Software Engineering

RU Bus Management System

Report 2

Group #11

Yaocheng Tong
Yufeng Lin
Minghao Qin
Yuhai Zhang
Lu Jin
Haowei Li
Aaron Hu

Instructor: Prof. Ivan Marsic
Teaching Assistant: Parsa Hosseini
Graders: Aymen Al-Saadi, George Koubbe

Github Page

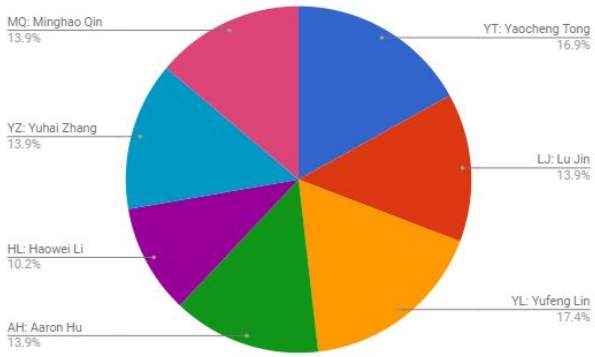
Spring 2018

Table of Content

Individual Contributions Breakdown

Report 2	Score	Team Member Name						
		YT	LJ	YL	AH	HL	YZ	MQ
Interaction Diagrams	30	4.29	4.29	4.29	4.29	4.29	4.29	4.29
Classes + Specs	10	1.43	1.43	1.43	1.43	1.43	1.43	1.43
Sys Arch & Design	15	2.5	2.5	2.5		2.5	2.5	2.5
Alg's & datastruct	4	0.66	0.66	0.66	0.66		0.66	0.66
User Interface	11			5.5	5.5			
Testing design	12		3	**3**			3	3
Project management	18	8	**2**		**2**	**2**	**2**	**2**
Total	100	16.96	**13.88**	**17.38**	**13.88**	**10.22**	**13.88**	**13.88**

Name	Rate
Yaocheng Tong	
Lu Jin	
Yufeng Lin	
Aaron Hu	
Haowei Li	
Yuhai Zhang	
Minghao Qin	

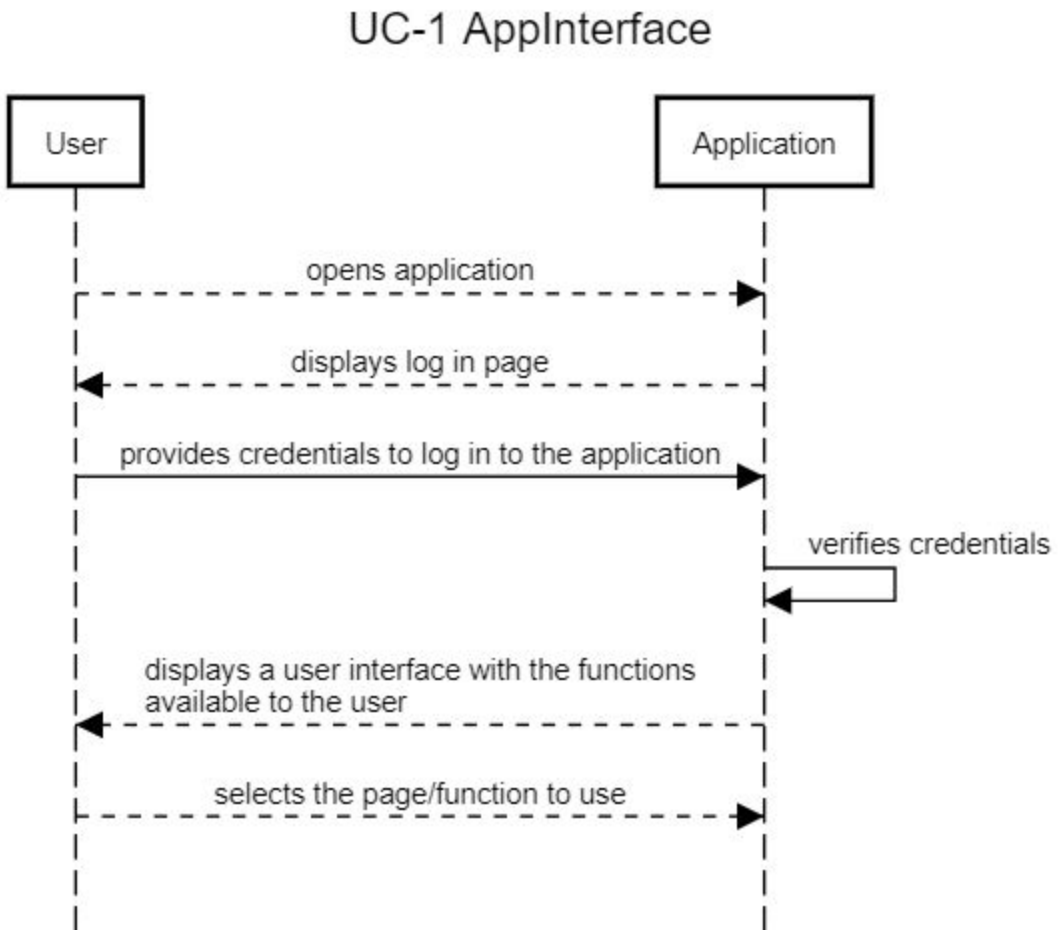


Summary of Changes

Unfortunately one of our group members (Viraj Patel) has dropped the class, so our group currently have 7 people.

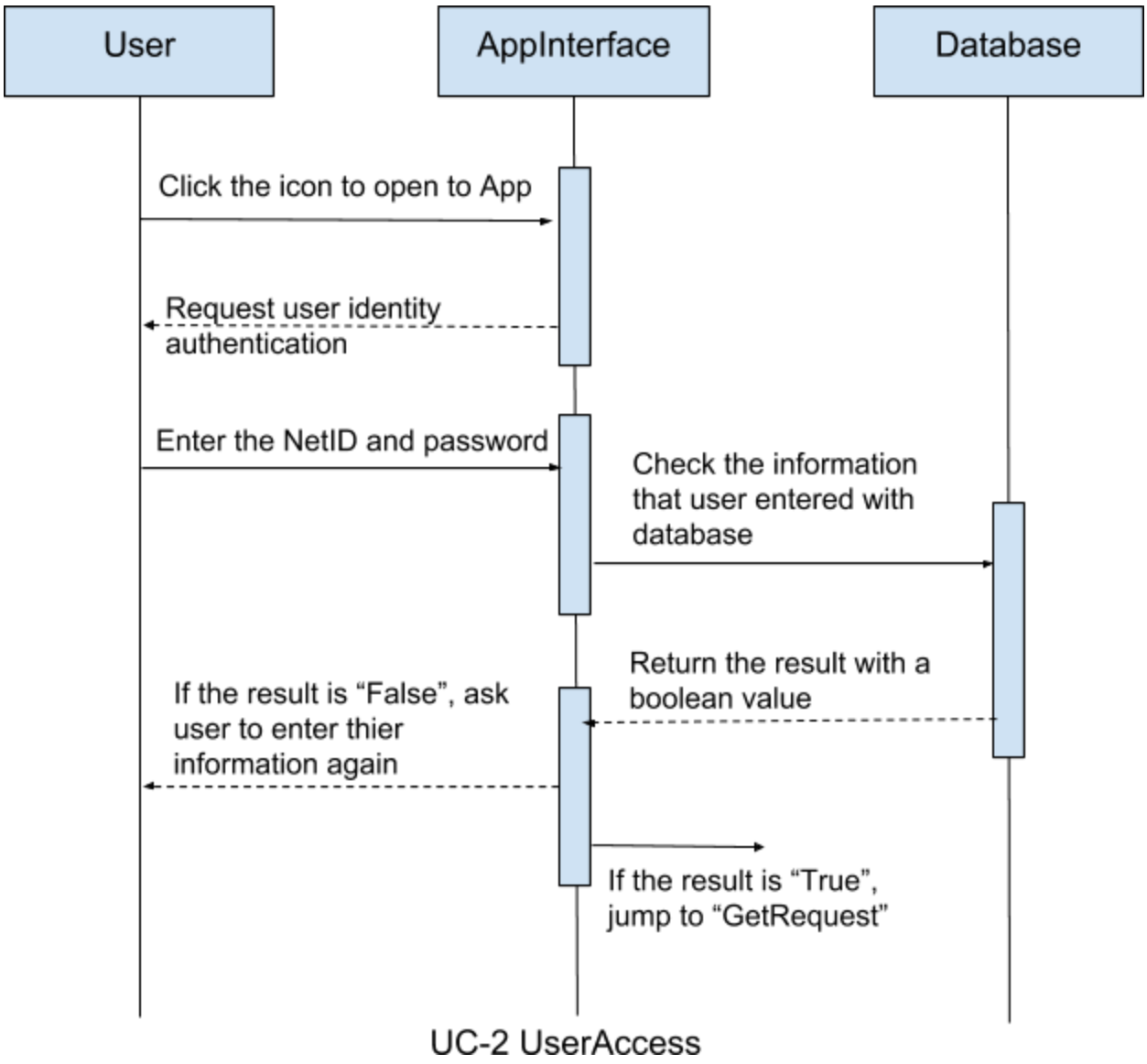
Section 1. Interaction Diagrams

UC-1



The App interface is designed to interact with the built in user functions. The main purpose of App interface is to help users visualize which button or function the they want to use. The app interface is also based on the successful implementation of functions and display correct pages when needed.

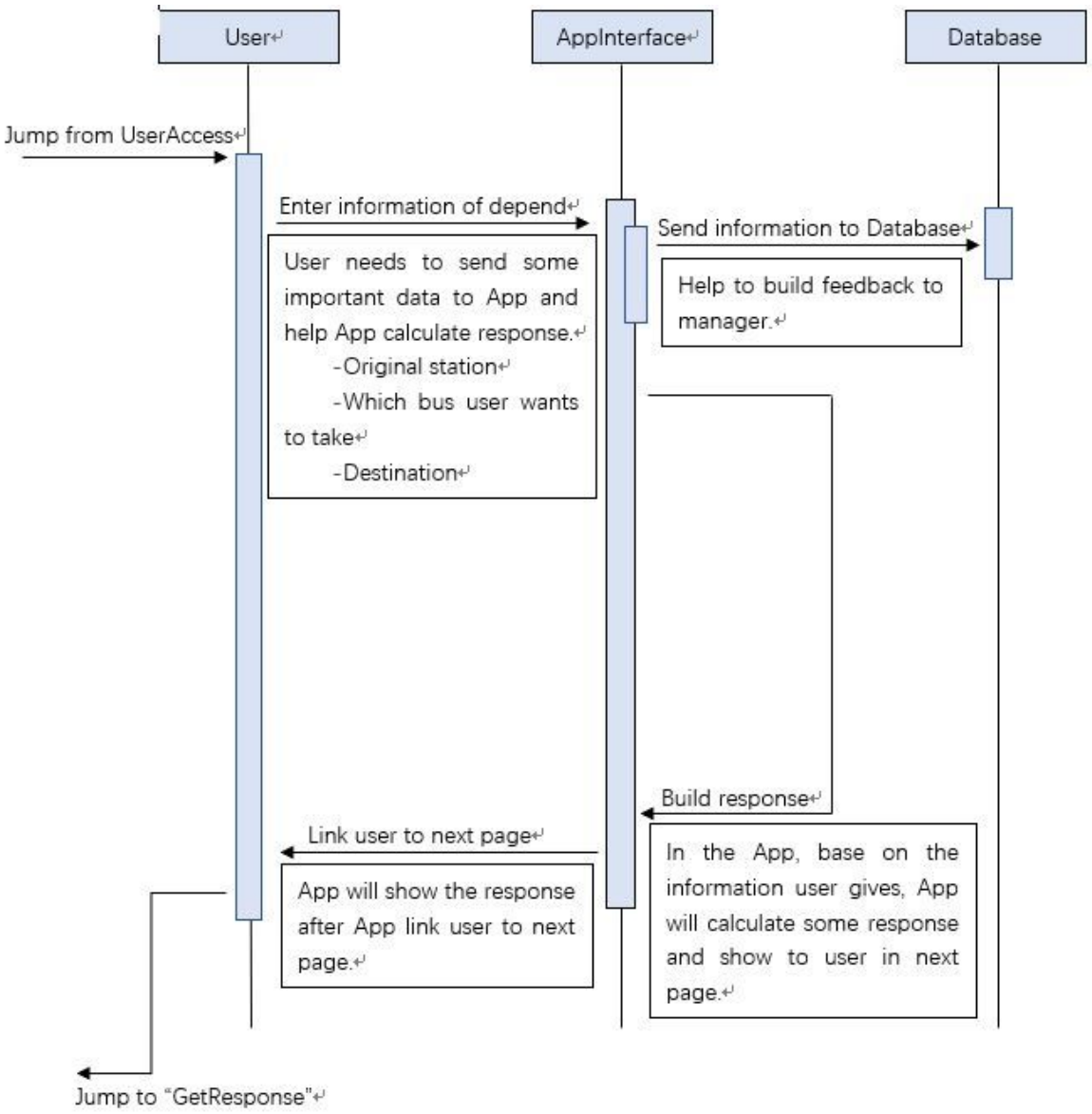
UC-2



UC-2 UserAccess

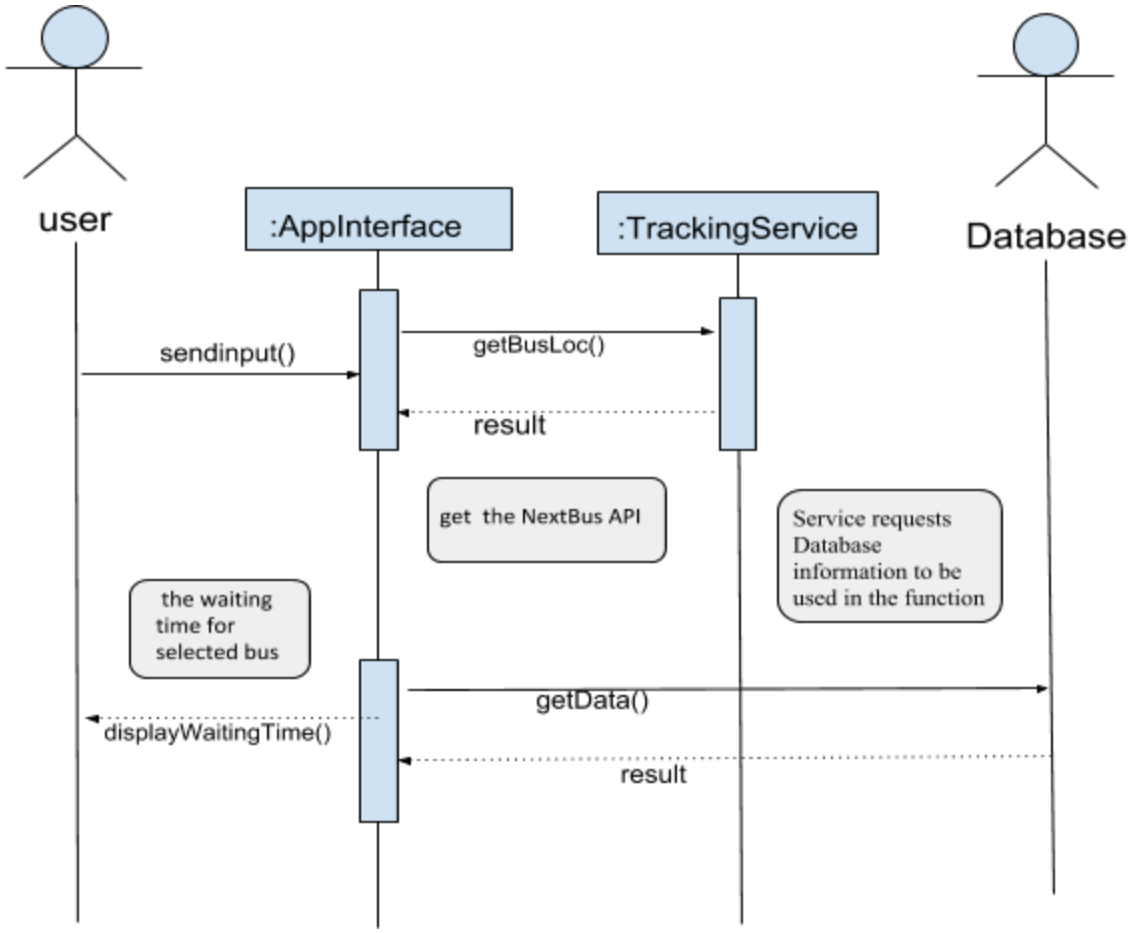
This use case has the responsibility of checking user’s identity. The AppInterface provides an interface to let user entering their information then AppInterface send the information to Database. Based on the information stored, Database will have a output with a boolean value. If it is true, that means the information that user entered match the information stored in repository. If it is false, that means match failed and Database will ask AppInterface to ask user enter their information again. The high cohesion principle applied to this use case since the role of Database is checking the information only.

UC-3



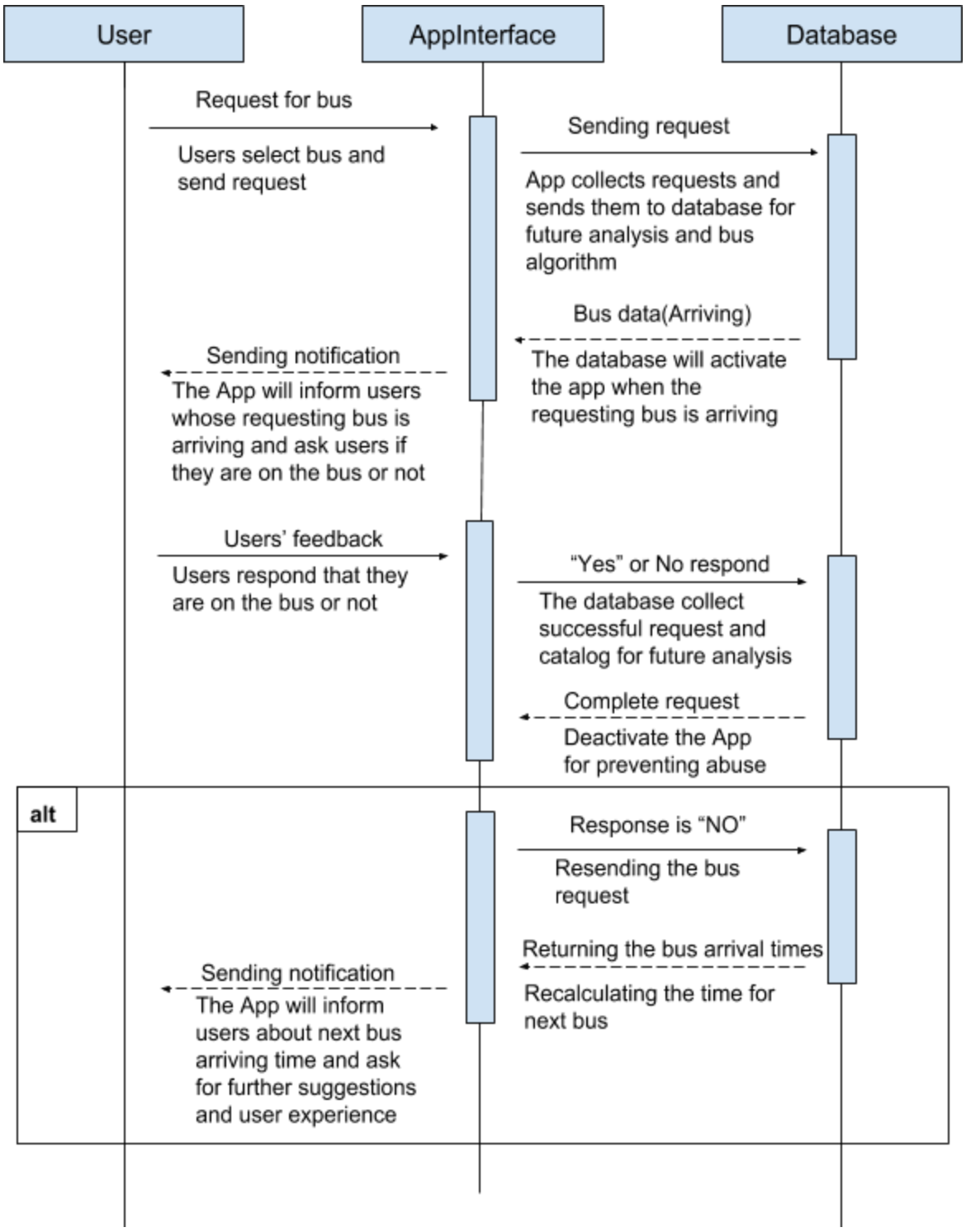
This use case is to call users to enter what their demands. In this page, users need to enter original bus station, destination and the bus they want to take. Appinterface will send information that is from users and will link users to next page to receive response.

UC-4



The use case has the responsibility to know the estimated time of arrival (ETA) for buses. user enters his or her data to the bus Application and sends user selected bus to the Tracking Service. The Tracking Service requests Database information to be used in the function and display the correct estimate waiting time.

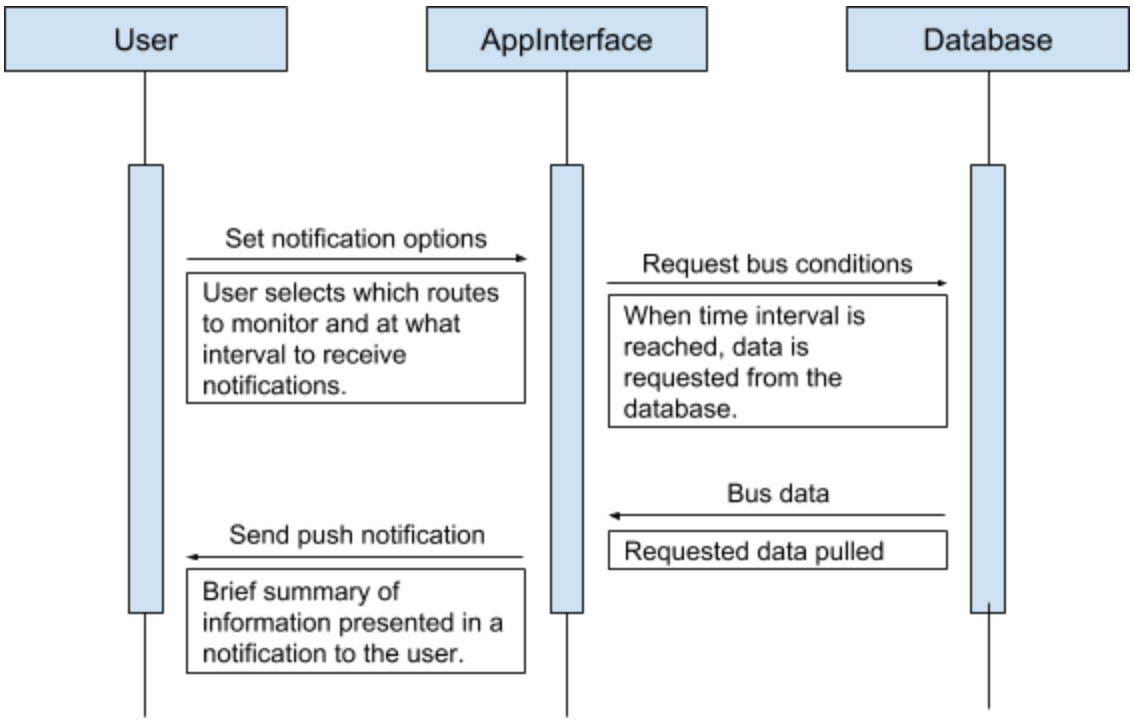
UC-5



This diagram represents UML for UC5 which is responsible for check users' status and feedback. The Appinterface's duties are informing users about their requesting buses and classifying their request for database. The duties for database are

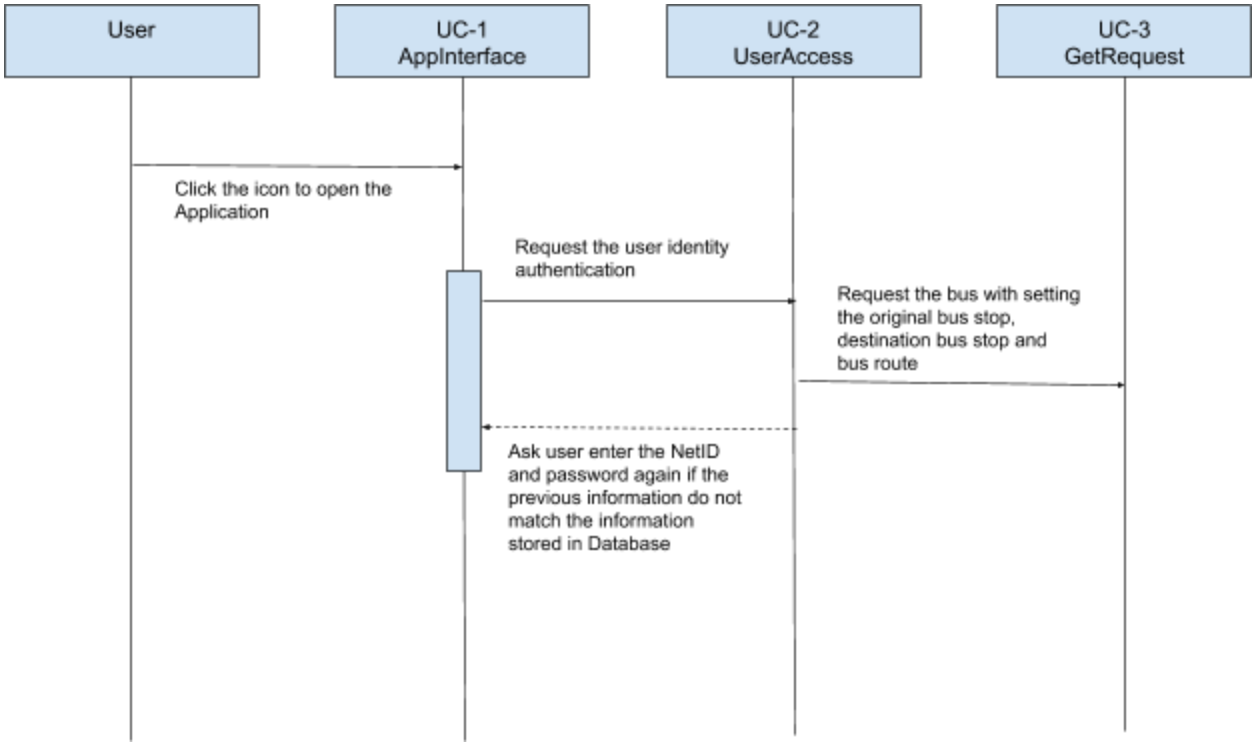
analyzing users' requests and determine if users get on the bus or not and make relative decisions. There is no obligation for users to respond for checking status. However, if their requesting buses pass the station and they do not respond, the system will automatically assume they are on the bus and complete the request for preventing abuse.

UC-6

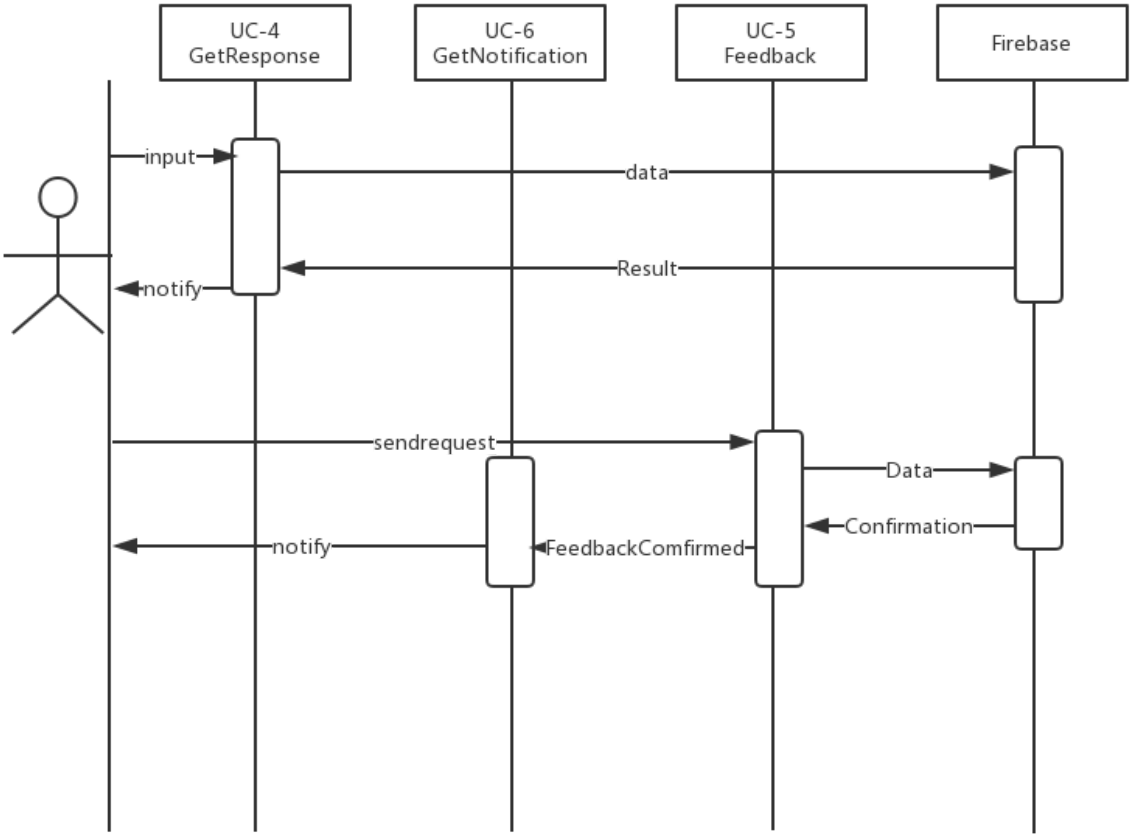


This use case is the notification system of the mobile app. It will occur while the app is being used as a background service, so it has minimal interaction. No user input is required for this function aside from the precondition of notification preferences being set beforehand. The android notification API will be responsible for the majority of the function and a simple request for data is all that is required to complete the entire process.

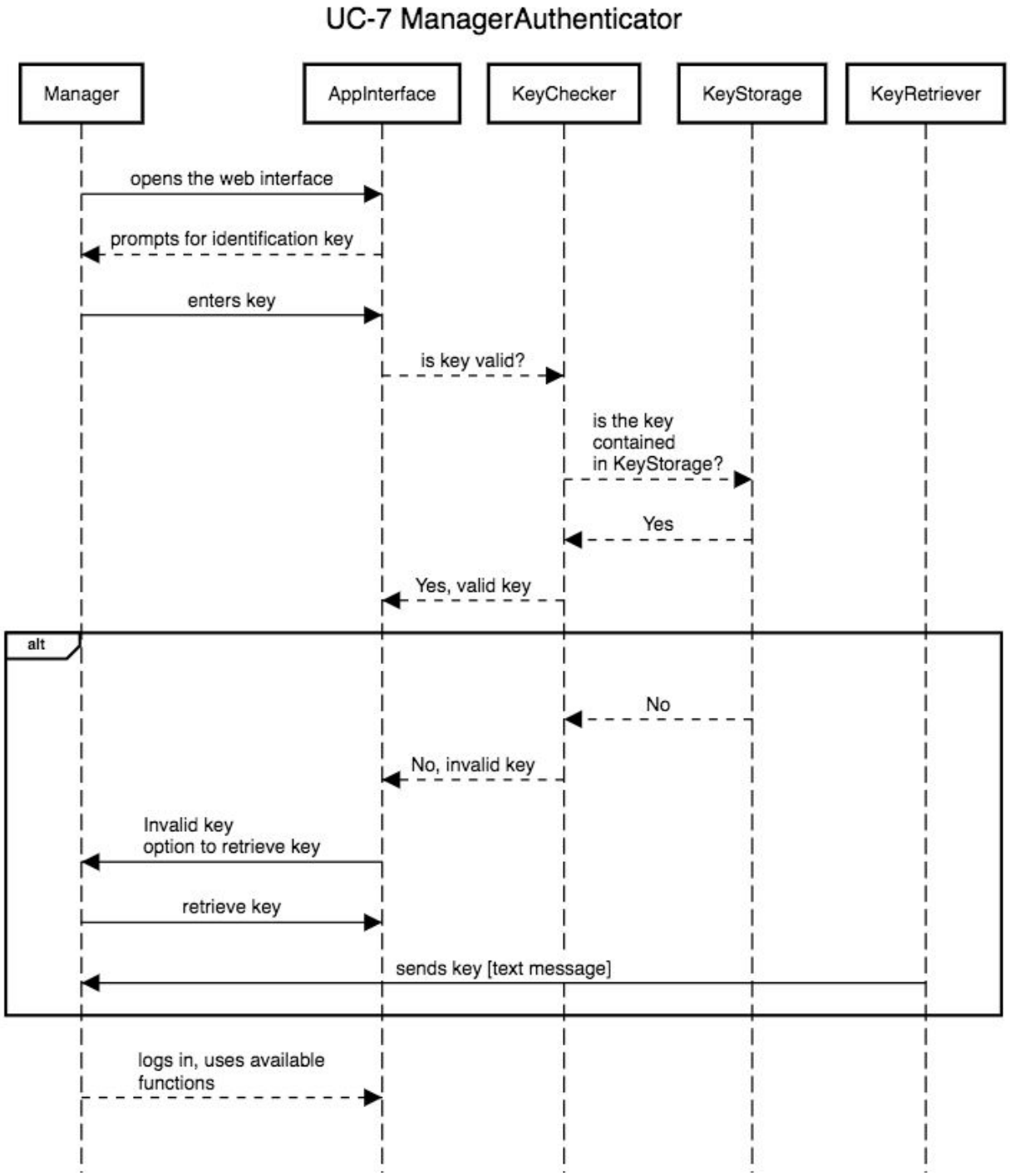
User Case 1-2-3


User
UC-1
AppInterface
UC-2
UserAccess
UC-3
GetRequest
Click the icon to open the Application
Request the user identity authentication
Request the bus with setting the original bus stop, destination bus stop and bus route
Ask user enter the NetID and password again if the previous information do not match the information stored in Database

User Case 4-5-6



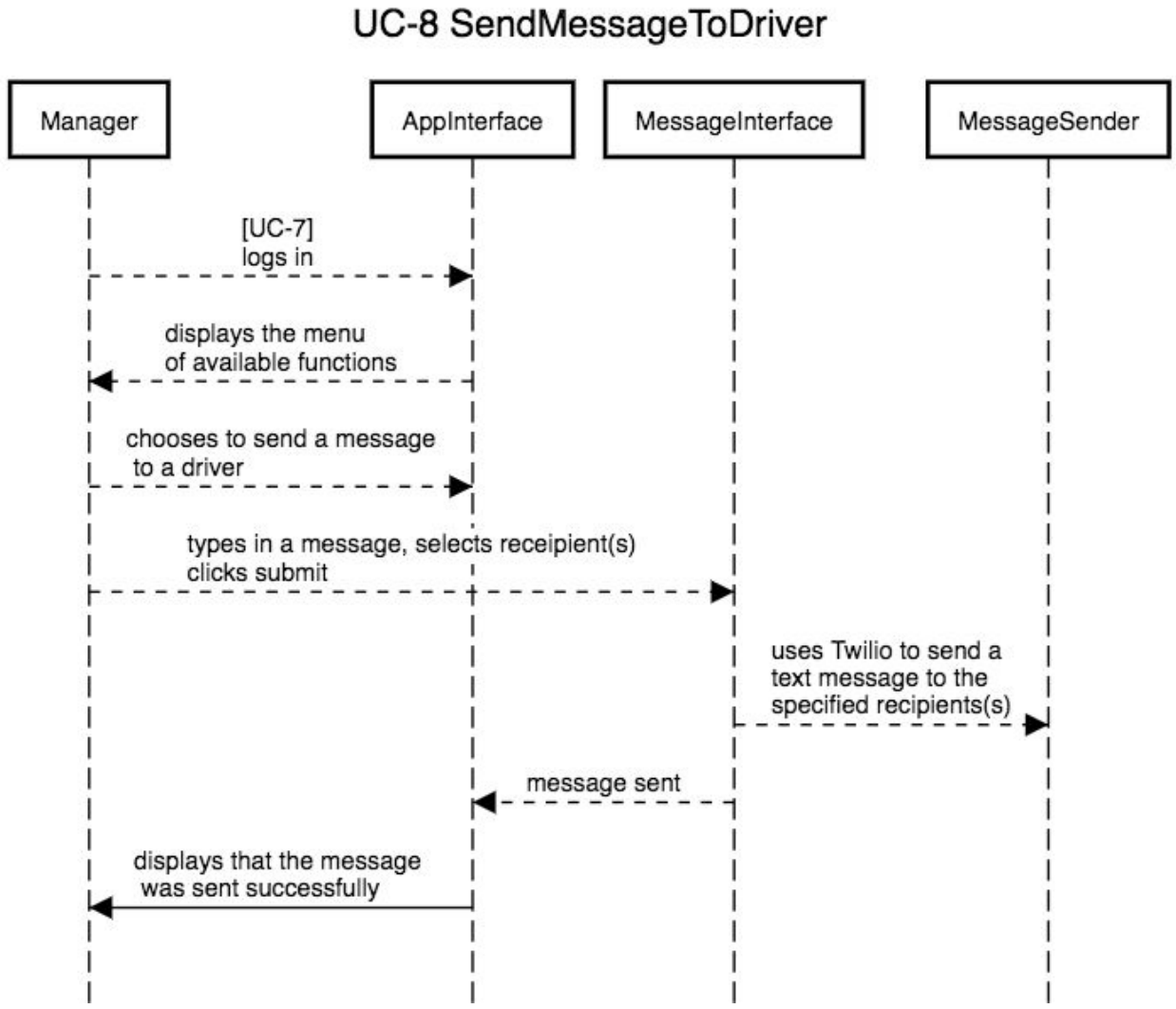
UC-7 ManagerAuthenticator



This web interface is to be used by the managers to log in to the bus management system in order to perform their day-to-day tasks. The manager plugs in the key into the AppInterface, which then uses the function KeyChecker() to check in the KeyStorage if the key is contained in the KeyStorage and if it is valid or not. If it is

valid, the KeyChecker will return to the AppInterface that the key is valid. In an alternate scenario, if the key is not valid, the manager will be given an option to retrieve the key. At this point, the AppInterface will invoke the KeyRetriever to send a text message to Manager's cell phone containing the valid key.

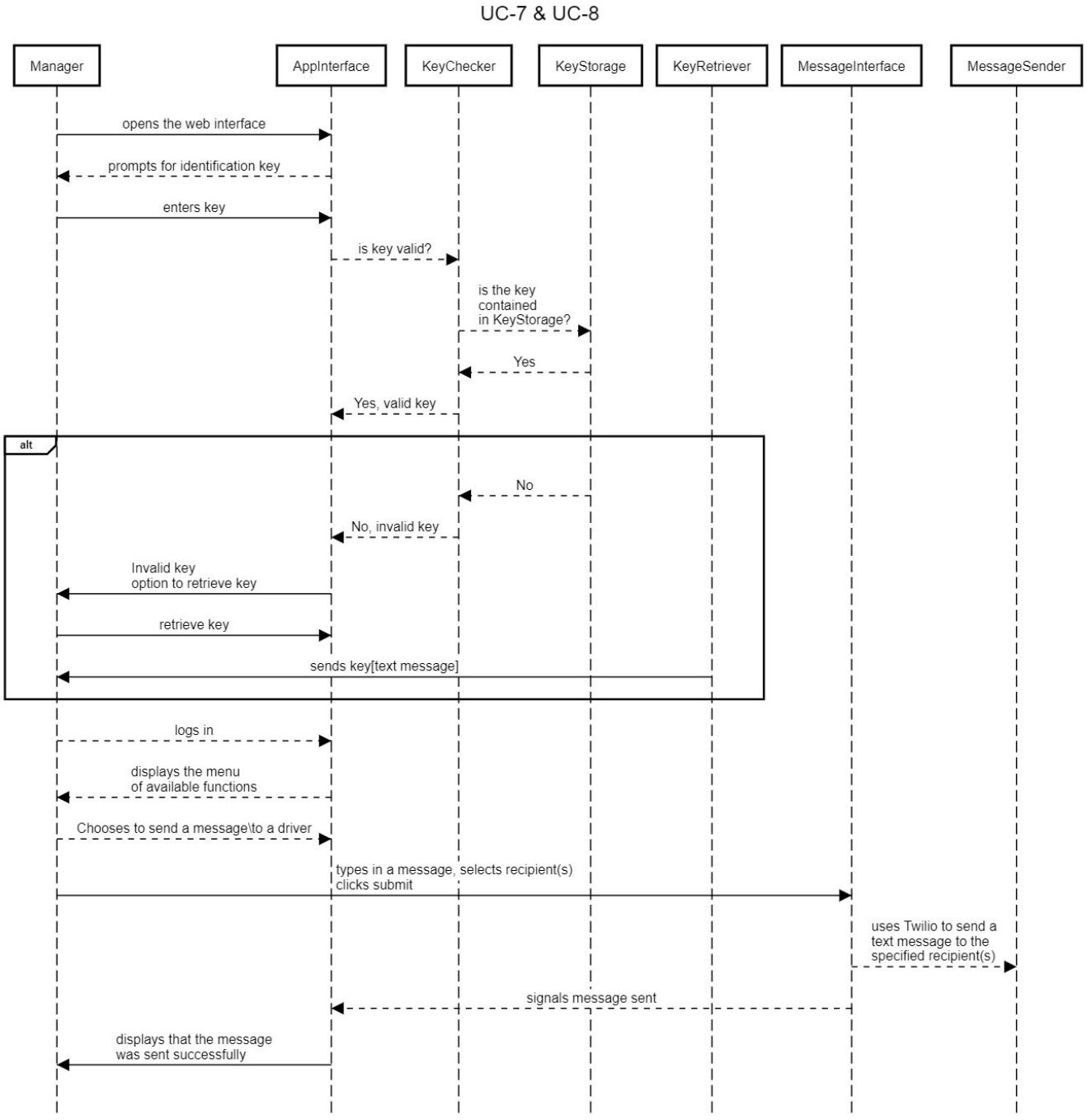
UC-8 SendMessageToDriver



Here, we assume that the Manager is logged in to the AppInterface and chooses (from the menu of available functions) to send a message to a driver or broadcast a message to all the drivers. He then types up the message in the MessageInterface, which is a part of AppInterface, and clicks submit. The MessageInterface then invokes MessageSender to use Twilio text messaging service to send the message

to the selected recipient(s). The MessageInterface then signals the AppInterface that the message was sent, after which the AppInterface displays to the Manager that the message was sent successfully.

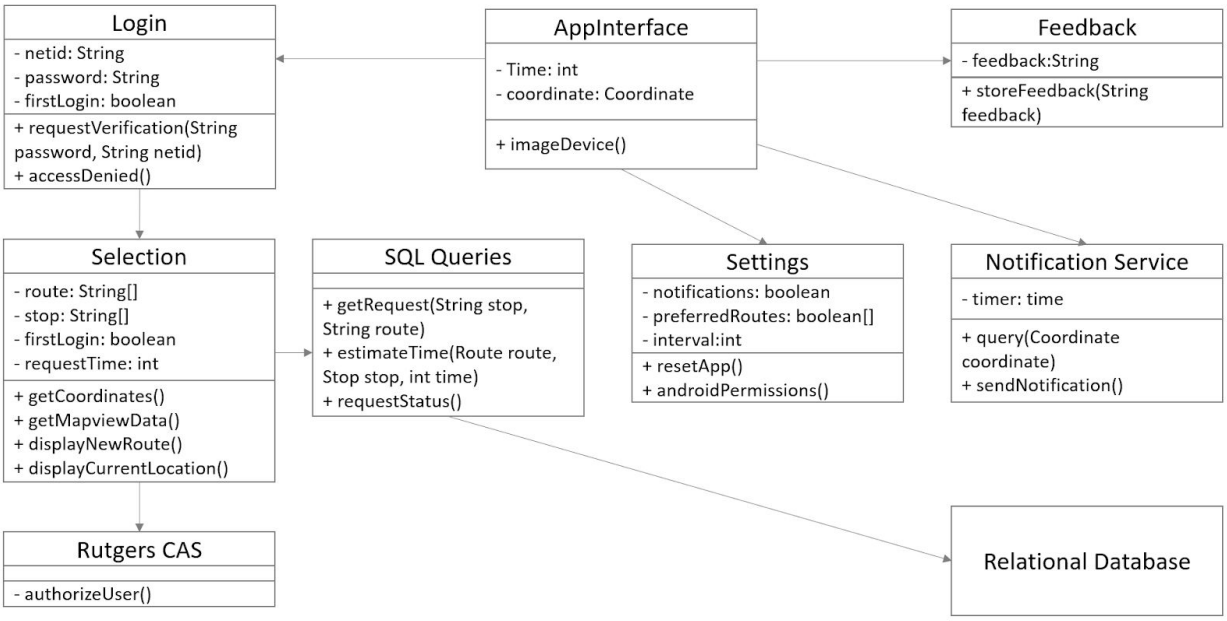
UC-7 and UC-8



Section 2. Class Diagram and Interface Specification

A. Class diagram

Mobile Application



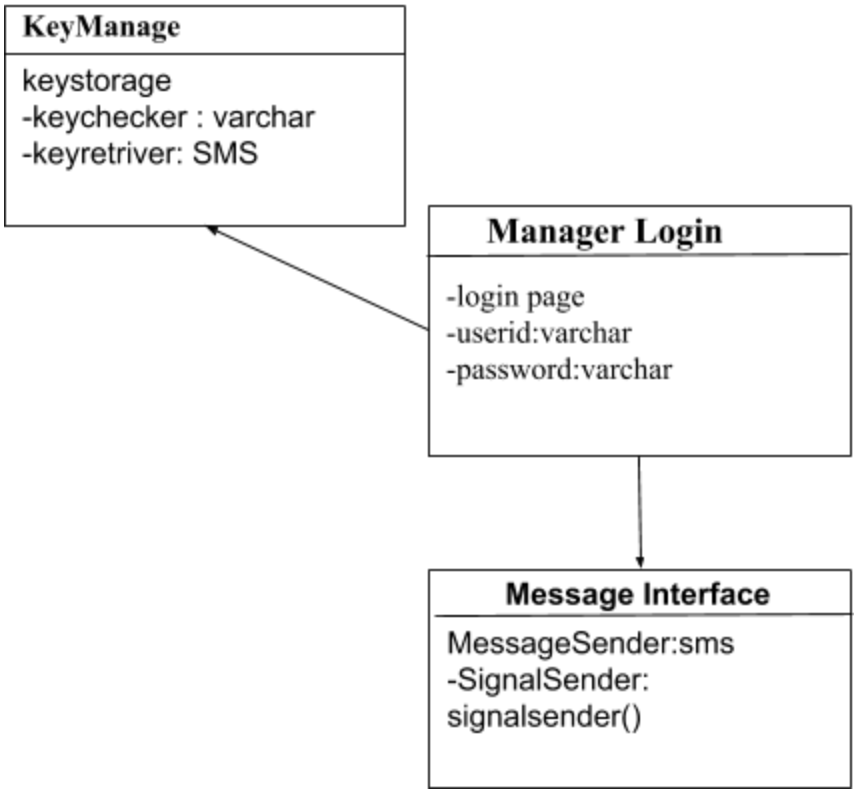
Use Case 2:
From the Login and Selection classes, the Rutgers CAS class can be reached. The user enters their NetID and password into the login screen prompt. Rutgers CAS authenticates the user's input information. The Relational Database can be reached by the Rutgers CAS. This system is used to count the number of users and provides data that helps us optimize the user experience during login.

Use case 3:
Feedback needs to monitor demands from the users (original stop, destination, etc.). The system will send this information to the database.

Use case 4:
A response page will display an ETA. The system will fetch the GPS coordinates from the NextBus API to track the bus's current location and show the real time of bus (debating implementation because the given coordinates from the NextBus API are relatively inaccurate and increase user frustration).

Use Case 5:
The feedback class can collect and classify students' responses of whether or not they have boarded the bus. The device will make decisions according to their responses. The "Yes" and "No Response" will be sorted as a successful request in the database and the "No" requests will be treated as failures. They will be resent to the request path and will repeat the steps again until they turn into successful requests and are sorted in the database.

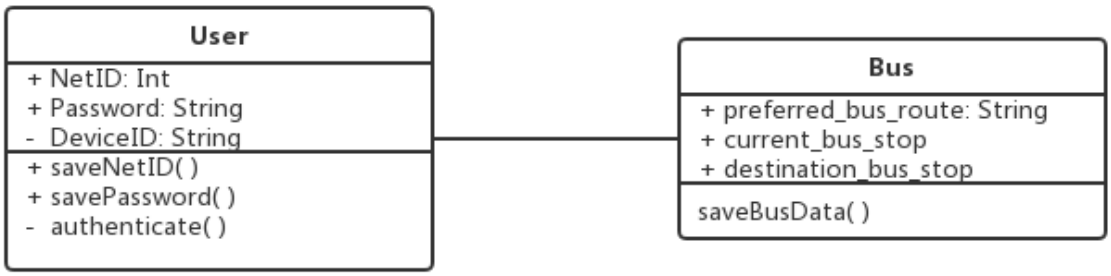
Manager Interface



Use Case 7 & 8:

These two use cases will be implemented in the manager interface. It includes two main parts: The first part is login page which is going to identify the authenticity of the users, the second part is going to monitor the real time buses requests and send messages & signals to drivers if necessary.

Database



B. Data Types and Operation Signatures

Mobile App

AppInterface - acts as a middleware class for various public functionalities

Attributes:

- Time: int // keeps current time as variable
- coordinate: Coordinate // keeps current coordinates as variable

Operations:

+ imageDevice() // UI elements

Login - CAS authenticate user's information then log user in

Attributes:

- netid: String // user input netid text
- password: String // user input netid password text
- firstLogin: boolean // tracks if app is launcher for the first time

Operations:

+ requestVerification(String password, String netid) // request authentication from CAS
+ accessDenied() // displays error message if CAS does not authenticate successfully

Feedback - allows users to give either text feedback or boarding/missing bus info

Attributes:

- feedback:String // input user feedback

Operations:

+ storeFeedback(String feedback) // sends feedback to database

Selection - allows user to choose their desired route and stop

Attributes:

- route: String[] // array that holds possible routes
- stop: String[] // array that holds possible stops
- firstLogin: boolean // detects first login
- requestTime: int // updates current time

Operations:

+ getCoordinates() // fetches current coordinates from Android location services
+ getMapviewData() // requests map data with Google Maps API key
+ displayNewRoute() // updates mapview to currently selected route profile
+ displayCurrentLocation() // displays a pin marker on mapview to current location

SQL Queries - contains various database query functions between the app and database

Operations:

+ getRequest(String stop, String route) // fetches user requests with given data
+ estimateTime(Route route, Stop stop, int time) // estimates ETA of next bus

+ requestStatus() // requests bus information about selected route

Settings - allows user to change notifications preferences or to reset the app

Attributes:

- notifications: boolean // allows user to receive notifications
- preferredRoutes: boolean[] // currently selected routes
- interval:int // time interval at which to receive notifications

Operations:

+ resetApp() // resets all settings to their defaults
+ androidPermissions() // requests Android location/internet/notification permissions

Notification Service - takes care of Android notifications

Attributes:

- timer: time // time until next notification

Operations:

+ query(Coordinate coordinate) // pulls data from database
+ sendNotification() // initiates Android notification service request

Rutgercs CAS - Rutgers' authentication server

Operations:

- authorizeUser() // checks data against valid netid/password combinations

Manager Interface

Login - CAS authenticate user's information then log user in (For manager)

Attributes:

- netid: String // user input netid text
- password: String // user input netid password text
- firstLogin: boolean // tracks if app is launcher for the first time

Operations:

+ requestVerification(String password, String netid) // request authentication from CAS
+ accessDenied() // displays error message if CAS does not authenticate successfully

Rutgercs CAS - Rutgers' authentication server

Operations:

- authorizeUser() // checks data against valid netid/password combinations

Selection - allow manager to monitor the current status for each bus

Attributes:

-Requestscounter: display requests for each route & bus stops // pull from database

+getschedule: get the original schedule

-sendmessage: send message to bus drive (SMS)

Database

User - storage user data to identify users

Attributes:

+ NetID: int // unique id generated by rutgers student system

+ Password: String // authenticate user

- DeviceID: String // implementation of remember me

Operations:

+ saveNetID() // storage netID into JSON file

+ savePassword() // storage password into JSON file

- authenticate() // use to compare user's profile and controlling access

Bus - Used for collecting data of user's input

Attributes:

+ prefered_bus_route: String // storage user's desire

+ current_bus_stop // storage user's desire

+ destination_bus_stop // storage user's desire

Operations:

+ saveBusData() // storage data into JSON file

C. Traceability Matrix

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
Login		X						
Feedback			X		X			
AppInterface								
Selection				X				
Settings								
Notifications								
SQL Query			X		X			
KeyManage							X	
ManagerLogin							X	X
MessageInt								X
User	X	X	X					
Bus				X	X	X	X	X

Section 3. System Architecture and System Design

A. Architectural Styles

The main architectural style in this project is client-server model.

From the definition, the client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. In this project, The server receives user's bus request from a simple, easy-to-use mobile application. Then give the application its output. The client are the mobile application which design for user to request the bus and the web that let bus manager better manipulate the bus dispatch.

The whole system is easily manipulated with client-server model style since our target user and main function are clear. That is why client-server model is a good fit in this case.

B. Identifying Subsystems

The bus management system includes several subsystems, including mobile app, manager website, data collection and database.

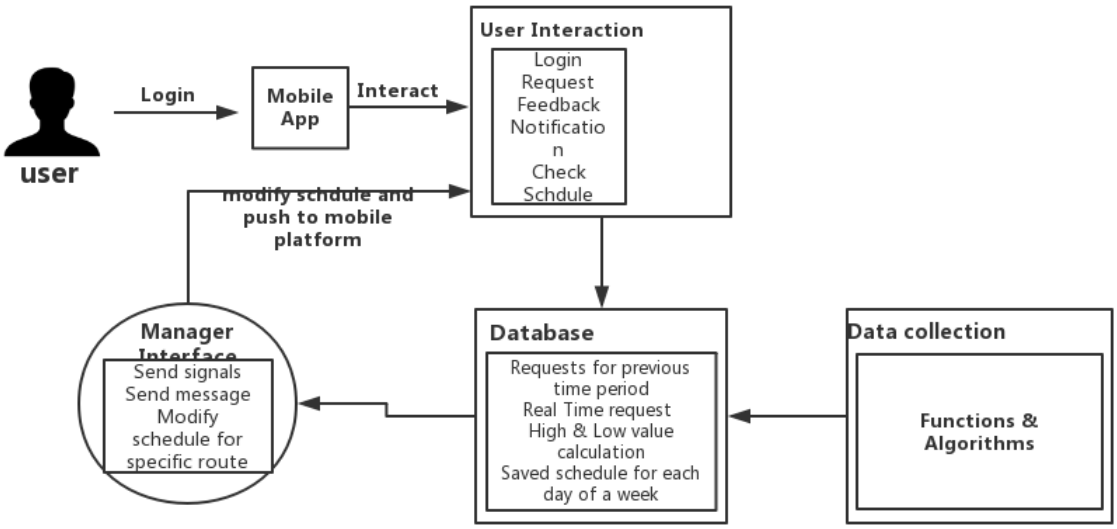
The mobile app will be the platform the users will interact with. When users approach the nearby bus-stop, they can request their preferred route. At the same time, the mobile app will collect the inputs and send feedback to manager website if any route reaches a peak value of request. The mobile app will try to optimize the static schedule for buses.

The manager website will be the platform for the managers to monitor the real time requests of each bus stops and routes. This can help manager to decide which route may need more dispatch.

The data collection system involves the number of requests services. This subsystem also connect to Database.

The database subsystem will be the place for saving data including the number of high , low and medium values of requests of each day and each week.

UML Diagram



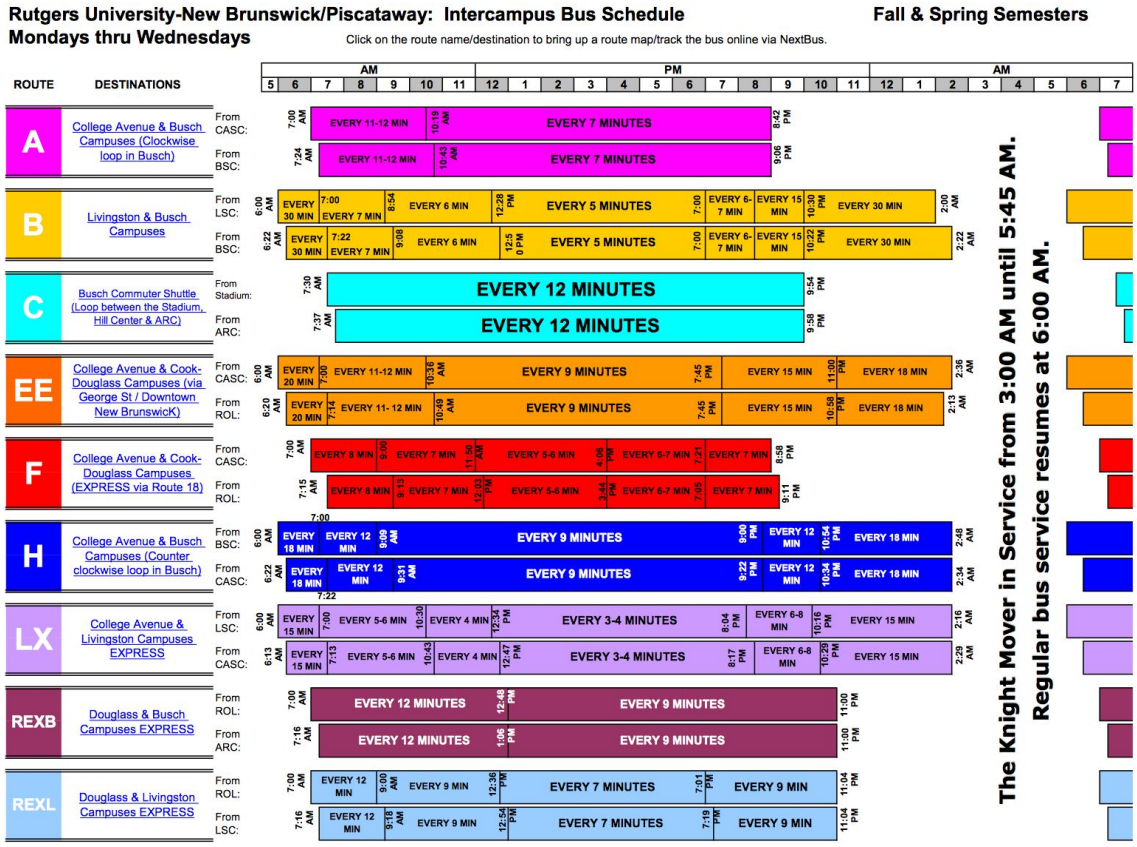
C. Mapping Subsystems to Hardware

We have three kind of subsystem to hardware:

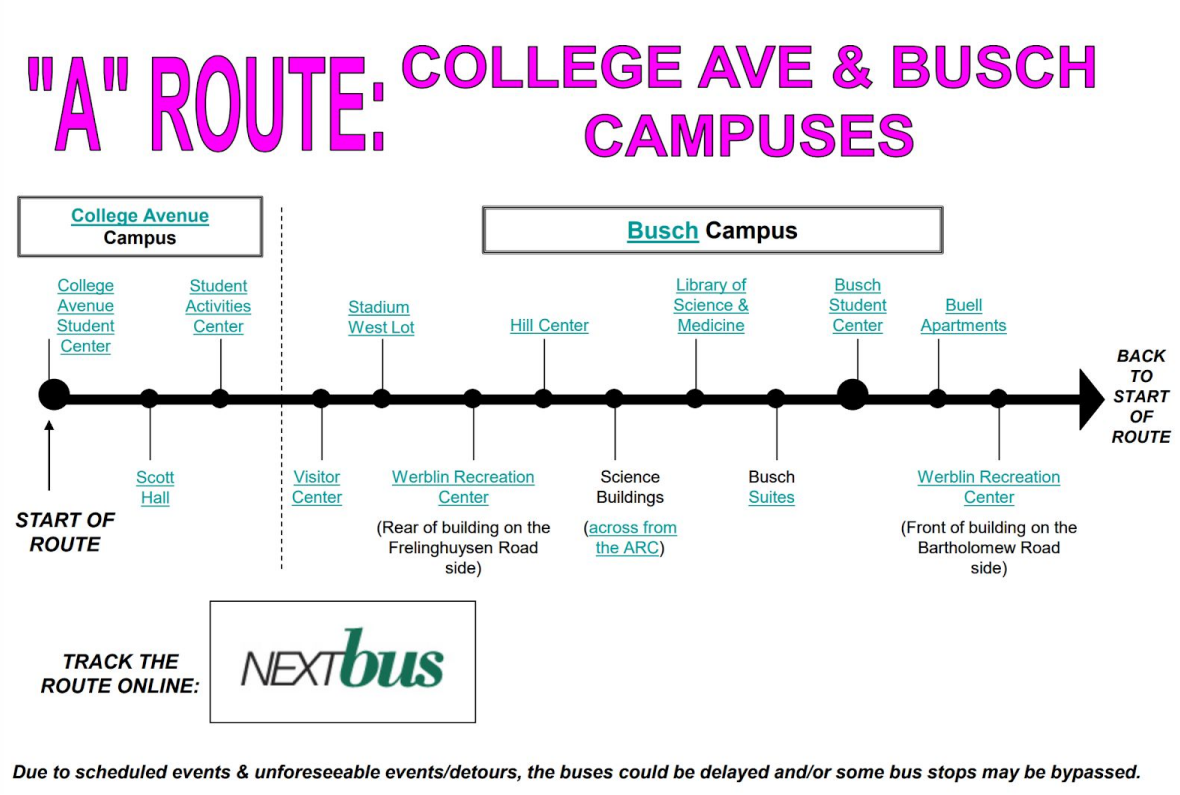
1. Android applications run on mobile device
2. Web clients run on web browser
3. Google Firebase service (cloud) use to storage data and implement algorithm

D. Persistent Data Storage

We will use the firebase (cloud) to store our data
Bus routes and Bus schedule are stored in a relational database



Courtesy of the Rutgers Transportation Department Website <http://rudots.rutgers.edu/campusbuses.shtml>



<http://rudots.rutgers.edu/PDF/2017/bus/arouteonlinedescription2017-18.pdf>

E. Network Protocol

HTTP JSON XMPP

F. Global Control Flow

Our system is a kind of procedure- driven. The system is procedure- driven in that users can use application any time to choose the buses they want to take. All demands and information users enter in application will immediately send to database and retrieved. On the other hand, our system is also event-driven. According to the database, manager will change the schedule every week.

Our system is a kind of procedure- response type. That means all of the information that users have and receive is real time.

Our system makes use of two threads. One is main thread is based on the UI and input from users. Another one is network thread that connect our application with service to share the information of database.

G. Hardware and Software Requirements

- For mobile application:

  User is required to have a smartphone with android 1.6 or iOS 6 system, at least 1GB storage space, inside GPS receivers and access to the Internet.

- For web management:

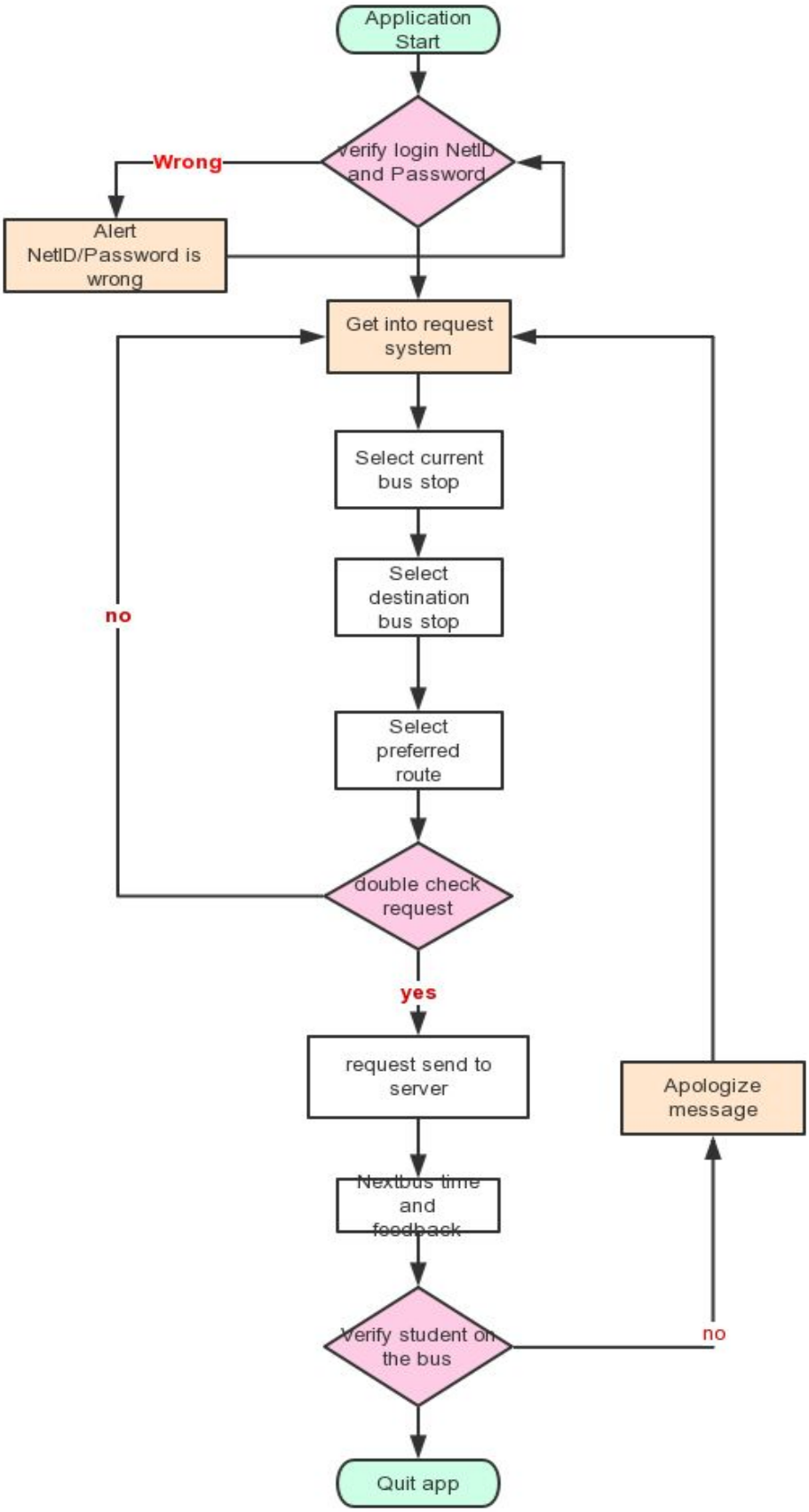
  User is required to have a computer that can run popular browsers such as Safari, Google Chrome and Firefox, with a screen resolution of at least 800 * 600, at least 512 MB memory space and 1GB disk space

Section 4. Algorithms and Data Structures

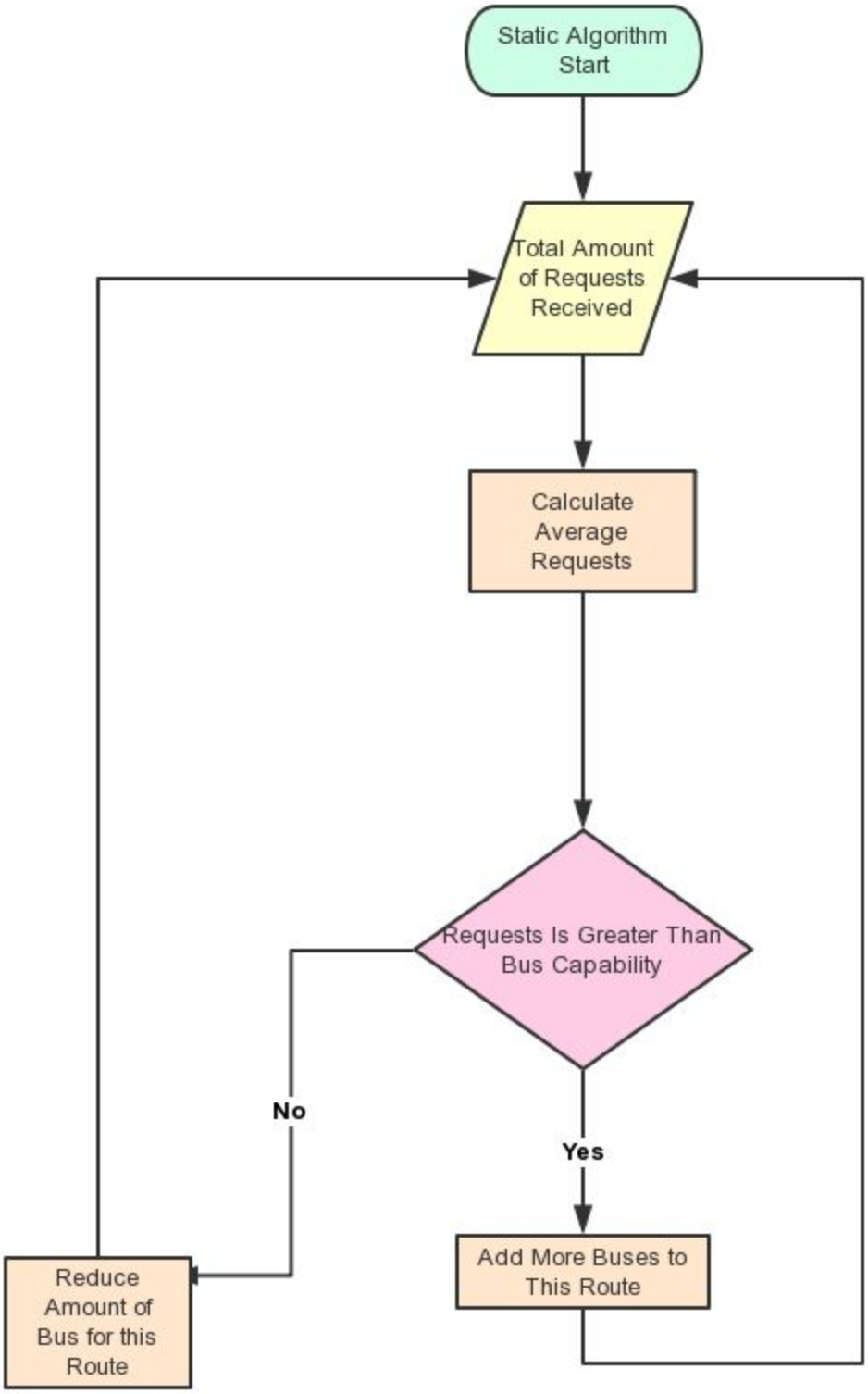
A. Mobile App Algorithm and Data Structures

The mobile app itself does not require extensive use of many data structures since much of the data is pulled in from the server where all the data manipulation is done, however basic arrays are used. In the Values XML for the app, the list of routes and destinations are stored as hard coded string arrays that can be universally accessed from any activity in the app. Transfer of data from the server when pulling may also be pushed in arrays, but a different method will most likely be used since only a small amount of information is needed per pull. Again, no complex algorithms are used in the app aside from some UI elements like filling list objects and requesting map data from Google Maps with an API key. An issue that may arise in the future will be whether each new user will need a unique Google Maps key, however, we will conduct small-scale testing first before proceeding.

The flow chart of general operations for the app can be seen below. Some portions may be bridged with the addition of a Hamburger menu for even more intuitive app navigation.

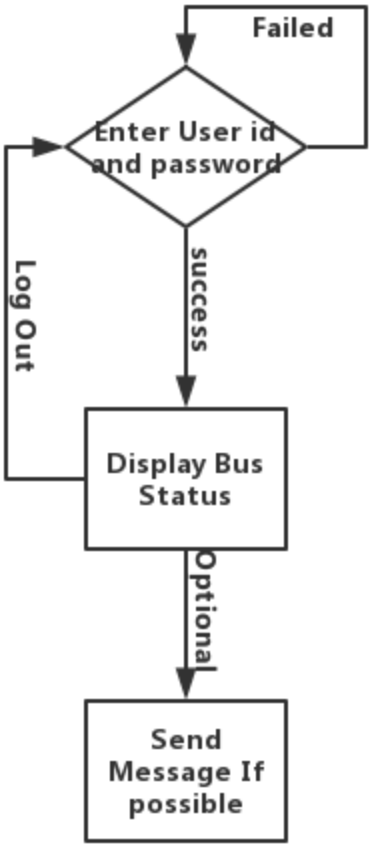

Application Start
Verify login NetID and Password
Wrong
Alert NetID/Password is wrong
Get into request system
Select current bus stop
Select destination bus stop
Select preferred route
double check request
no
yes
request send to server
Nextbus time and feedback
Verify student on the bus
no
Apologize message
Quit app

B. Server Side Algorithm and Data Structure



C. Manager's Web Algorithm and Data Structure

The manager interface mainly contains two parts: Login & Send messages and check the status of each bus. No complex data structures are used in manager web design, the status chart of buses are mainly come from the algorithm of mobile app development and shown in the manager interface. For data types, there will be integers, floats, strings,arrays and characters.



Part 1 Algorithms

The algorithms for static bus schedule are used to calculate the average requests period for a bus route in a a same time period in every week.

1. Calculating the total requests in every minute by adding requests in each station in a same bus route.
2. Calculating the average requests in a time period(three hours) in a week by adding the total requests in every 5 minutes and divide by the 36.
3. Calculating the average requests in a time period(three hours) in all weeks by adding the average requests in the time period in each week and divide by the number of weeks

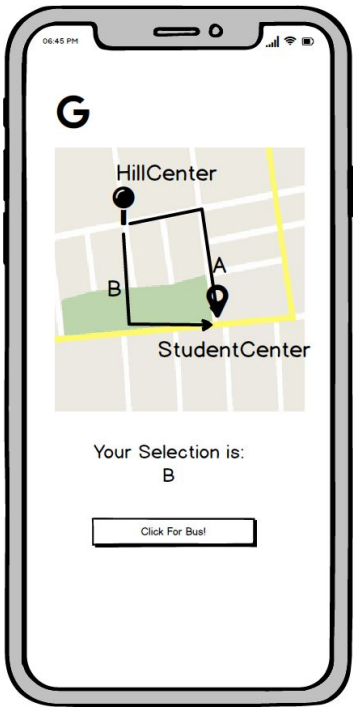
Part 2 Schedules

The average requests will determine the amount of bus for a route.

Section 5. User Interface Design and Implementation

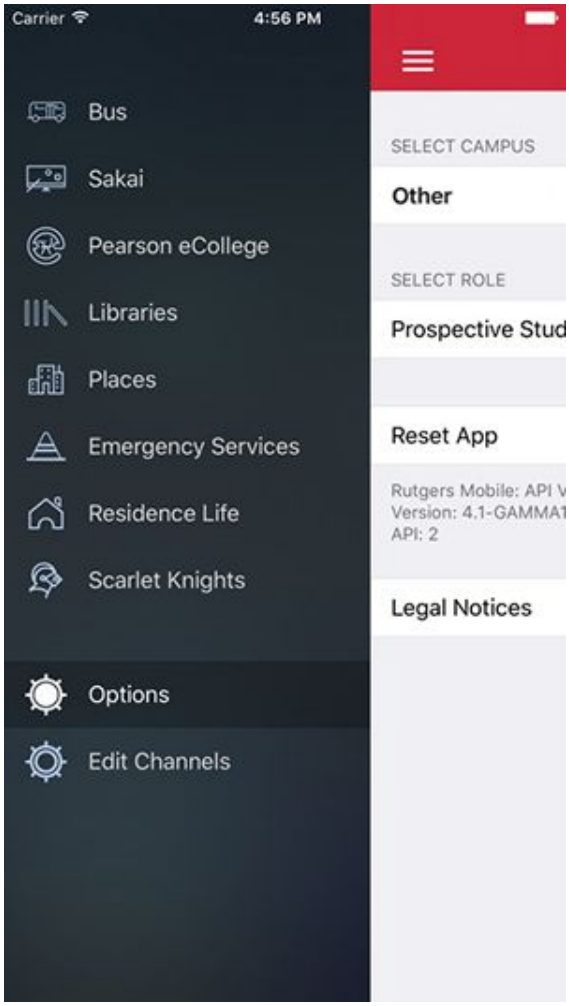
A. Mobile App Interface

Since the mobile Android app is being developed through Android Studio (now SDK 27 for Android version 8.1), some alterations were made to the UI design in order to implement a few of Google's official Material Design guidelines. A basic guide of this can be found at the URL [https://material.io/guidelines/]. This decision was made to streamline development since Android Studio employs these design cues by default and making an exact model of the original mockups would require the creation of custom assets. Other changes were due made due to differences between original design intentions and the functionality of the available development tools that are now at our disposable. Our preliminary design for the routing screen shown below was originally supposed to provide route selection by selecting various route outlines overlaid upon a map on a Google Maps MapView window.



Unfortunately, developing this specific feature is not possible with the default Google Maps API without creating separate helper tools from scratch, which would be both difficult and time consuming. Instead, we will show an outline of the currently selected route on the MapView window and a simple set of drop down menus will be allow users to input their desired route and stop. There will still also be an indicator for the current location of the user and the destination as well.

Another change that was made was for the placement of the settings activity. Initially, we were going to access the settings screen through a discrete button placed somewhere near the greeting screen, but this would mean a user would need to navigate back to the main screen whenever they wish to edit the app settings, which can be unintuitive. Following Google's general guidelines, we decided to join certain navigational aspects into a Hamburger menu (triple vertical bars) as seen below:

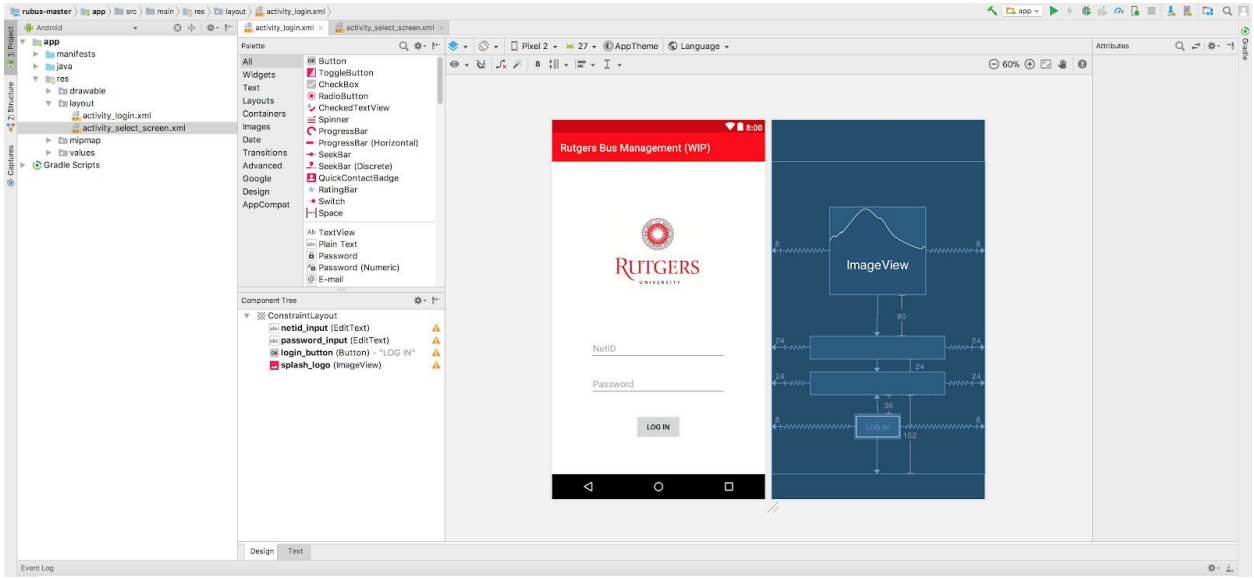


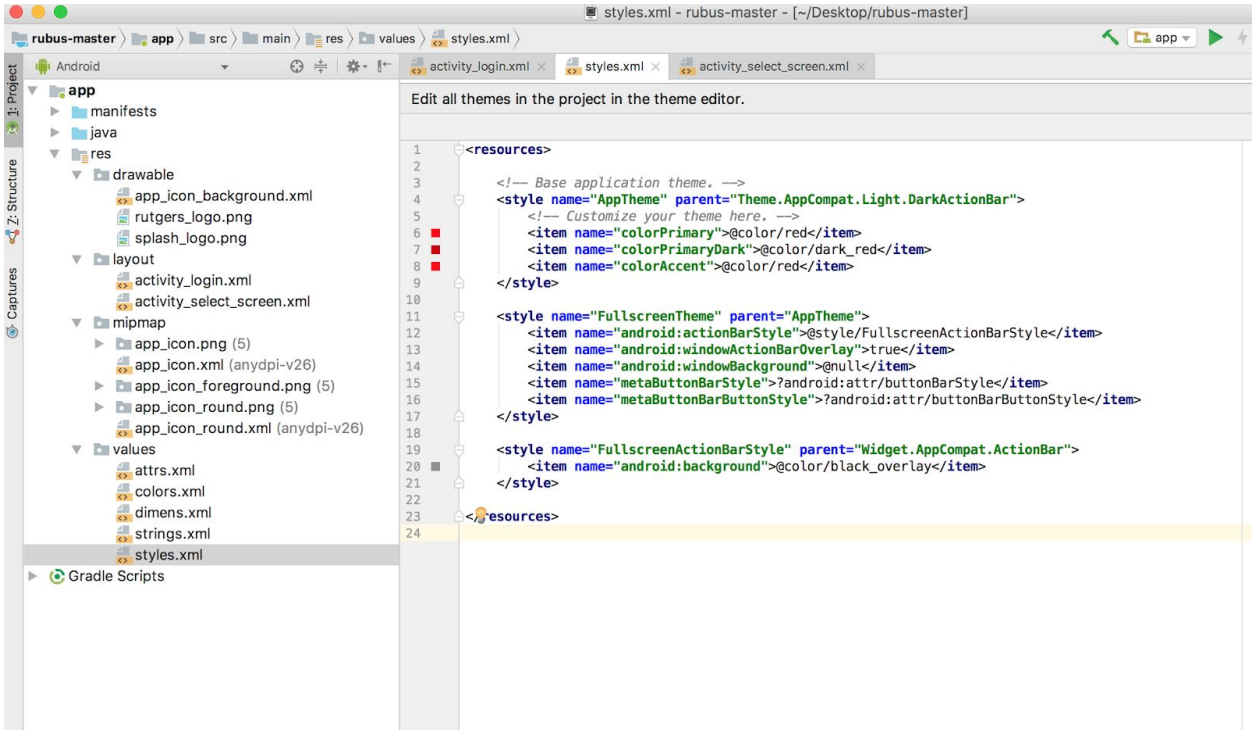
The current iteration of the Rutgers App utilizes a similar feature as well for the Android version. While we were planning to use a simple overflow dropdown menu, Google seems to have deprecated this specific feature in favor of the Hamburger. This would increase ease of use for the user since many of the screens would be universally accessible from any other screen and there is no need to incorporate reverse window navigation history.

Until we get the database and algorithm portions properly connected with the app, we are planning to use dummy data for login and authentication. Whether this will be implemented

before the first demo remains to be seen as it is probably one of the more complex portions of the entire project.

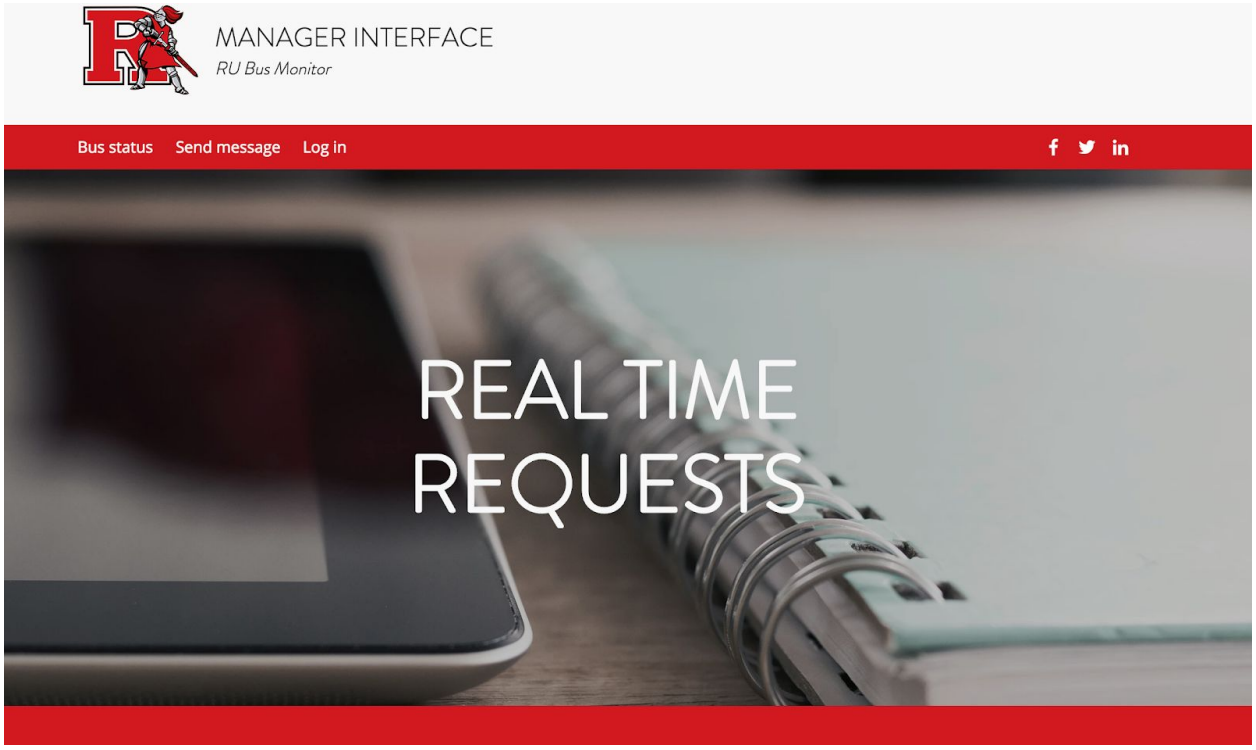
The only other changes were minor aesthetic changes to match Material Design. A use of a selected “colorPrimary”, “colorPrimaryDark”, and “colorAccent” for easy enabling of a universal app theme corresponded to typical Rutgers colors, as seen in the Styles XML.

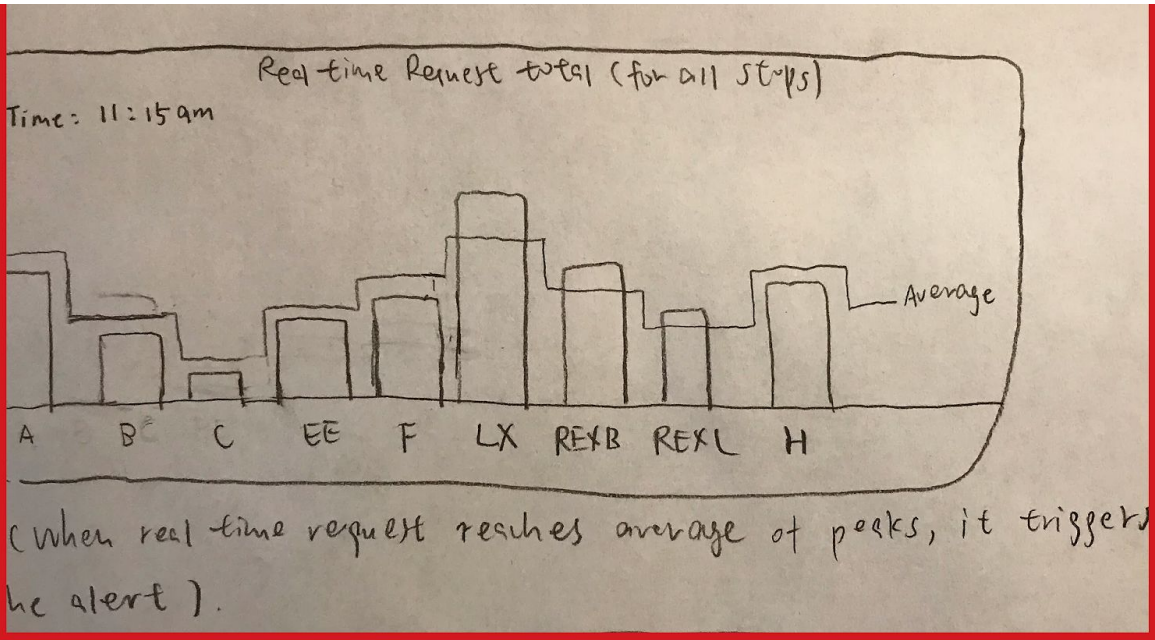




B. Manager Interface

The preferred design will be shown below:





REAL TIME REQUESTS

In this section, the real time requests for each routes and stops will be displayed. For each stop, the number of requests will be the total of each route.

SEND MESSAGE

Schedule & Operation Update

Name | Email

Subject

Message

Send

The elementary design for manager interfaces are shown above, including function of log in, check real time status of school bus and send message to bus driver. However, these functions are based on the successful design of mobile app and connect the data.

Section 6. Design of Tests

Integration Testing

There are variety of strategies to execute Integration testing, such as “bottom-up” integration, “top-down” integration and “big bang” integration. For our system, we will choose the “top-down” method. In top to down method, the testing will go through from the top to the bottom that following the control flow of the system. It will do good to develop and maintain the test on priority. For instance, if we get failed the test, our major flaws function could be found in the very beginning and we could check and fixed the issue through the environment. Therefore, the integration testing could help our system do it in simplicity and efficiency way, without the integrating testing the system testing long time.

Test Coverage:

The test coverage measure the effective way for the testing by providing data on different items. We will measure it by mapping the requirements and acceptance to our test cases. Some other tests will be create and modify in the future design.

Unit Testing :

Our unit testing will involve check through each of main classes and their modules separately.

Acceptance Test Cases

For acceptance tests, it formal testing with respect to user demands, requirement, and business processes to test the system whether satisfies those acceptance cases. We divide into two parts for the internal acceptance testing and external acceptance testing. For the bus managers, they will use the website for monitoring and dispatch school buses, hence we need to test the system is satisfies their goal or not. Furthermore, since students using the phone’s application to request the bus and get the time schedule, there is the most of test cases comes from. In order to to generate the integration testing, the acceptance test cases should form the user-like. In the lower part, we list our acceptance test cases.

Test Cases:

Test-case Identifier:	TC-2
Use Case Tested:	UC-2, main success scenario
Pass/fail Criteria:	The test passes if user enter the correct information and then successfully logged into the system.
Input Data:	Number, letter and punctuation
Test Procedure:	**Expected Result:**
Step 1. Open the App Step 2. Try to enter the correct information and see if system allow to jump to bus request page	App successfully runs and ask user for their information. System check the user's information and jump to bus request page if their input is correct or ask them to type in again if their input is wrong.

Test-case Identifier:	TC-3
Use Case Tested:	UC-3 and UC-4
Pass/fail Criteria:	The test passes if the user enter bus routes and bus stop that are contained in the database, and then the page will show the estimated time of arrival.
Input Data:	Numeric code,Letter
Test Procedure:	**Expected Result:**
Step1. Choose the incorrect Bus line and correct Bus stop Step2. Choose the correct Bus line and Bus stop	System will stay in the UC-3 page until user get correct information. Prompt the user to try again; System received the information and then jump to the UC-4 to indicate success; display the estimated time of arrival

Test-case Identifier:	TC-5

Use Case Tested:	UC-5, main success scenario
Pass/fail Criteria:	The test passes if users have no response or response “Yes”. The test fails if users response “No”
Input Data:	letter
Test Procedure:	**Expected Result:**
Step 1. Open the App Step 2. Click “Yes” or “No” button	App successfully runs and ask if users are on the bus or not. CheckDevice determines the next step. If the response is “Yes” or no response, the request is completed and deactivate the App for preventing abuse. If the response is “No”, the request will be resent and it will repeat the UC4.

Test-case Identifier:	TC-6
Use Case Tested:	UC-7 and UC-8
Pass/fail Criteria:	The test passes if users can successfully log in and sending message.
Input Data:	numbers,letters
Test Procedure:	**Expected Result:**
Step 1. Open the App Step 2. Click “Yes” or “No” button	Users can log in with their ID and password successfully. Other wise, the web should stay at the current page and notify users to input the correct data. For sending message, if it is passed successfully, we can get notified a new incoming mail in the testing email address.

Section 7. Project Management and Plan of Work

A. Merging the Contributions from Individual Team Members

All Team member have equal overall contribution currently.
Every team member contributed with quality works and debugged their work together by voice meeting.

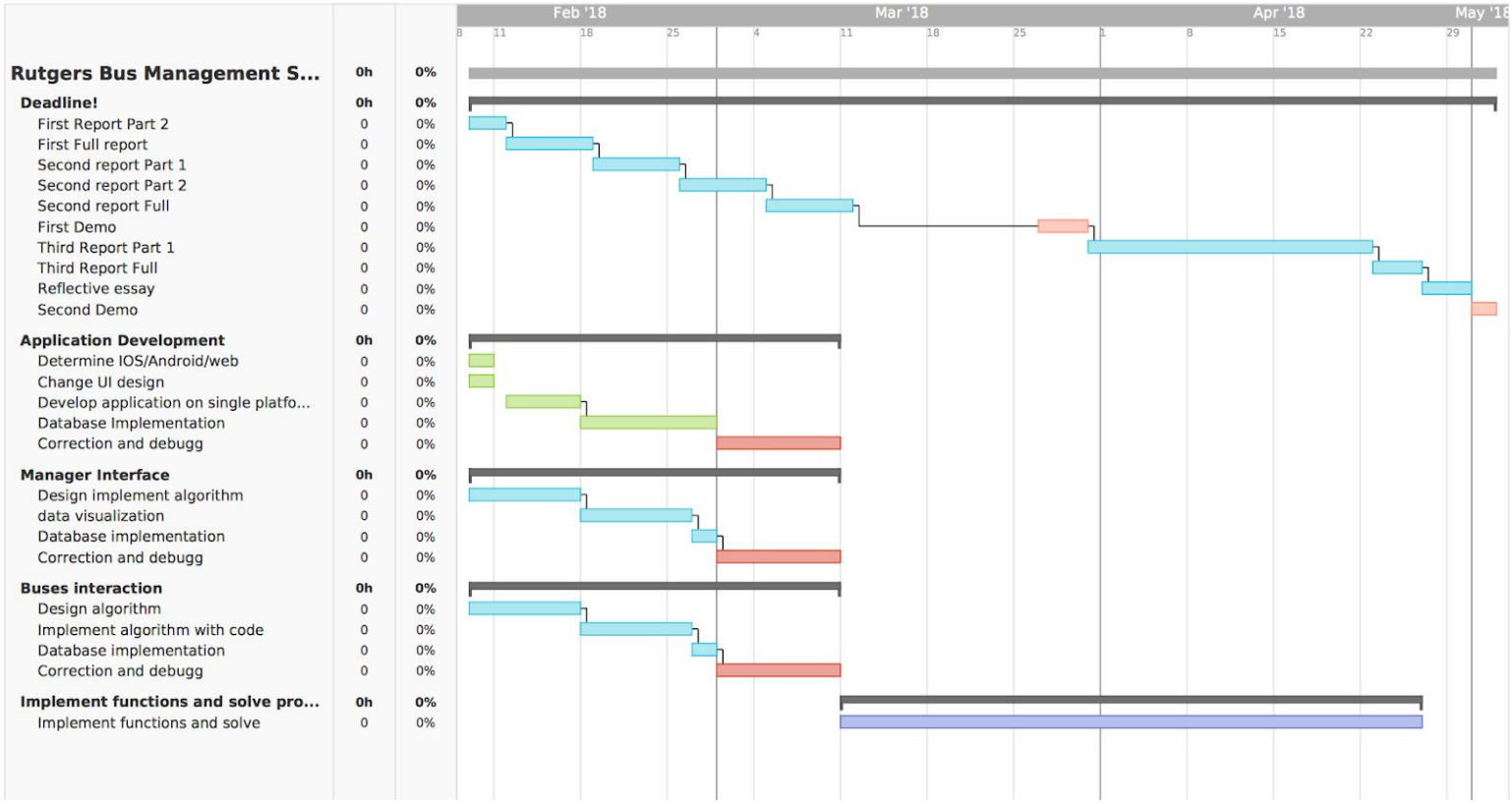
Problems	Solution
Procrastination	Changed the working due date of report on Saturday night and schedule voice meeting to debug our report on Sunday
Lack of communication (We have 75% native Chinese speaker; language issues)	It is required to discuss project in English and we work more hard on grammar correction of report.
Lack of Cooperation	Using Google doc for writing documents together and screen sharing with TeamViewer.
One group member dropped class and abandoned his work in the middle of semester	Keep doing project, not a big deal

B. Project Coordination and Progress Report

User cases 1-6 are the cases are related to scheduling buses and User cases 7,8 are related to bus manager. Our project currently have not yet finish developing applications and we will implement most of user cases in our programming projects. We are facing issues of connect Firebase service and NextBus API implementation.

C. Plan of Work

Gantt Chart



Breakdown of responsibilities is at the start of report.

Current Develop progress

1.Android Development:

Thus far, the UI components have been set and basic menu navigation has been implemented in Android Studio using the latest SDK 25 (Android 7.1) for maximum compatibility with both new and old devices. The design and placement of navigational items are being arranged based on the Google Pixel 2 cell phone. With a 5 inch screen sporting a 2880x1440 resolution (538 ppi density) at an 18:9 aspect ratio, it strikes a good balance between many of the popular Android phones out in the wild at the moment. Additionally, Android will natively scale up or down to whatever resolution and screen ratio is required due to the planned use of HDPI assets.

We are still in the process of researching the most elegant method of integrating Google's Firebase API into our app. It seems relatively simple at first since Firebase is integrated into the Android IDE, but we will require further communication with the backend development team to determine how they will be implementing their code and the best way to configure our side to work with it.

2.Manager Web Development:

As far as the development of manager's web interface goes, we have laid out a general plan of which features will be included in the web interface, which APIs will be required and the general layout of the interface.

Our interface will include the following:

- A chart with bus request statistics on it
- A map with the locations of bus stops and real-time locations of the buses
- An interface to send a text message to drivers
- A table of the bus numbers with corresponding routes and bus drivers

We will require the following APIs:

- Google Maps API
- NextBus Public API
- Sugabas API

Thus, we have finished with the process of figuring out what data will be required from which sources. The next step is to start the implementation process. The website will also based on the data shown in mobile app, and update the status simultaneously.

3.Backend Development:

Our backend developments are separated into two regions. One is static bus distribution schedule and other one is real-time dispatching controls.
The static schedule is a planned weekly schedule. It will be adjusted by the algorithms according to the requests which are collected before.

The first algorithm: For number of users
We assume that 5% of students use our app in the first week and the number of users increased by 5% each week. At last, we believe the number of users will be stable at 80%.

The second algorithm: For static schedule
We will average the requests for each bus in the certain time period in the each day of the week. For example, we collect 60 requests in the Monday 7am-11am in the first week and 50 requests inthe Monday 7am-11am in the second week for Bus C. Then we will add 50 by 60 and divide their sum 110 by 2. We get 55 for the average requests. Once we have large samples and data, we can adjust the bus static schedules according to the average request, because the average should represent the regular bus request and the relative bus schedule can solve most of lacking issues.

The third algorithm: For real-time dispatch and revising the static schedule
We will calculate the total requests of each bus route.
We firstly add all the requests from the stations of a same bus route and we get a total requests. Once, the bus arrive a station, the request on this station will automatically reset to zero, and the new total requests = original total requests - the requests of the arriving station.
After the bus passes the station, we will receive the new numbers of request in this station(including the resended requests which are sent by the users who do not get on the bus and new requests) and add it to the new total requests.
We will set a warming line, once the total requests exceed the warming line for a certain time, let’s assume about an hour. In this case, we will consider to dispatch a bus from another route whose total demands are lower than the average requests to relieve the pressure of demands.
The database will collect this case and it can improve the static schedule for next week at the same time.

D. Breakdown of Responsibilities

Android Development: Aaron Hu, Haowei Li

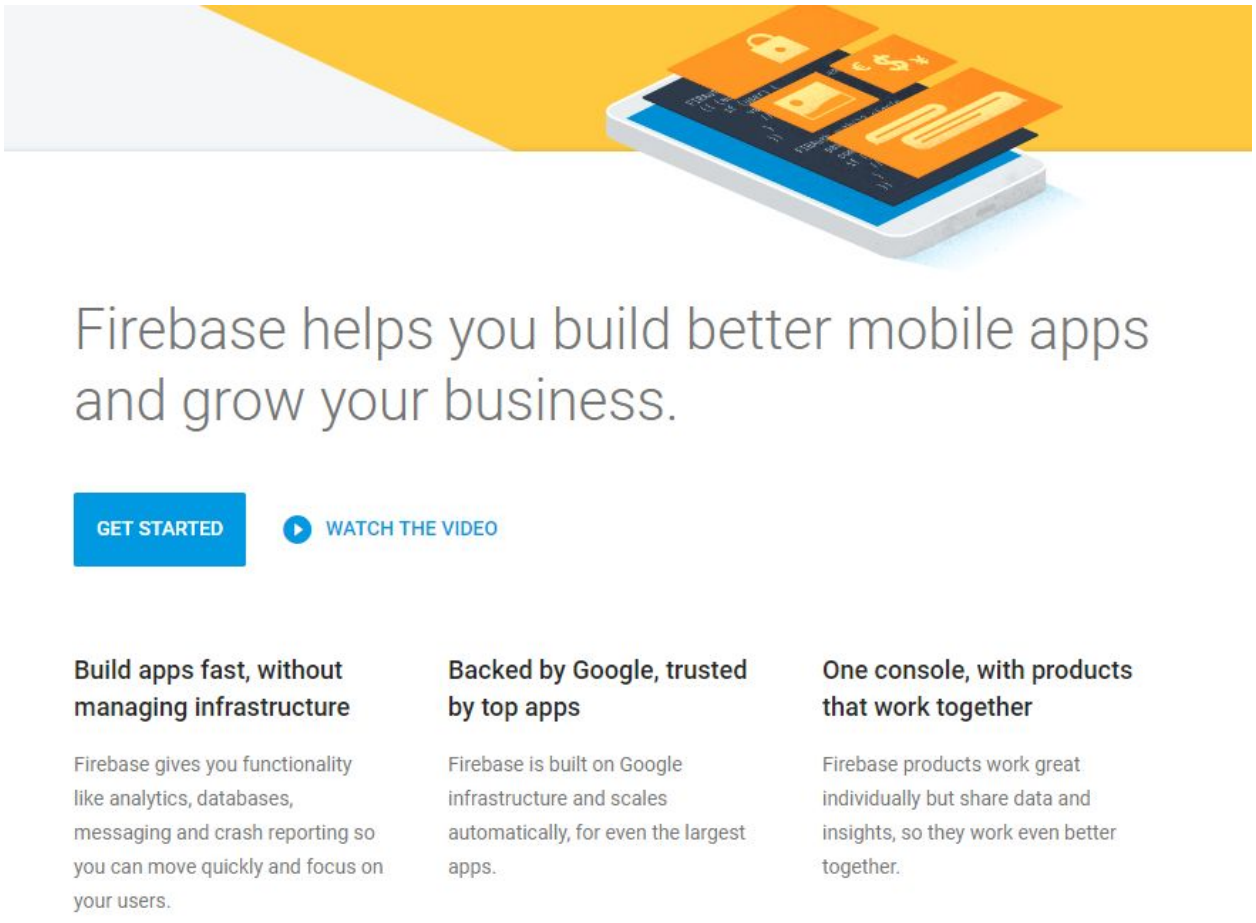
Manager Web Development: Viraj Patel, Yufeng Lin

Backend Development (Bus interaction): Yaocheng Tong, Lu Jin, Minghao Qin,Yuhai Zhang

E. Third-Party Platform

Our server will be heavily based on Google's cloud service called "Firebase".
<https://firebase.google.com/>

We can storage application data into the firebase in real-time.



Instruction Video Link:
https://www.youtube.com/watch?list=PLl-K7zZEsYLmOF_07IayrTntevxtbUxDL&time_continue=68&v=iosNuIdQoy8

References

1. Marsic, Ivan. 2012. *Software Engineering*.
2. *Google Maps*. Feb. 4, 2018. Retrieved from: https://www.google.com/maps
3. *Rutgers University-New Brunswick/Piscataway: Intercampus Bus Schedule.* Feb. 4, 2018. Retrieved from: rudots.rutgers.edu/campusbuses.shtml
4. *Nextbus API*. Feb. 4, 2018. Retrieved from: http://api.rutgers.edu
5. *Rutgers Campus Buses.* Feb. 5, 2018. Retrieved from: https://en.wikipedia.org/wiki/Rutgers_Campus_Buses
6. *Rutgers Central Authentication Service.* Mar 4, 2018. Retrieved from: https://idms.rutgers.edu/cas/how_does_it_work.shtml