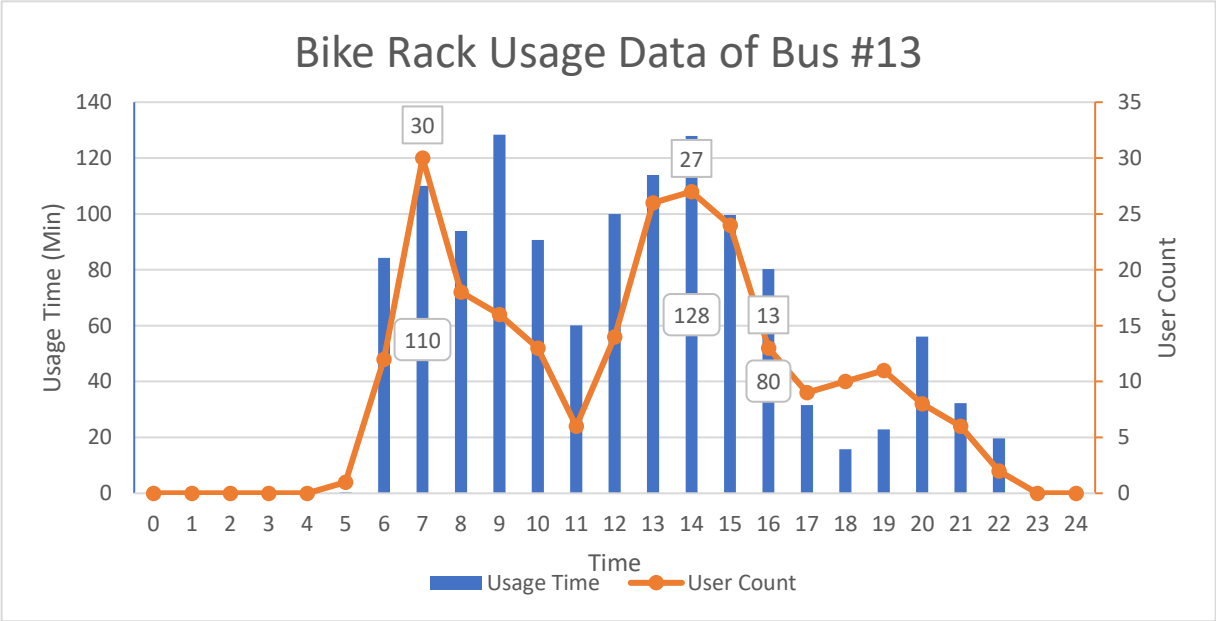
Figure 4-10. The map (a) and the bike rack usage data (b) of Bus #16

4.3.5 Bus #13

Bus #13 shuttles between the Reitz Union Station and the Station near the Cottage Grove Apartment in SW 13th St, which covers southwestern Gainesville as shown in Figure 10a. Interestingly, while this route departs from the Reitz Union, the usage of the bike rack is a lot lower compared with other bus routes including Reitz Union as shown in Figure 10b. One speculation is that the distance is relatively close (e.g. less than 3 miles) for multi-modal commuting. This may need further investigation to rule out sensor malfunction.



(a)



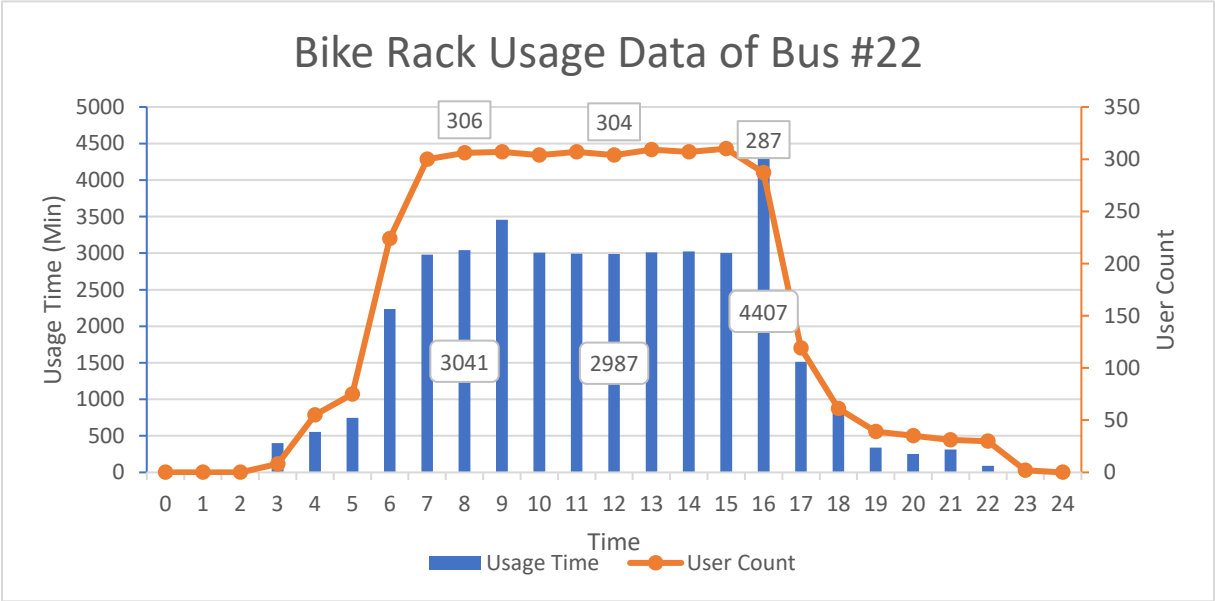
(b)

Figure 4-11. The map (a) and the bike rack usage data (b) of Bus #13

4.3.6 Bus #22

Bus #22 shuttles between the Cultural Plaza Station and the Station near NW 7th Ave and NW 12th St passing through the UF campus as shown in Figure 11a. The usage plot shows very steady and stable usage during daytime as shown in Figure 11b. Since the route directly passes through the campus, many students may use the bike racks between classes or for commuting.

(a)



(b)

Figure 4-12. The map (a) and the bike rack usage data (b) of Bus #22

4.4 Discussion and Conclusion

The bus bike usage is mainly distributed during the daytime (7 am – 6 pm) between rush hours, which is well matched with the class hours and working hours of the University of Florida and Shands Hospital and is thought to be reasonable. The weekday usage is higher than weekend usage, which shows that the main purpose of the bike usage is for commuting. Meanwhile, Saturday usage during fall semester is higher than other seasons, which is attributed to the football games during fall. In general, multimodal commuting is higher during summer than fall or spring. The reduced bus operations during summer might motivate more multimodal commuting. There are significant usage differences observed among different bus routes. Most routes starting from or passing through the UF campus and Shands hospitals show the highest usage of bike rack and multimodal commuting.

Due to reduced RTS service during summer, buses used during that season may benefit from having three slot bike racks to accommodate increased bike rack demand. Buses departing from and ending at the UF campus and Shands hospital need three-slot bike racks. For some bus routes far from campus, two-slot bike racks may be sufficient.

Part D: CURRENT STATE and FINAL RECOMMENDATIONS

5 Current State of the Project

5.1 General Architecture

The complete BikeRide system consists of two main components: a mobile client application and a web API server.

5.2 Mobile Application

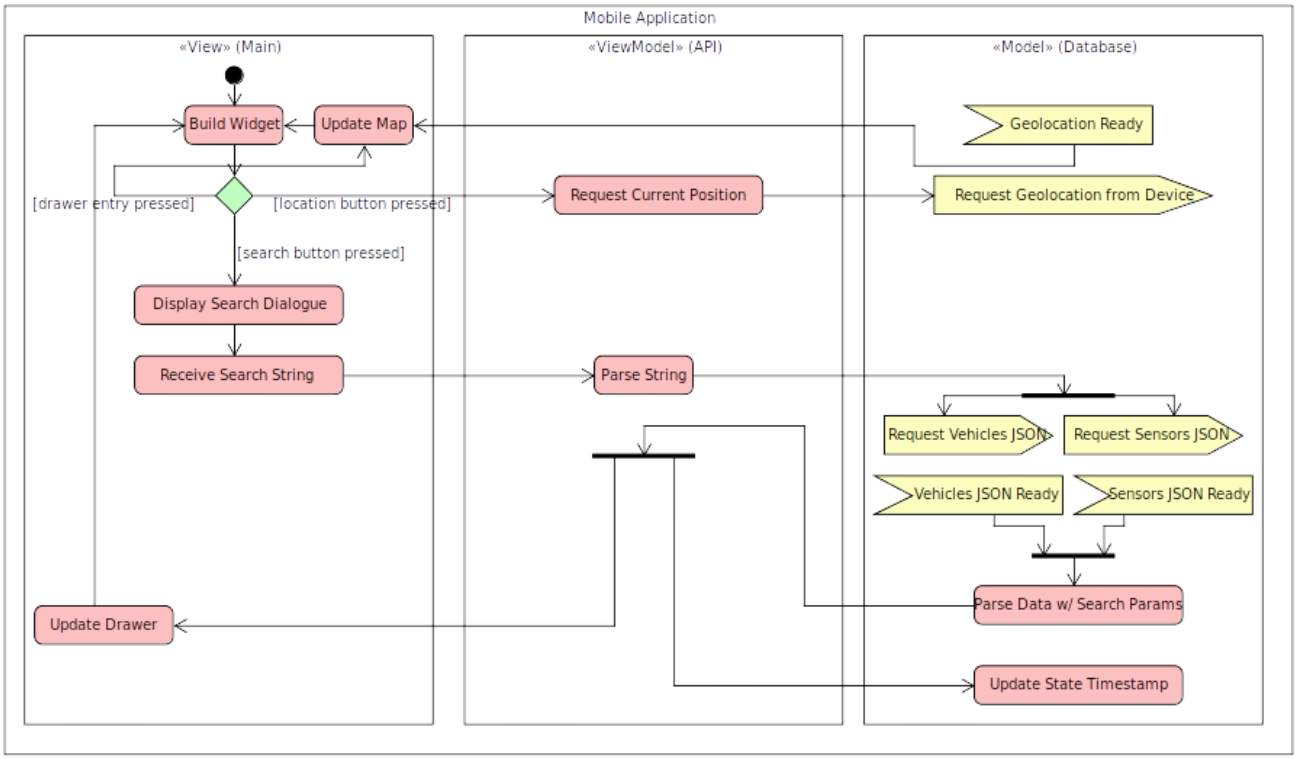


Figure 5-1. UML activity diagram for BikeRide mobile application

The mobile application was designed using the Model-View-ViewModel (MVVM) design principle, in which the frontend implementation of the application is isolated from the backend implementation of the application via a delegator interface. This separation of implementation is intended to improve long-term maintenance of the application, as backend changes to existing features can be made to the data model easily without making any corresponding changes to the View. The addition of new UI features is also simplified, as new routines in the UI need to only make a single function call from the ViewModel, which will then make the necessary calls to the back end's (Model's) relevant routines to prepare the data for the View.

As is standard with MVVM, system function calls should remain isolated to the Model and system UI function calls should remain isolated to the View, with the ViewModel mediating the

interaction via a single-entry point for that View feature. The ViewModel should also be responsible for updating the state of the Model, either triggered via user events or periodically on a timer.

5.3 Web API

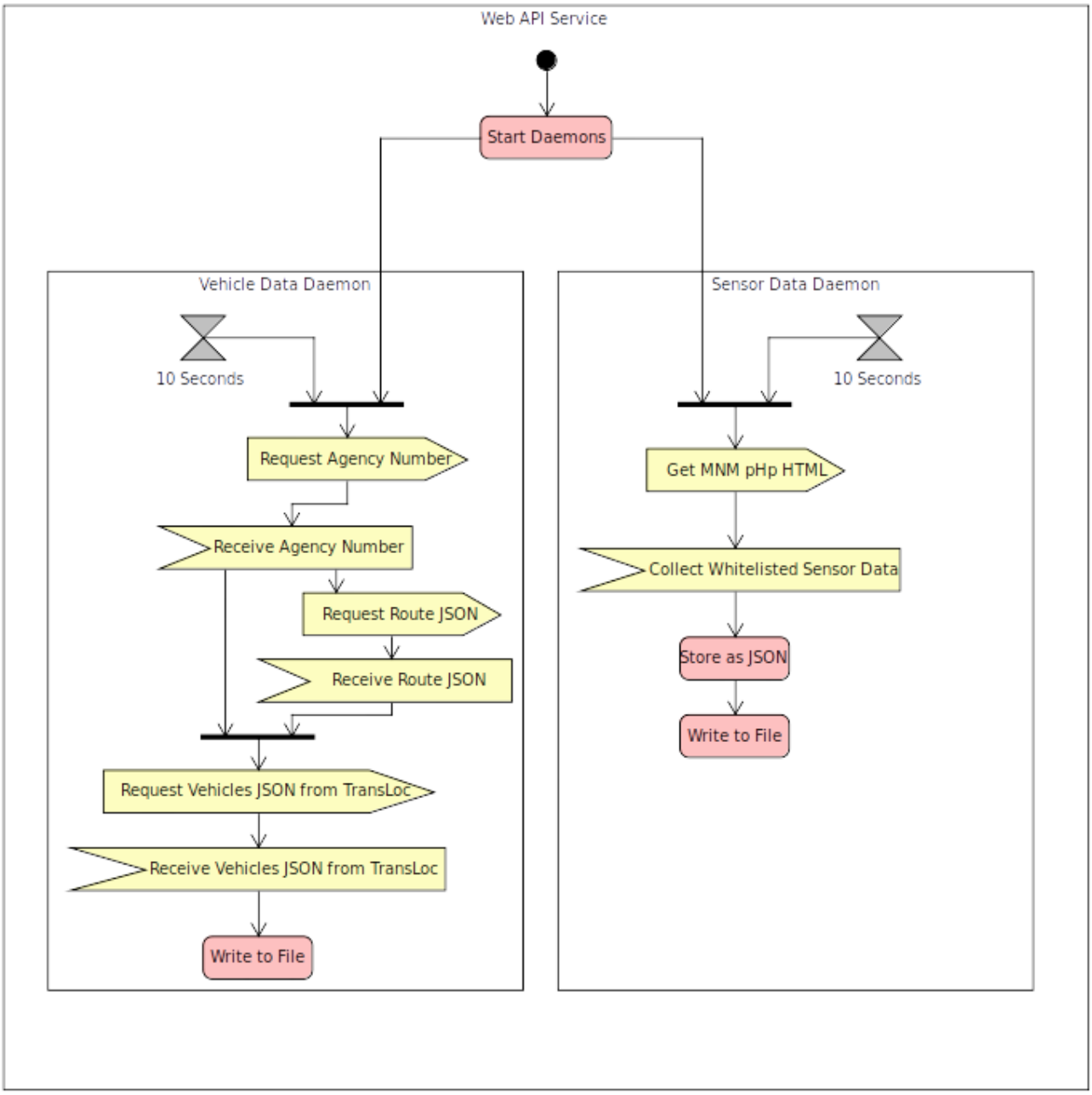


Figure 5-2. UML activity diagram for the BikeRide web API

Data from multiple sources must be gathered to meet the basic requirements of the mobile client application. Bus data (geolocation, route name, vehicle id, etc.) is hosted on the TransLoc OpenAPI while the bike rack sensor data (available slots, total slots, vehicle id, etc.) is hosted on an ECE SQL database.

There were two ways to tackle this problem. The first option was to write the mobile client application to request data from both websites and handle parsing the data. The second option was to write the mobile client application to request data from a single custom API that would be responsible for collecting the relevant data from TransLoc and the ECE SQL database.

The second option was the obvious choice.

While the first option potentially allowed for the elimination of the custom API requirement (and the maintenance costs that come with it), there were both technical and long-term planning problems with this first approach.

The most pressing issue was the fact that there was no operational RESTAPI for the ECE SQL database. Mobile applications (on Android and iOS) generally cannot make explicit SQL requests, and even if they could, the ECE SQL database blocks traffic from IP addresses outside of the University of Florida's designated prefix. Furthermore, allowing mobile applications direct access to the SQL database via login credentials represented a major security risk: Android apps can be easily decompiled, analyzed, and modified, so the login credentials could be extracted for malicious use or the application could be modified to send malicious SQL queries to the SQL database.

The first option also presents itself as a maintenance issue. If behaviors of the dependent systems change (either TransLoc or the ECE SQL situation), large portions of the mobile application's backend would have to be rewritten to fit the changes to the data format. By moving the data gathering responsibility off of the device, we not only decrease the power consumption of the mobile application, but we also gain control of the protocols that we use the communicate with the application in terms of URL name schemes, API method, and data format. This means that if changes happen to the external API format or source (but the data is still provided), modifications can be made on the server end without having to modify the mobile application and issue a user update.

5.4 Moving Forward with Deployment and Long-Term Support

The project is currently in good shape for future modification and maintenance.

If the FDOT is committed to the long-term deployment and usage of the application, it is necessary for the FDOT to do two things.

First, the necessary budget would need to be allocated for the annual maintenance of the following:

- Apple Developer License ($99 per annum for individual, $299 per annum for enterprise)
- Google Developer License ($25, one-time fee)
- Cloud Services Fee ($65.34 per month**, $784.08 per annum**)
    - Recommended hardware requirements*
        - Always-on/Non-preemptible
        - 2 CPU cores
        - 8 GB RAM
    - * Requirements based on temporary development implementation created for application demo/proof of concept. Actual requirements may increase or decrease depending on the production implementation (most likely increase, as increased web traffic will need to be considered).
    - ** Cloud price estimation based on Google Cloud Platform pricing for a non-preemptible n1-standard-4 server in the South Carolina region with 3 years committed usage and 30GB of storage. Prices may fluctuate as Google's rates change monthly.

This works out to an estimated total of $1,108.08 for the first year and $1,083.08 for subsequent years, ignoring inflation and future price hikes to cloud service costs.

Second, it would be preferential for the FDOT to hire a full-time software engineer dedicated to the project, or at the bare minimum, have the project added to the responsibilities of an existing software engineer. Relying solely on part time contract workers will make it more difficult, if not impossible, to realize the vision for the project; people come and go, and proper documentation and coherent implementation tends to become lost in the process. Issues with this has already been encountered: attempts to gain insight into the project's working history through prior collaborator McTrans were largely unfruitful: the necessary documentation and source code for the existing web API and web application was incomplete, the original McTrans team members could not be contacted, and McTrans was not committed to continue operating the web API because the original grant had expired.

The nature of the work also points to the requirement of full-time maintenance. A mobile application needs constant updates to remain usable through mobile device updates (mobile device operating systems and hardware are constantly in flux). Web servers, even if run through a cloud service provider like Amazon AWS and Google Cloud Platform, require real time support to monitor and react to changes in the security, operation, and performance of the web server in a timely manner. These sorts of demands simply cannot be fulfilled by part time contract workers, due to the time overhead of searching for, hiring, and training workers (it can

take time for developers to acclimate to a system they have not seen before) every time a new problem arises.

6 Configuring, Building, and Installing the Bikeride Application

6.1 Introduction

The following will detail how to set up the build environment, configure the application to use the latest servers and accounts, build the application, and install the application on a development device. Instructions for setting up the build environment in Windows is not detailed but can be found in the documentation of each tool used in the process. Simply follow the download links to each component, and the install instructions for Windows can be found on each download page. Detailed MacOS and Linux instructions are provided due to their added complexity and relatively poor official documented support.

6.2 Installing Flutter SDK

First, download the latest Flutter SDK package.

6.2.1 MacOS/Linux

Once you have that, do

`cd ~/Downloads`

`sudo mv ./flutter /opt/flutter`

to install the application.

If you want to add it to the path, do

`sudo ln -s /opt/flutter/bin/flutter /usr/local/bin/flutter`

Now run

`flutter`

to make sure the SDK was installed correctly.

In order to make sure dependencies were installed, run

`flutter precache`

followed by

`flutter doctor --android-licenses`

You must accept the licenses to use Flutter (enter y for all).

6.3 Installing Android Studio

First, download the latest Android Studio package.

6.3.1 MacOS

Install the .dmg like you would with any other MacOS application.

6.3.2 Linux

Once downloaded

```
cd ~/Downloads
```

```
sudo tar -zxf <ANDROID_STUDIO_PACKAGE_NAME> /opt/
```

```
sudo ln -s /opt/android-studio/bin/studio.sh /usr/local/bin/android-studio
```

to install Android Studio and add it to the executable path.

To add a desktop icon for easy access, run

```
android-studio
```

and in the GUI, navigate to Tools -> Desktop Entry at the top toolbar.

6.4 Configuring Android Studio

Open Android Studio and go to the Plugins section under Android Studio -> Preferences (⌘,). Search the Marketplace for Flutter and download that plugin.

Since we installed the Flutter SDK in /opt/flutter, we will have to make sure Android Studio is aware of the path.

To do this, navigate to Android Studio -> Preferences (⌘,). Open the Languages & Frameworks tab and select Flutter. Under the field *Flutter SDK Path* in the *SDK* block, enter

/opt/flutter

Then select Dart under the Languages & Frameworks menu tab. Under the field *Dart SDK Path*, enter

/opt/flutter/bin/cache/dart-sdk

Because we are targeting Android Lollipop 5.0 as a minimum requirement, we will need to install the SDK for it manually, as it is no longer default (at time of writing, Android Pie 9.0 is the default).

Under the Appearance & Behavior tab, open the System Settings tab and select Android SDK. Under SDK Platforms, check the box next to *Android 5.0 (Lollipop)* to install it and hit *Apply*. Follow the install prompt and wait for completion.

If you do not have a physical Android Lollipop 5.0 device, you also need to install the Android Lollipop 5.0 Emulator. To do this, navigate to Tools -> AVD Manager.

(If this option is missing, try restarting Android Studio to see if additional components need to be installed, specifically Intel HAXM. You should be prompted on the bottom right hand corner.)

Select *+ Create Virtual Device* and select any hardware to emulate (Nexus 5X is a good option) and hit *Next*. In the System Image page, select the x86 Images tab and scroll down to "*Lollipop* Download |*21 * | *x86_64 * | *Android 5.0 (Google APIs)*" and select the *Download* link. Follow the install prompt and wait for completion. Once complete, select the system image you just downloaded and hit *Next*. On the Android Virtual Device page in the *Emulated Performance* block, change the *Graphics* field from *Automatic* to *Hardware - GLES 2.0*. Hit *Finish* to add the device to the AVD list.

6.5 Wrapping Up the Installation

Once everything else is done, plug in an Android device (and/or iOS device if you are running MacOS) to your computer if you have one handy.

Then run

```
flutter doctor
```

to see if everything was installed correctly.

If you did not plug in a device, expect

```
[!] Connected device
    ! No devices available
```

If there are no other errors, everything is properly configured. Otherwise, follow the instructions that were given by the flutter command to resolve the errors.

6.6 Cloning the Project

6.6.1 Using Android Studio's VCS

In the top toolbar, navigate to Android Studio -> Preferences (⌘,). Open the Version Control tab and select GitHub. Add your GitHub account to the entry list using the *+* button at the left-hand corner of the table. Follow the prompts on the display. Hit *Apply* and *Ok*.

In the top toolbar, navigate to VCS -> Git -> Clone.... Input the URL of this git

https://github/austinjkee/bikeride.git

into the field. Hit *Clone*. When prompted to add the newly cloned source as an Android Studio project, select *Yes*.

If you accidentally selected *No*, go to the section on Manually Adding The Project To Android Studio.

6.6.2 Using Git CLI

If you want to use this option, you likely already know how to use Git CLI, but out of an abundance of caution:

```
cd <PATH_TO_DEVELOPMENT_WORKSPACE>
```

```
git clone https://github/austinjkee/bikeride.git
```

6.6.3 Manually Adding the Project to Android Studio

Manually adding the Flutter project to Android Studio is necessary when using Git CLI or if something went wrong in the automated GUI cloning.

To do so, open Android Studio to the Welcome to Android Studio window.

Select the *Open an existing Android Studio project* option and navigate to the location of the cloned project.

It should add it properly if Flutter was installed correctly; if Android Studio prompts to use Gradle with the project, something is wrong with the Flutter install and

```
flutter doctor
```

should be run to verify.

If the status comes back clean, restart Android Studio and try again.

6.7 Building the Project

6.7.1 Ensuring the Project is Functional

The developer Google Maps API key has been removed from the project.

When ready for production, a new Google Maps API must be provided in the following files:

ios/Runner/AppDelegate.swift

```
  ) -> Bool {
    GMSServices.provideAPIKey("API_KEY_STRING_GOES_HERE")
    GeneratedPluginRegistrant.register(with: self)
```

android/app/src/main/AndroidManifest.xml

```
  <meta-data android:name="com.google.android.geo.API_KEY"
    android:value="API_KEY_STRING_GOES_HERE"/>
```

6.7.2 Generating App Code and Building the App for Target Devices

Once the project has been given a valid Google Maps API key, you can build the program from the flutter source to the target device native code.

In Android Studio's Terminal or in your preferred shell environment, navigate to the main directory of the project and run:

```
flutter build
```

This will generate the necessary code for app creation.

6.8 Disclaimer

Building an iOS version of the application requires an Apple computer with at least macOS Mojave/Xcode 11.3 for up to iOS 13.2 and macOS Catalina/Xcode 11.4 for iOS 13.3 and above.

To build the application binaries for target devices, first make sure you have a valid device attached, either physical or virtual (simulators).

6.9 Android Minimum Requirements OS

Marshmallow 6.0 (API 23) Physical: 64-bit ARMv8 Compatible CPU Software: Google Play Services Enabled

6.10 iOS Minimum Requirements OS

iOS 12.4.5 Physical: iPhone 5S (Apple A7) or newer Software: macOS Mojave/Xcode 11.3, macOS Catalina/Xcode 11.4 Preferred

The application can theoretically be built for iPad, Apple TV, Android TV, and as a WebApp, but they are considered niche use cases and will not be covered here.

To make sure the device is attached and recognized, in Android Studio's Terminal or in your preferred shell environment, run:

`flutter doctor`

Once you have verified that you have a device attached, proceed with running the flutter native builder.

Optionally, you may verify that the app builds correctly before bundling.

In Android Studio's Terminal or in your preferred shell environment, navigate to the main directory of the project and run:

`flutter run`

It should run on the attached device and all features should w

work if the minimum requirements are met.

To build a bundle for release, the package must be signed with a production key. Further instructions for release can be found here:

https://flutter.dev/docs/deployment/android

PART E: SUMMARY AND CONCLUSION

7 Summary and Conclusion

7.1 Summary

For residents in the City of Gainesville, the Regional Transit System (RTS) is important for travel to work, school, or other destination. But, because all destinations cannot be reached directly using the RTS, people need to use bikes, resulting in Alachua County having the second highest bicycle mode share in the state. An apparent problem is capacity constraint because the two-slot bike racks currently available on RTS buses do not provide enough bike slots. Our team collected data for rack usage to determine whether adding three-slot bike racks would resolve this problem by, as shown in Part C.

The bike rack sensor fabrication procedure is shown in Part A, with a very detailed explanation of each step. The estimated cost of installing the bike rack sensor in the front of each bus was also calculated, showing a result of $121.90 per bus, The additional cost for a cellular plan to connect the sensors was $7.72 per bus.

In Part A, we described the hardware. In Part B, we described the software aspects of a mobile application through which passengers can get real-time data about the availability of bike slots or buses. Part B covers the many aspects of the mobile application, including the user interface, functional and nonfunctional requirements, and a tutorial for how to use the BikeRide mobile application step by step. At the end of Part B, we provide the code for the application.

In Part C, we present the data collected from the bike rack sensors installed on several RTS buses. For the one-year period from May 1, 2018, to April 30, 2019, we sorted the data by different time conditions: daily, weekly, monthly, seasonally. Daily usage was well matched with commuting and school hours. Weekday usage was higher than weekend usage. Usage was low in March, apparently because of spring break. The highest seasonal usage was in summer, but seasonal differences were not large. Interestingly, Saturday usage was highest during the fall semester, corresponding to colleg football games. Data correlated with bus route showed which routes most need larger bike racks.

In Part D, we suggest the current state of the project and recommendations. The detail explanation of UML activity diagram for BikeRide mobile application and web API is shown, and two obstacles for this project are identified. The system is expected to be operated without malfunction and generating too much maintenance costs for each part. For this, the costs for maintenance are mentioned with some examples about how to handle the maintenance issue. At last point of Part D, there is guidance for configuring, building, and installing the BikeRide application for each operation system (Android, MacOS/Linux).

7.2 Conclusion

If any problems or inconveniences happen to us, we want to handle it obviously. But it cannot be always resolved with feasible techniques when it comes to considering about the expenditure for public facility. That is why this project has meaningful values. The reasons are as follows.

First, we are suggesting detailed installation costs for bike rack sensors and bike rack mobile application through examples and actual installations, which will be a great help for accurate budgeting.

Second, from the data obtained by the installed bike rack sensors, it was possible to grasp how many people actually use the bike rack, and it is possible to install and change the bike rack flexibly by predicting the trend of usage change according to their needs, time, and environment also resulting in avoiding wasting money.

Third, by making a mobile application that can check bike rack slots in real time and making it easy, it follows the trend well-suited to the modern mobile environment, and if it can be applied to a bus location system such as an RTS application, it is meaningful enough.

Unlike other big cities, the eco-friendly City of Gainesville has buses for transportation, but it is quite difficult to use for those in the suburbs that are not close to the bus route. In particular, on weekends and at night, the traffic volume of the bus is drastically reduced, so many people experience inconvenience. To alleviate this discomfort, if we can build an environment where people can use both buses and bicycles by having the infrastructure to use bicycles, this will lead to a great improvement in the quality of life for the residents of the Gainesville area and other areas wishing to consider such a system.

Possible following-up efforts and projects would include a pilot project testing the developed cell phone app for the biker riders in the City of Gainesville and Alachua County. An extended application could include a smart bike parking station on campus, where each bike rack in the parking station is equipped with a smart sensing module enabling us to quantify the usage of bikes on campus. Also, this system could be used in a eco-friendly carbon-free smart city, where people ride bikes in a residential area and commute via a multi-modal transportation system.