Software Engineering

RU Bus Management System

Report 3

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

2018 Spring

Table of Content

Individual Contributions Breakdown

Every team member contributed equally.

Summary of Changes

Project Objectives

Customer Statement of Requirements - Problem Diagnosis:
The system was designed to actively find extra buses and add them to the required route whenever receive the requests but since the amount of buses are limited, we modify this feature by not allowing it look for a bus but we would like to let it compare the time between the next coming bus and the time might take to add extra bus in this route.
Proposed Solution: since we cannot get real-time traffic condition, we remove “real-time traffic visualization for users” and “real-time traffic notifications”

Glossary of Terms:
Mobile Application: software on Android OS for user to request buses. User cannot actually have the access to bus management system.

Requirement Specification

Actors and Goals:
User application only forward requests to database not the dispatch center. We did not know how we were going to connect them and now we have figured them out after the first demo.

Use Cases:
UC-5: Actor goal: after demo 1, we decided to make the use of feedback and help us create the future bus dispatch schedule. When there are lots of students cannot get on the bus, that means we may increase dispatch rate during that period. When all of them can get on the bus, that means at that period, that route might be eligible to have lower dispatch rate.
UC-6: GetNotification: Since it is hard to get real-time traffic information, we decide to delete this use case.

System Design

Enumerated Functional Requirements
ST-2
We originally allow passengers to select two routes.
Due to our database, we have to allow each passenger choose one route in one request.

ST-5
We originally will notify passengers about their buses arriving time.
Because we restrict them to select only one route, we will only send the notification about the selected route.
ST-19
This ST is similar to ST-15.
We replace it with authority of manager to access the static bus schedules which produced by our algorithm.
Enumerated Non-Functional Requirements
ST-24
We allow passengers to report in the feedback if there is emergency happened.
On-Screen Appearance Requirements
ST-36
We allow manager to check the demands of different routes.

Effort Estimation

Student's Mobile Application
1.Authentication
Add save for Rutgers NetID and password

Domain analysis

Domain Model
UC-1
Add: displaying information about selecting route.
UC-4
Add: displaying dispatch rate
UC-6
Add: Timer for arriving time and dispatching rate
Domain Traceability Matrix
UC-1
Add: MapService
UC-4
Add: MapService
UC-5
Delete: MapSerice

Class Diagram and Interface Specification

Updated UML diagram of our existing android software.
Updated UML diagram of each module in android application.

Section 1. Customer Statement of Requirements (CSR)

Background

Rutgers is becoming more diverse because there are many new students matriculating each year. At the same time, the school bus requirements are facing challenges due to high pedestrian traffic. In the past few years, Rutgers has been vulnerable to a cyber attack due to a poor implementation of the bus scheduling system. While it has been worked on, it can still be improved to increase reliability, functionality, and user friendliness. The core of the issue is that the current transportation system is unable to account for external factors such as traffic during rush hour time slots between popular class periods at various hotspots like the student centers, Scott Hall, and Hill Center. Therefore. Therefore, having a improved version of school bus manage system seems to be necessary and critical.

Problem diagnosis

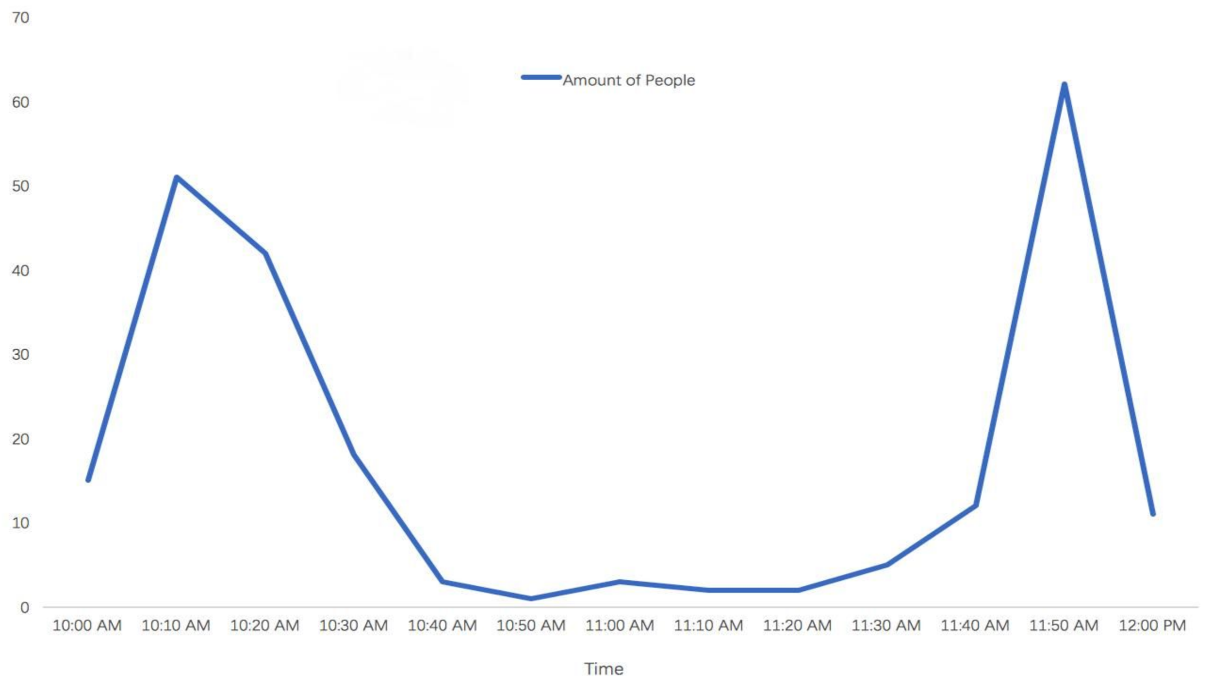
The current school transportation system takes quite a bit of work for the school to manage and the university needs to put forth more effort in order to improve the flow of bus operations. We could take surveys of Rutgers students to identify and analyze the current problems confronting us or garner some suggestions for improvements to the system.

Many of the students use the MyRutgers mobile app to check when the next bus is arriving, but due to some specific time slots such as noon, evening, after-class, and exam periods, many popular bus stops such as Student Activity Center at College Avenue Campus, Hill Center at Busch campus, and Livingston Plaza at Livingston campus, face unexpected pedestrian traffic. As a result, some students are unable to get on the buses on time even after waiting for a long time. This scenario shows a relatively commonplace issue that may need a better solution to resolve. Therefore, this app is our plan to provide students with a better platform to reflect their needs. How can the data reflect the level of needs? If there is a student arriving at the Hill center and taking "H" bus back to College Avenue Campus, he/she can click the "H" option below "Hill center" and the app will tell students the fastest way to get there (either waiting for the bus based on the current schedule or add extra buses to this route). A student may only click once for their preferred route within a 20 minute timespan. When the number of requirements reaches a peak, it can be determined as a urgent and the system will send notifications to the bus drivers and reduce the time gap between two successive buses. Also, students can leave comments on the app and the app will collect data then send it to the bus management office for improvement.

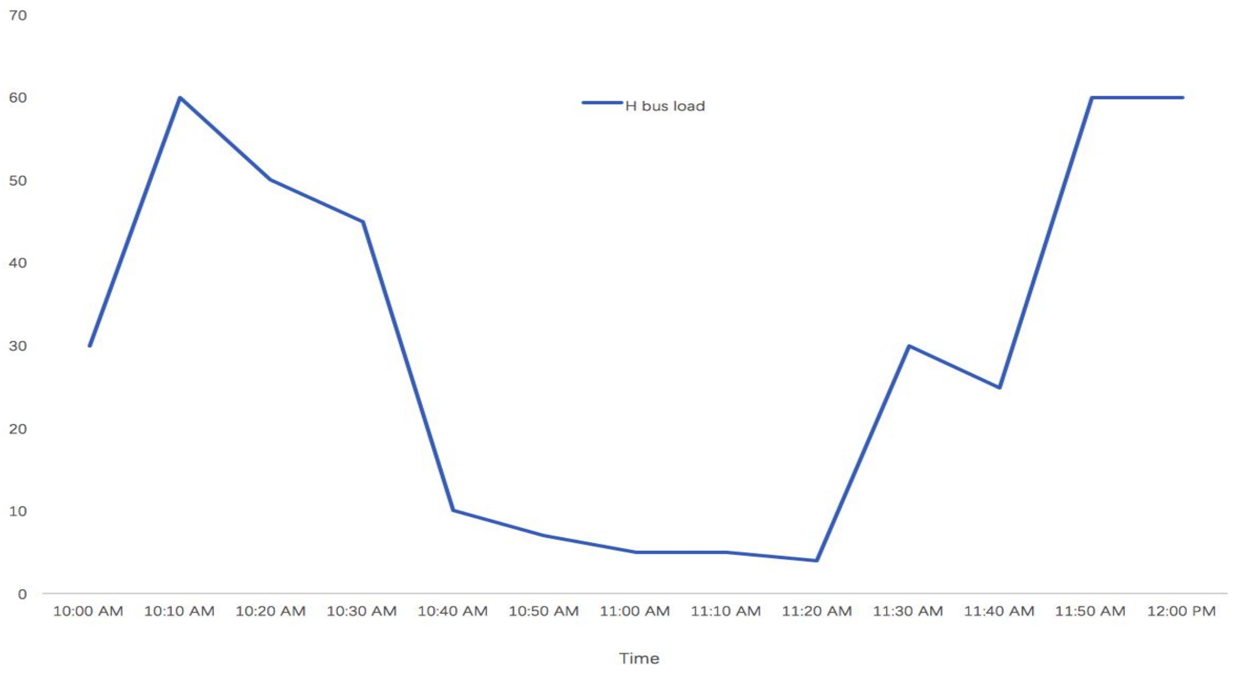
From our observations, there are two main issues with the current system and several common scenarios exist that justify the need for a new school bus management system. First, there is the problem of lack of peak-hours transportation capability. Rutgers has a massive student body and at a specific time, the bus stops will have hundreds of student waiting for campus buses. Since the buses will be overcrowded, a huge portion of waiting students can not board upon arrival. Those students who are left behind at the stop have only left the options of waiting for the next bus, which could take a significant amount of time depending on other factors. There are even situations in which students may miss multiple consecutive buses due to the increasing crowding at stops. This results in many students being tardy for class and a cyclical degradation of bussing services until the crowds are slowly dispersed over time.

During the peak hours, buses are usually loaded to capacity with as many students as they can hold, which can bring up bus overloading and safety concerns. Whenever such a situation occurs, not a single student can get on or off the bus easily, further increasing delays. Another issue to a lesser degree, the bus will be much heavier than usual and consequently have reduced control, resulting in increased chances of a traffic accident. To observe these relatively common scenario, we have recorded some pertinent data and plotted it in the following charts.

The number of people waiting at the Hill Center bus stop on the Busch Campus between 10 AM to 12 PM, a common rush hour time slot. (Data recorded on January 17th, 2018)



The number of passengers on the H-bus spotted at the Hill Center bus station between 10 AM to 12 PM (rough estimation). (Data recorded on January 17th, 2018)



Parsing through these two data sets, we can see that between 10:10 AM and 11:50 AM, the buses are unable to handle the large influx of bystanders. Students remained at the bus stop and continued to wait for the next bus causing possible lateness to their classes. The H-Buses were loaded to capacity with passengers between 10:10 AM and 11:50 AM. Some relevant information on the vehicle used for the H route from the manufacturer's website:

H-Bus type: ENC AXESS
ENC AXESS specifications:

TECHNICAL SPECIFICATIONS	35′ MODEL	40′ MODEL
GVWR	43,420 lbs.*	43,420 lbs.*
Body Length	35′ 11.5″	40′ 11.5″
Body Width	102″	102″
Wheelbase	215″	275″
Seating	Up to 35	Up to 43

The currently used model is designed for 35 seats, but during peak hours it usually loaded up to 50-60 passengers. As long as it loaded to such numbers, the bus will be much heavier and more difficult to handle. Due to this, the H-bus will have higher risks of traffic accidents on the way to College Avenue.

Another issue found within the current system is an inefficient use of resources during off-hours (i.e. not between class scheduling blocks). By controlling the intervals of the arrival times of two adjacents buses and reducing their variation, the peak load during rush hour can be significantly reduced while off-hour transport would see minimal impact. During normal class times, Rutgers bus stops have many less students waiting because most students are in class. However, the frequency of bus routes are exactly the same as during peak-hours, so generally only a few passengers are transported per bus (sometimes not even a single student at all). Looking back at the previously mentioned data sets, between 10AM and 12PM, Hill center has an H-bus arriving every 9 minutes, but between 10:40 AM and 11:20 AM only a few passengers are aboard.

Another scenario in which the inefficiency of the current system is evident. When a bus is broken down on the road (some emergency issue) or cancellation of bus service. In both cases, there will be a large gap in service that cannot be filled due to the inflexibility of the current implementation. Students who are waiting at the bus stop for a long time have method of being notified about current conditions other than an estimated arrival time, leading to an understandably frustrating and unintuitive setup. Furthermore, when bus service are up and running again, the scheduling will not be in sync with what it was before the disruption in service, causing further issues later on.

The primary function of the bus system improvements should allow students and the bus control center that monitors bus activity to account for external factors and maintain flexibility in such cases in order to reduce the waiting time and reduce service disruption. It would also allocate resources more evenly during peak hours to increase efficiency without increasing the number of total routes per business day.

Proposed solution

We are planning to tackle the issue of data collection head on by collecting rough pedestrian traffic numbers at various points in time at hotspots like Hill Center. By plotting the data, we can see the general trend of crowding throughout the day and consequently the time periods in which buses routes are required at shorter intervals. Section 3 will go over how this will be achieved in greater detail. The below figure shows a simplified visualisation of the I/O workflow of the proposed software.

Benchmark goals by which we will be judging the functionality of our software will include:

- Ease wait time for bystanders at bus stops
- Reduce wait time between bus arrivals during peak hours
- A method of bus dispatch depending on transportation needs
- A scheduling system in which bus drivers will know when peak hours are occuring

Value

Successful implementation of the before-mentioned system will result in more efficient use of the currently allocated resources with reduced overhead. Buses will be running more frequently during busy periods and less frequently during off hours. As a result, students can plan their daily schedules with more leeway and increase productivity as well as academic performance.

Section 2. Glossary of Terms

- *Application(APP)* - main program that the user will be interacting with; the program will be designed to meet the needs of the many students at Rutgers
- *Schedule* - buses will have a separate schedule for regular class days, weekends, events, and holidays. Schedule includes dispatch time, dispatch location and corresponding bus route
- *Bus driver* - person driving the bus, taking directions from the manager/administrator
- *Manager/administrator* - person(s) who will manage and coordinate the bus drivers and the bus allocations for each route
- *Student* - undergraduate or graduate students who will take Rutgers buses
- *Database* - server that will contain user data, bus routes, peak-hours
- *Mobile application* - software on Android OS that allow users to request buses.
- *Mobile application user* - someone who is using the mobile app
- *Peak-hours* - times of the day when more traffic is expected on the road
- *Off-hours* - times of the day when less traffic is expected on the road
- *GPS(Tracking Service)* - (Global Positioning System) devices/services to track the current location
- *API* - application programming interface
- *NextBus API* - API that can output tracking data of Rutgers buses
- *Firebase* - Google cloud database and calculation services for applications
- *Mobile Interface* - interface that can interact with mobile users
- *Web Interface* - an interface that can show real time status to manager (auxiliary tool for mobile app), enable manager to visualize the condition of each bus route.

Section 3. System Requirements

A. Enumerated Functional Requirements

Identifier	Manager User Story	Size
ST-1	As a passenger, I can log into the application with my NetID and Password.	2 pts
ST-2	As a passenger, I can find all types of campus buses in menu selection. I can select one route in each request in the application.	3 pts
ST-3	As a passenger, I will be asked by application about whether I allow application to find my current location or not.	3 pts
ST-4	As a passenger, my current bus stop will be automatically notified if I agree application can access my current location data. I can select my current bus stop if the application made mistake on my current bus stop.	6 pts
ST-5	As a passenger, I can be notified next bus arriving time after I selected route which I want to take.	4 pts
ST-6	As a passenger, I can give feedback on selecting if I am successfully get on the bus on time.	2 pts
ST-7	As a bus driver, I am able to receive alerts on what time I should begin my route according to a preset schedule	5 pts
ST-8	As a bus driver, I am able to find the fastest way when I receive alerts.	2 pts

ST-9	As a bus driver, I am able to receive alerts on which campus I should begin a route on.	2 pts
ST-10	As a bus driver, I am able to send notifications if I can not finish schedule and managers can reschedule the driver.	4 pts
ST-11	As a bus driver, I am able to receive messages from the manager.	3 pts
ST-12	As a manager, I can check the real time status of each bus route.	6 pts
ST-13	As a manager, I may know which bus stop may need extra buses when observing the real time data	4 pts
ST-14	As a manager, I will be able to notify bus drivers using hardware	1 pts
ST-15	As a manager, I can post some announcements on the web if necessary	3 pts
ST-16	As a bus driver, I am able to access the future static bus schedules which produced by the algorithms.	2 pts
ST-17	Aa a manager, I can use my credentials to log in to the web interface to access the bus management system.	2 pts

B. Enumerated Nonfunctional Requirements

Identifier	Manager User Story	Size
ST-21	As a passenger, I can change my password by clicking the link in login page to transfer to Rutgers NetID management website.	1 pts

ST-22	As a passenger, I can select my destination of prefered bus route.	2 pts
ST-23	As a passenger, I want to be able to have contact information of Rutgers police to keep the safety.	0.5 pts
ST-24	As a passenger, I can give feedback of whether bus is crowded or not. I can report also report if there is any emergency happened.	0.5 pts
ST-25	As a passenger, I need announcements when buses routes are being changed.	1 pts
ST-26	As a passenger, I want to be able to see some announcements (road closed, traffic accident, holidays etc), since I could know any emergency issue and aware any maintenance around Rutgers Campus, so I can arrange my schedule.	4 pts
ST-27	As a passenger, I can find the current bus dispatch rate.	0.5 pts
ST-28	As a passenger, I can uninstall the app if I want.	0.5 pts
ST-29	As a bus driver, I am able to know some special activities according to a schedule.	2 pts
ST-30	As a bus driver, I am able to get extract paid if I need to work longer because of the special activities.	1 pts
ST-31	As a bus driver, I am able to schedule the time that I can work for special activities.	2 pts
ST-32	As a manager, I know how to communicate with all the bus drivers	1.5 pts

C. On-Screen Appearance Requirements

Identifier	Manager User Story	Size
ST-33	As a passenger, I can access rutgers campus map data to check, so that I will have a clearer overview of rutgers building.	1 pts
ST-34	As a passenger, I can use application interface to check information I need.	4 pts.
ST-35	As a manager, I can see the real-time requests from different bus stops and demands of different route.	4 pts
ST-36	As a manager, I can monitor the traffic and thereby schedule buses accordingly (Data Visualization)	2 pts

D. FURS Table

Functionality	Use to record and analyze student demand in order to arrange buses. All users will use this system everyday. Well security by Rutgers log in system
Usability	Clear Map instruction inside the applications and very easy to use,click,send request,check information they need.
Reliability	Application used by student who need bus to take them to the class, manage need to check the current bus demand and driver need instructions of schedule. Every operation process is very simple for system users and cloud-based database management can ensure important data will be used and not leaked.
Performance	Very quick responses of real-time demand from student.Clear bus schedule will be created to driver. NextBus API will used to track buses which will be very useful.
Supportability	Only support Android device right now, web application only for manager, and driver only have a sheet of paper which has their schedule on it. Cloud-based server side can created our own API for future uses.

Section 4. Functional Requirements Specification

A. Stakeholders

- Students
- Bus drivers
- Managers

B. Actors and Goals

- Students
  - Initiating Actor
  - Students have the ability to access the mobile Application that is designed for authenticated users. They may have a better chance of getting on their bus earlier.
- Bus driver
  - Participating Actor
  - The goal of bus driver is to follow the schedule from the Application that has been designed by a manager and provides feedback in terms of how useful and efficient the current system is.
- User Application
  - Participating Actor
  - The goal of User Application is to receive the requests from the user and forward them to database. The App is designed for optimizing the dispatch rate and dynamically rescheduling the current bus route.
- Manager Application (Web interface)
  - Participating Actor
  - The goal of the Manager Application is to tell both the manager and bus driver when and where extra buses might be required and suggests several options.
- Database
  - Participating Actor
  - The goal of Database is to tell the manager the most likely predictions and decisions based on the large amount of crowdsourced information (bus routes, time estimates, number of students at each bus stops, requirements for each bus routes, etc.) stored in the database. It will also store the previous user demand data from user Application as well as feedback from users.
- Mapping Service
  - Participating Actor

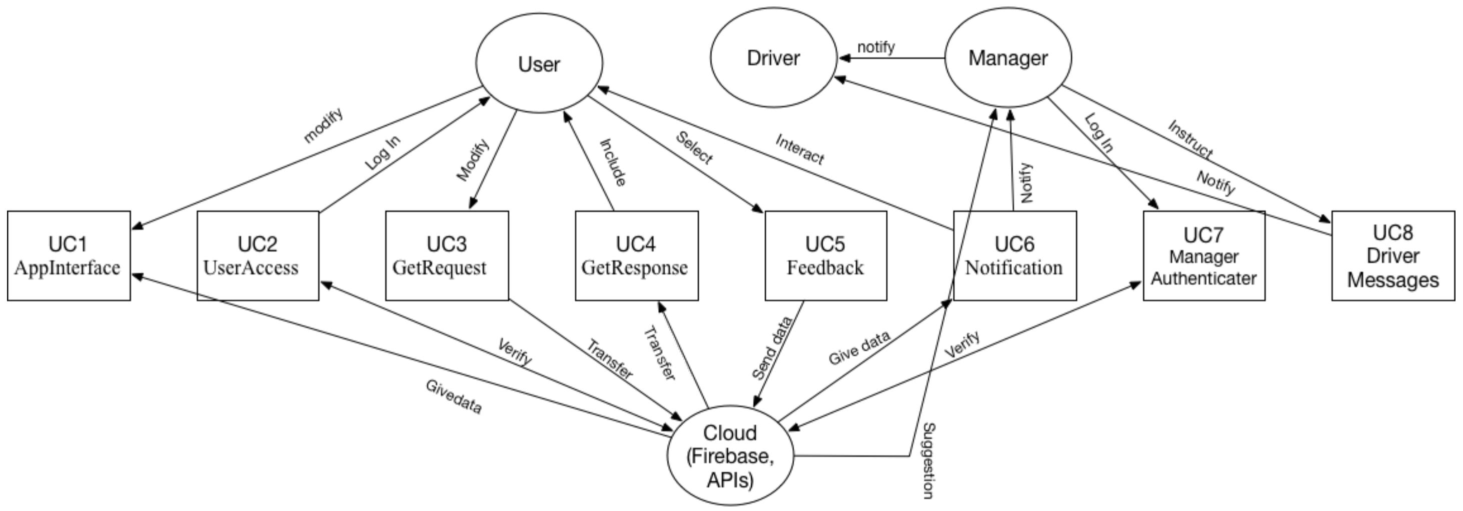
- ○ The goal of mapping service is to show users' current location and based on the current location, determine whether they can make a request or not.

C. Use Cases

I. Casual Description

- UC1: AppInterface
  - Pages of the application that enable users to log in, make request, know the time of arrival and send feedback.
- UC2: UserAccess
  - A login page will be created to verify that the user is a Rutgers student. When the user successfully logs in.
- UC3: GetRequest
  - AppInterface will request some information from the user to specify their needs. User will tell the AppInterface the original bus stop, the preferred bus route, the destination bus stop.
- UC4: GetResponse
  - A response page will be created after the user puts all the details into the UC3 page. Users are able to know the estimated time of arrival (ETA).
- UC5: SendFeedback
  - User can based on bus ETA, send some feedback to database to let the system know whether they get on the bus or not.
- UC6: GetNotification
  - User will receive notifications from the AppInterface. Notification include the expected rush hour in rutgers and the unexpected demand surge.
- UC7: ManagerAuthenticator (sub-use case)
  - Manager will be able to log in to the web interface designed for accessing the bus management system. They can visualize the number of requests for each routes and stations on the screen. (Details are shown in the manager webpage interface design) Will use dummy variables or not activate the login function since Rutgers will not allow us to access their database.
- UC8: SendMessageToDriver
  - Manager will be able to send message to a specific bus driver or broadcast a message to all the bus drivers. (Considering the safety issues, the tool will be the hardware system of rutgers as default)

II. Use Case Diagram



III. Traceability Matrix

Req	Priority Weights	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
ST1	2	X	X						
ST2	3	X		X					
ST3	3		X						
ST4	6			X					
ST5	4				X	X	X		
ST6	2	X				X			
ST7	5								
ST8	2						X		
ST9	2								

ST10	4								
ST11	3								X
ST12	6								
ST13	4							X	
ST14	1								X
ST15	3						X		
ST16	2								
ST17	2							X	
ST21	1	X	X						
ST22	2	X		X					
ST23	0.5					X			
ST24	0.5	X				X			
ST25	1						X		
ST26	4								
ST27	0.5								
ST28	0.5								
ST29	2								
ST30	1								X
ST31	2								
ST32	0.5								
ST33	1								
ST34	4	X	X			X			
Total PW		14.5	10	11	8	11	15	6	5

IV. Fully-Dressed Description

UC-1: App interface

Use Case UC	App Interface
Related Requirement:	All STs interacting with Application
Initiating Actor:	App Users
Actor's goal:	Includes several interfaces (Log in page, Request page, feedback page). Interfaces are going to help users visualize which button to press and request.
Participating Actor:	Application
Preconditions:	● For Request page and feedback page, it may require users to log in successfully by entering correct username and password.
Postconditions:	● Display correct page when requested
Flow of Events for Main Success Scenario	1. App users start to use the application, display log-in page. 2. App can display correct pages automatically all the time (If authentication failed, remain at loggin page etc) 3. Icons can be pressed and are able to jump to specific pages when needed.

UC-2: UserAccess

A login page will be created to identify the user's rutgers student identity. When user successfully login, the application will determine whether user can make the request or not based on current user location.

Use Case UC	UserAccess
Related Requirement:	ST-1
Initiating Actor:	Students or visitors
Actor's goal:	To login in the app and request a bus
Participating Actor:	Bus driver, User Application, Database, Mapping service
Preconditions:	● The system can recognize the user's identity ● System ask users for NetID and password
Postconditions:	● system allow users who enter the correct information to another page ● system make users who enter the wrong information stay at this page
Flow of Events for Main Success Scenario	-> 1.Student or Visitor touch the icon on smartphone to open the "User Application". 2. Student or Visitor type in their NetID and password. <- 3. System check the information and jump to page "GetRequest"

UC-3: GetRequest

Use Case 3	GetRequest
Related Requirement	ST-2 ST-4 ST-22
Initiating Actor	User who wants to take bus
Actor's goal	App need to collect some information from users to specify their demands.
Participating Actor	AppInterface, Database
Preconditions	-Users know which buses they want to take and where they want to go.

	-App can correctly receive information and can specify information from different users.
Postconditions	-App receive users' information and specify their demands. -After users enter information, AppInterface will link users to next page, "GetResponse".
Flow of Events for Main Success Scenario	1. User begins to enter three important information into App: -Original bus stop. -The preferred bus route. -Destination bus stop. 2. App receive information and specify demand. 3. Users will jump to next page, "GetResponse".

UC-4: GetResponse

Use case 4	Get Response
Related Requirement	ST-5
Initiating Actor	User who logged in and request for the bus
Actor's Goal	User will know the estimated time of arrival (ETA)
Participating Actor	Application,Database,Tracking Service
Preconditions	Application is available Database is not empty
Postconditions	1.Students know the bus route they are waiting. 2.Students know the waiting time of selected bus to their current bus stop.
Flow of Events for Main Success Scenario	←the system track the buses and display buses arrival time (ETA)

UC-5: Feedback

Use Case UC-5	Feedback
Related Requirement:	ST-5,ST-6,ST-23,ST-24,ST-34
Initiating Actor:	Users who login and requested for bus
Actor's goal:	To check users get on bus or not and help to create future schedule
Participating Actor:	Database
Preconditions:	A route of bus is requested by users
Postconditions:	A bus has passed the requested station
Flow of Events for Main Success Scenario	1. The App sends a notification when requested bus arrives. 2. Users choose "I'm on the bus" or "I'm not on the bus" 3. Users click "I'm on the bus" 4. System signals affirmation, signals to CheckDevice to complete users' request. signals to MessageDevice to send users a questionnaire about improving advices signals to ScheduleDevice to collect successful sample
Flow of Events for Extensions(Alternate Scenarios)	1. The App sends a notification when requested bus arrives. 2. Users choose "I'm on the bus" or "I'm not on the bus" 3. Users click "I'm not on the bus" 4. System signals affirmation, signals to CheckDevice to resend the request. signals to Timer to recalculate the bus arriving time signals to MessageDevice to send users a notification about next bus signals to ScheduleDevice to collect failure sample

UC-6: ManagerAuthenticator

Use Case UC-7	ManagerAuthenticator (sub-use case)
Related Requirements:	ST-12,ST-13,ST-17
Initiating Actor:	Manager

Actor's Goal:	To be positively identified by the system & Check the real time status
Participating Actors:	ManagerApplication (web interface)
Preconditions:	● The set of valid keys stored in the system (Since the web is auxiliary tool for mobile app, managers can check real time bus status without logging in)
Postconditions:	● The manager is authenticated and logged into the web interface for managing the buses/drivers
Flow of Events for Main Success Scenario:	← 1: System prompts the initiating actor for identification, e.g., alphanumeric key → 2: Manager (initiating actor) supplies a valid identification key. ← 3: System (a) verifies that the key is valid, and (b) signals to the actor the key validity
Flow of Events for Extensions (Alternate Scenarios):	← 2a: Manager enters an invalid identification key ← 1: System (a) detects error, (b) signals to the actor, (c) gives the actor an option of retrieving the identification key using text message → 2: Manager supplies a valid identification key 3: Same a Step 3 above

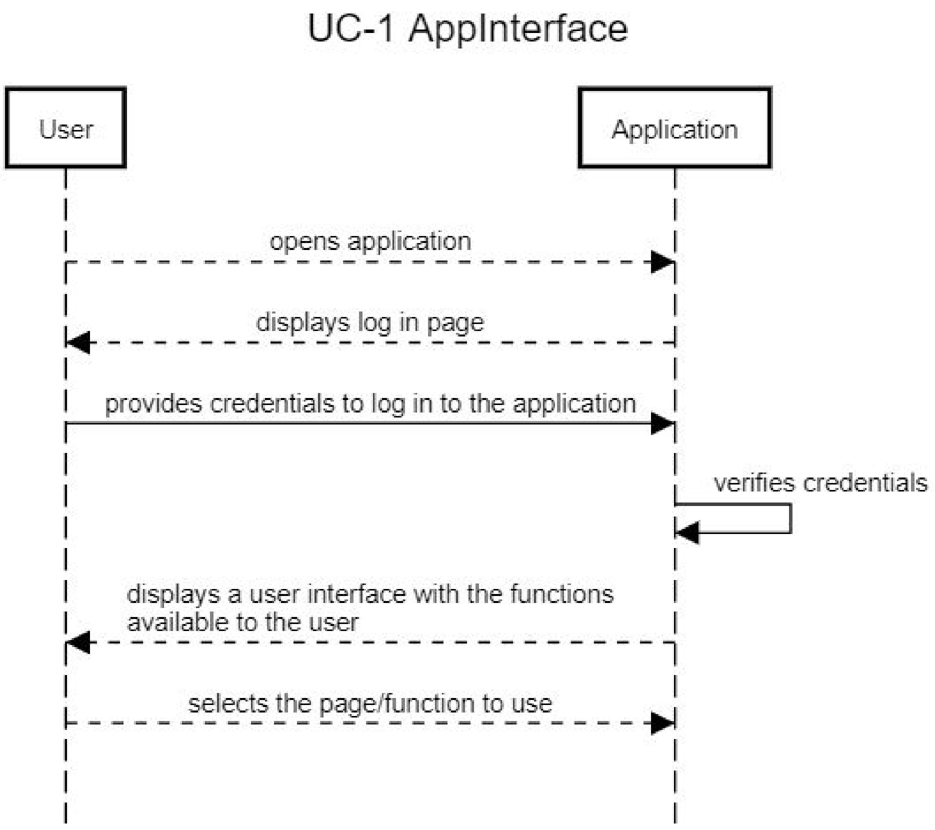
UC-7: SendMessageToDriver

Use case UC-8	SendMessageToDriver
Related Requirements:	ST-14,ST-15
Initiating Actor:	Manager
Actor's Goal:	To successfully send a message to a specific bus driver or broadcast a message to all bus drivers
Participating Actors:	Manager Application (web interface), Bus Driver
Preconditions:	● Managers first check the condition of real time status, if necessary, they will send message to bus drivers using hardware

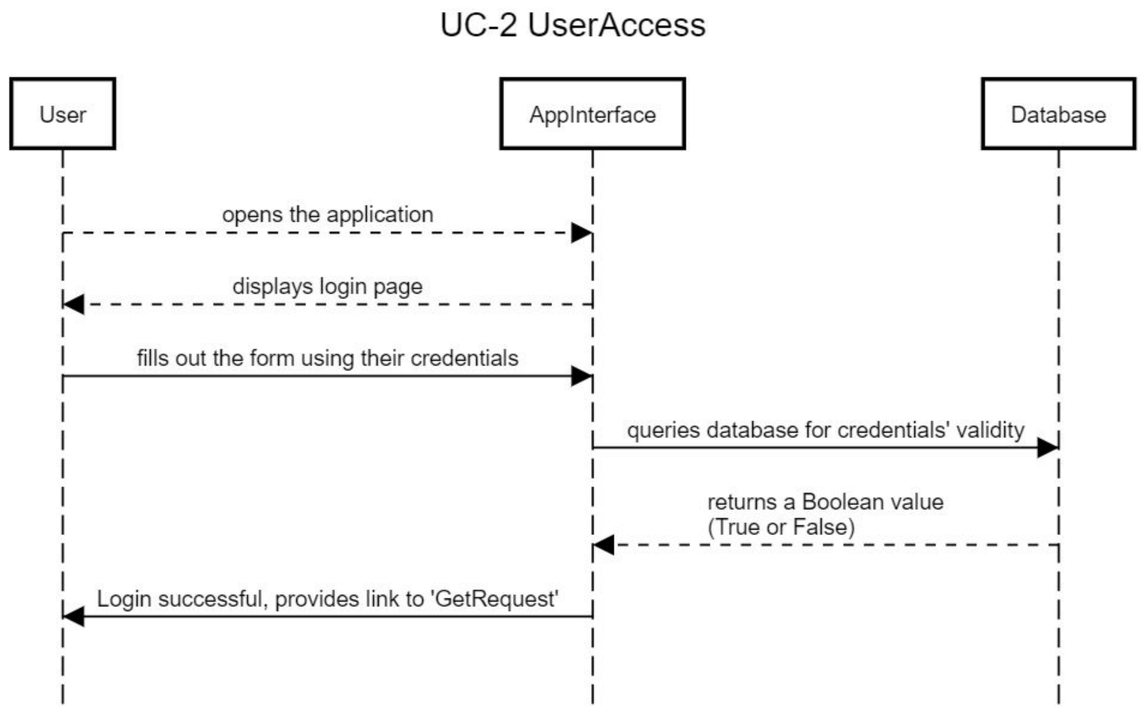
Postconditions:	● The message that the manager intended to send is received by the bus driver(s).
Flow of events for main success scenario:	← 1: The manager logs into the web interface and opts to send a message to either a specific bus driver or broadcast a message to all the bus drivers 2: include: ManagerAuthenticator → 2: The manager check conditions of bus route displayed on web ← 3: Message sent by managers by default → 4: The bus driver receives the message that the manager sent and either responds or acts accordingly.

D. System Sequence Diagram

Use Case 1: AppInterface

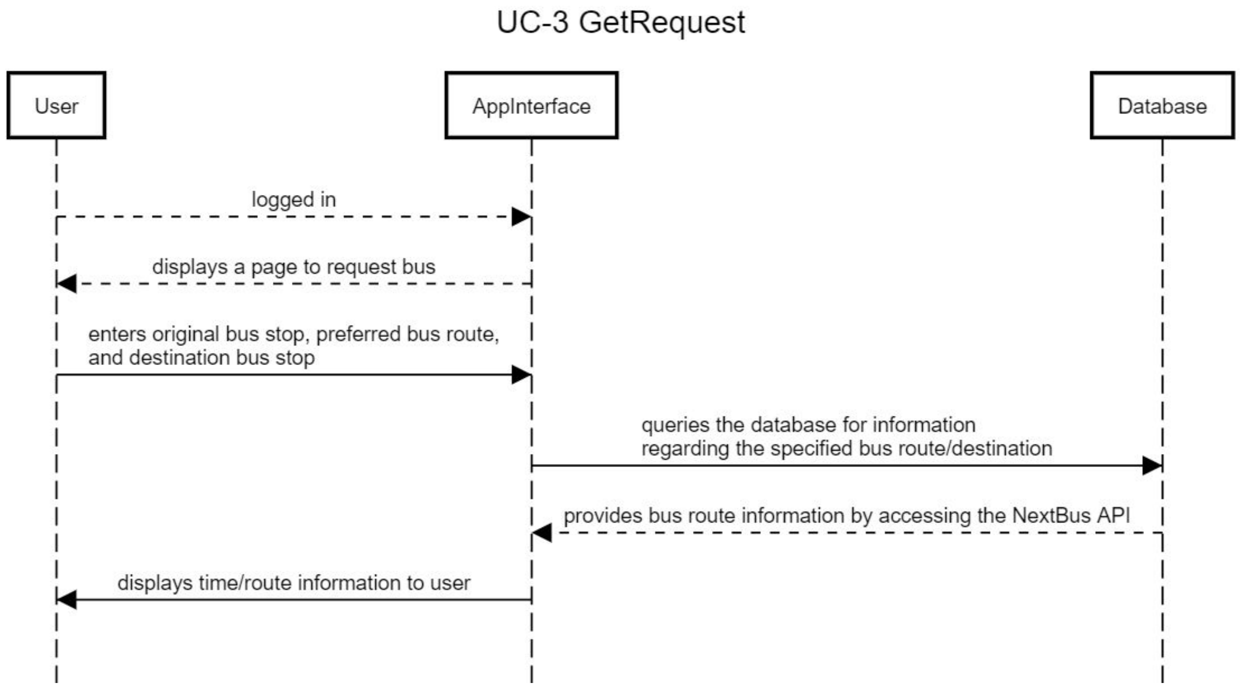

UC-1 AppInterface
User
Application
opens application
displays log in page
provides credentials to log in to the application
verifies credentials
displays a user interface with the functions available to the user
selects the page/function to use

Use Case 2: UserAccess

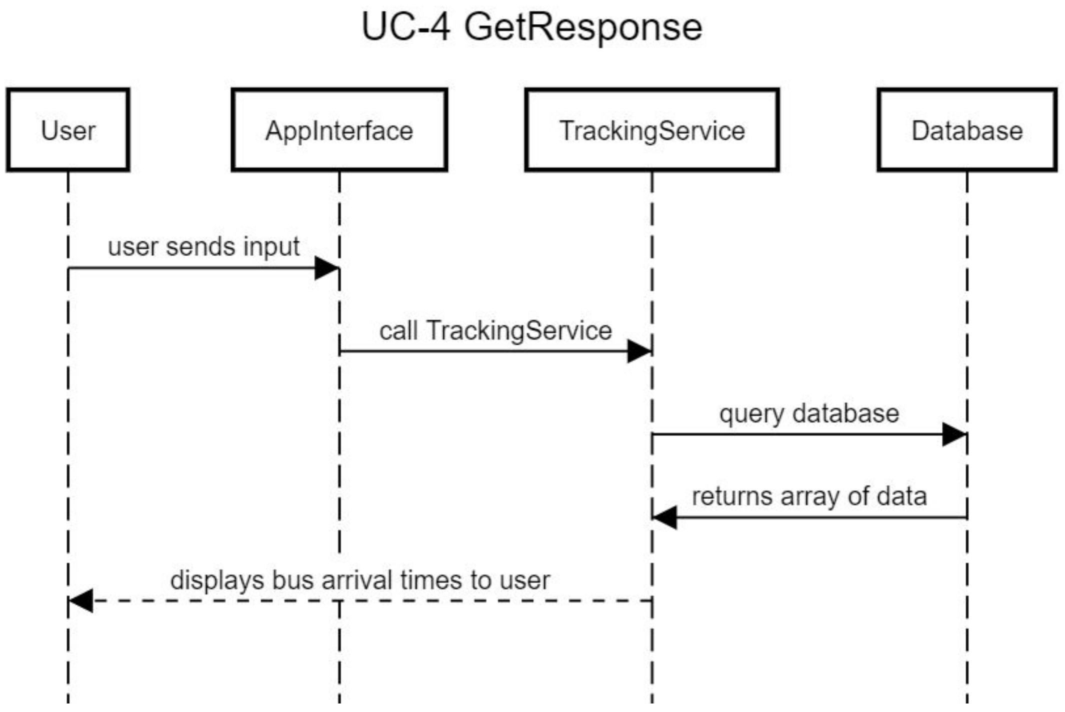

UC-2 UserAccess
User
AppInterface
Database
opens the application
displays login page
fills out the form using their credentials
queries database for credentials' validity
returns a Boolean value (True or False)
Login successful, provides link to 'GetRequest'

Use Case 3: GetRequest

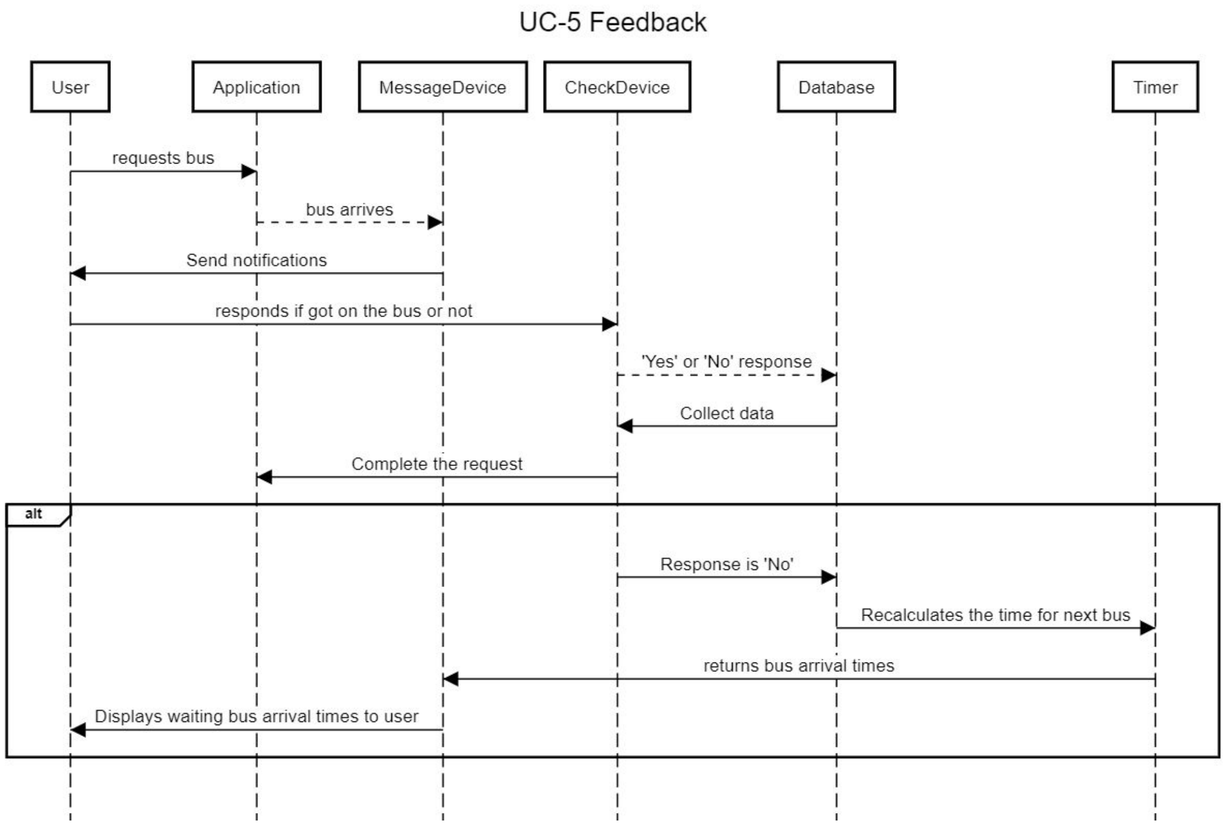
Note: This use case sequence takes place once the user has logged in successfully into the application and is on the page to select the original bus stop, preferred bus route, and destination bus stop.


UC-3 GetRequest
User
AppInterface
Database
logged in
displays a page to request bus
enters original bus stop, preferred bus route, and destination bus stop
queries the database for information regarding the specified bus route/destination
provides bus route information by accessing the NextBus API
displays time/route information to user

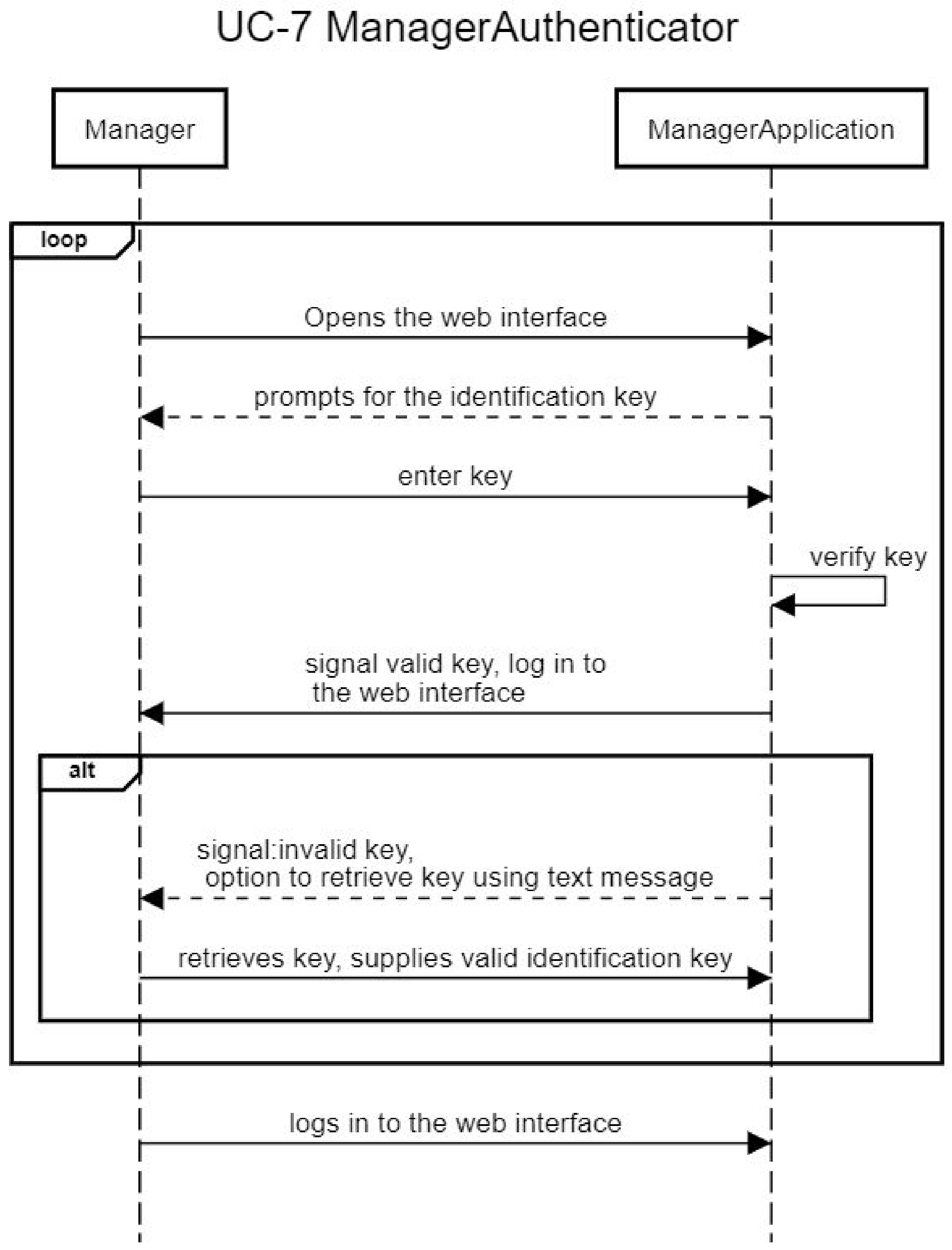
Use Case 4: GetResponse



Use Case 5: Feedback

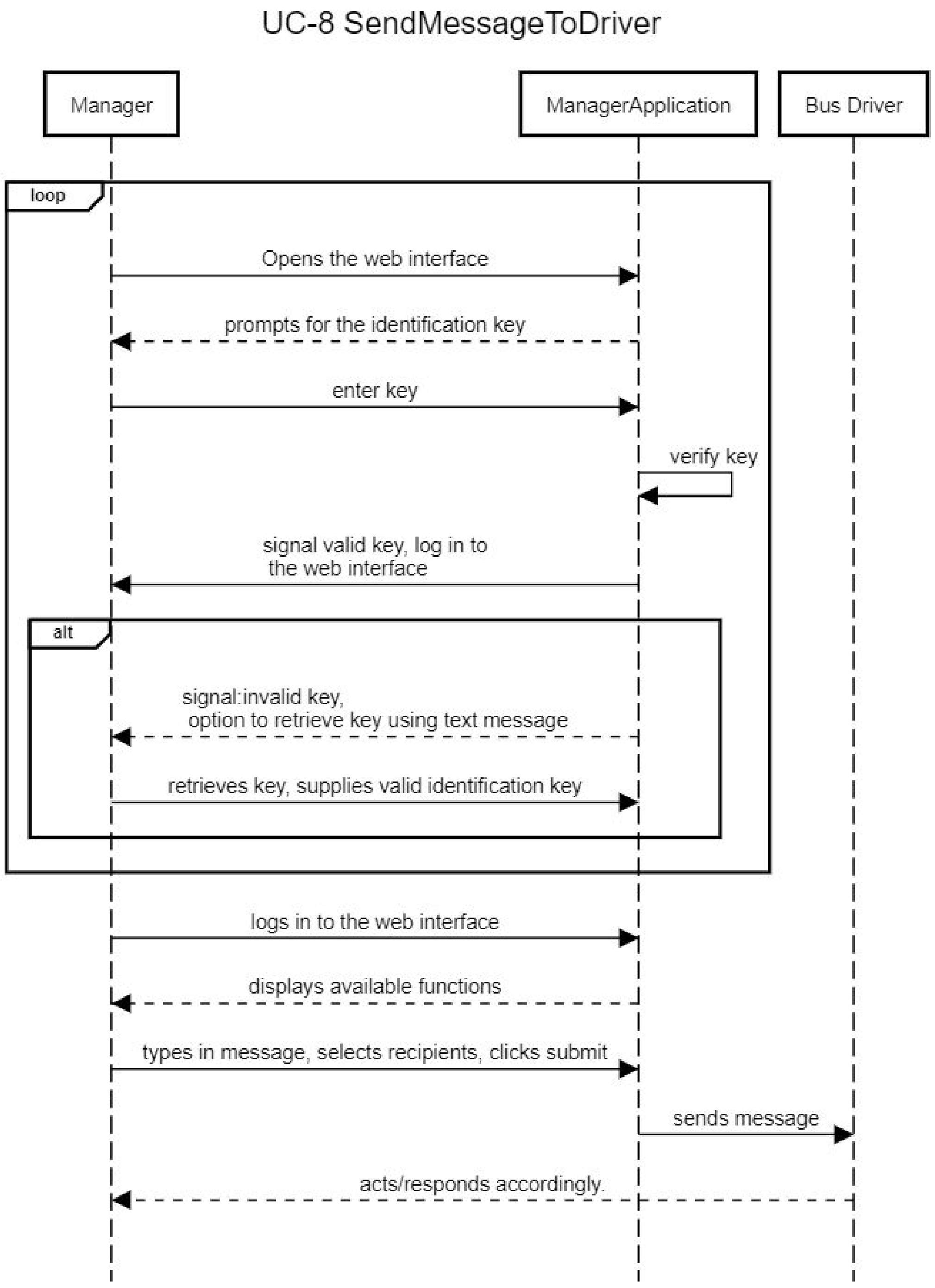


Use Case 6: ManagerAuthenticator


UC-7 ManagerAuthenticator
Manager
ManagerApplication
loop
Opens the web interface
prompts for the identification key
enter key
verify key
signal valid key, log in to the web interface
alt
signal:invalid key, option to retrieve key using text message
retrieves key, supplies valid identification key
logs in to the web interface

Use Case 7: SendMessageToDriver

(Revised :Considering the safety issue when driving, message will be sent by default (using hardware))


UC-8 SendMessageToDriver
Manager
ManagerApplication
Bus Driver
loop
Opens the web interface
prompts for the identification key
enter key
verify key
signal valid key, log in to the web interface
alt
signal:invalid key, option to retrieve key using text message
retrieves key, supplies valid identification key
logs in to the web interface
displays available functions
types in message, selects recipients, clicks submit
sends message
acts/responds accordingly.

Section 5. Effort Estimation

Student's Mobile Application

1. *Authentication (2 click and keystrokes)*
   a. Press the keyboards in mobile application in order to input Rutgers NetID and Password.
   b. Check for saving the Rutgers NetID and Password
   c. Click the "login" button to access application

2. *Navigation of Campus Bus Stops (3 clicks)*
   a. Application Auto-confirms the current bus stop, but the user can click their "current bus stop" in order to avoid system/GPS error (not necessary to count in clicks).
   b. Click the "destination bus stop"
   c. Click "bus route" in the Google Map interface with suggested route showed on screen.
   d. Click "Next" in order to send their requests

3. *Confirmation (0 clicks)*
   a. If the student feels they have their selected the wrong options, they can click "Go back" in order to change their request (not necessary to count in clicks).

4. *Feedbacks (1 click)*
   a. User can Click "On bus" or "Not on bus" in order to send feedbacks to the server to notify our system.

Manager’s Web Application: (Auxiliary tool for mobile app)

1. *Authentication (1 click and keystrokes)*
   a. Press the keyboard in web application to input username and Password.
   b. Click the “login” button to access application

2. *Operation of bus route(2 clicks)*
   a. Press “Check route” to check the real time bus status.
   b. Notify bus drivers if necessary by hardware

Bus drivers

Drivers simply need to check their static schedule and instructions sent by manager.

Section 6. Domain Analysis

A. Domain Model

Use Case 1: App Interface

- Concept definitions:

Responsibility Description	**Type**	**Concept Name**
Once the app is opened, display login page	D	App Interface
After logging in, display the bus request page and options	D	App Interface
Display information about the selected route	D	App Interface
Display feedback options when user finish their request	D	App Interface

- Association Definitions:

Concept pair name	**Associated Definition**	**Associated Name**
User↔Identity Check	In order to use the request function, users should login to the app successfully first (precondition). After identity check, users can request or search based on the display by interface.	Identity check

- Attribute Definitions:

Concept	Attributes	Attribute Description

App Interface	All Options button App Menu	Display space to login and options menu, help users to visualize the app
Database	Bus route data	Associated with the request page, display how much time left for bus routes
Database	Identity check	Associate with login page, display space for users to login and check if they used a valid username and password

Use Case 2: UserAccess

- Concept definitions

Responsibility Description	Type	Concept Name
Check the current App user's identity	D	Database
Once verify user's identity, show "GetRequest" page	D	App Interface

- Association Definitions

Concept Pair name	Associated Definition	Associated Name
User↔Identity Check	User fill out the NetID and password information. Using Rutgers authentication API check user's identity	Identity check
UserAccess↔GetRequest	Once confirm user's identity, link to "GetRequest" page	Linking

- Attribute Definitions

Concept	Attributes	Attribute Description
Application	Rutgers authentication	A system to check the user's identity
	Forget password	A Rutgers website that will help find password
Database	Rutgers authentication API	Table of valid NetID information

Use case 3: GetRequest

- Concept Definition

Responsibility Description	Type	Concept Name
Users enter information into AppInterface	D	App Interface
AppInterface sends demand to database	D	Database
AppInterface receives demand and show "GetResponse" page.	D	App Interface

- Association Definitions

Concept Pair Name	Associated Definition	Associated Name
User↔AppInterface	User enter their information into AppInterface including original bus station, bus and destination.	DataCollection
AppInterface↔Database	After users enter information, AppInterface sends demand to Database and Database will	DataCollection

	write down the information.	
AppInterface↔GetResponse	AppInterface receives demand and builds respond, then links user into next page, GetResponse.	Linking

- Attribute Definitions

Concept	**Attributes**	**Attribute Description**
AppInterface	All information button	Display all of demands that users need and receive information from users.
Database	Demands data	AppInterface receives information from users and sends to Database.

<u>Use Case 4: GetResponse</u>

- Concept definitions

Responsibility Description	**Type**	**Concept Name**
Uses the NextBus API to display the waiting time and dispatch rate for selected bus	D	Tracking Service
Take in user input to find which bus the user is requesting. Output selected the bus route.	D	Application
Contains the bus GPS coordinates and Those data can be accessed by the application.	K	Database

- Association Definitions

Concept Pair name	Associated Definition	Associated Name
Application↔Tracking Service	Application sends user selected bus to the Tracking Service. The Tracking Service returns the waiting time	Provide data
Tracking Service↔Database	The Tracking Service requests Database information to be used in the function and display the correct estimate waiting time.	Provide data

- Attribute Definitions

Concept	Attributes	Attribute Description
Application	Bus Routes	Different bus routes the user requirements
	Time	Display the waiting time
Database	Bus Routes Data	Different buses routes
	Gps Coordinates Data	Show the current buses coordinates

<u>Use Case 5: Feedback</u>

Responsibility Description	Type	Concept Name
Uses the NextBus API to determine the bus is arriving or not	D	Application/Controller

Check users get on the bus or not	D	MessageDevice
Collect users' responds to decide next step	D	CheckDevice
Recalculate the next bus arriving time if user click "I'm not on the bus"	D	Timer
Database collect completed request and failure request to help analyzing the future schedule	K	Database
Sending a notification to the users who do not get on bus about the next bus arriving time	K	Application/Controller

- Association Definitions

Concept Pair name	**Associated Definition**	**Associated Name**
Application↔MessageDevice	Application remind users about the arriving bus and ask for user status(on bus or not)	Notification Services
MessageDevice↔CheckDevice	Check user status and make next decision	Provide data
CheckDevice↔Database	Collect requests feedback for future analyzing and resend the request for failure case.	Provide data
Database↔Timer	Database sends information about the next bus and estimate waiting time.	Provide data

- Attribute Definitions

Concept	**Attributes**	**Attribute Description**

Application	Notifications	Remind for arriving bus
MessageDevice	Check user status	Get on bus or not
CheckDevice	Make decision for database	Determine if resend the request or not
Database	Collect cases Gps Coordinates Data	Completed and failure request Show the current buses coordinates
Timer	Display the bus status	Show users who do not get on bus about the next bus information

Use Case 6: GetNotification

- Concept definitions

Responsibility Description	**Type**	**Concept Name**
Request selected route conditions and GPS	D	Database
Request selected route dispatching rate and arriving time	D	Timer
Send notification with a summary of the requested information	D	App Interface

- Association Definitions

Concept Pair name	Associated Definition	Associated Name
Application↔Database	The application will request data from the database to notify the user of current transportation service conditions	Notification Service

- Attribute Definitions

Concept	Attributes	Attribute Description
Application	Rutgers authentication	A system to check the user’s identity
	Forget password	A Rutgers website that will help find password
Database	Rutgers authentication API	Table of valid NetID information

<u>Use Case 7: ManagerAuthenticator</u>

- Concept Definitions:

Responsibility Description	Type	Concept Name
Shows the login page to the manager	D	AppInterface
Container for manager’s authentication data, such as identification key	K	Key
Verify whether or not the	D	KeyChecker

identification key entered by the manager is valid		
Container for the collection of valid key(s) associated with manager(s)	K	KeyStorage
Send text message to retrieve key	D	KeyRetriever

- Association Definitions:

Concept Pair Name	**Associated Definition**	**Associated Name**
AppInterface ↔Key	The manager enters identification key into the login page of the web interface	Identity Check
Key↔KeyChecker	Key is handed off the KeyChecker to verify its validity	Check Key
KeyChecker↔KeyStorage	KeyChecker will check if the provided key is contained in the KeyStorage	Check in KeyStorage
AppInterface↔KeyRetriever	Sends text message to manager containing the ID key	Retrieve Key

- Attribute Definitions:

Concept	**Attributes**	**Attribute Description**
AppInterface	Authentication	A system to verify manager’s identity

	Retrieve Key	Interfaces with KeyRetriever for manager to retrieve the identification key
Key	Provided key	The key that the manager provides
KeyChecker	Check if Key is valid	Interfaces with the KeyStorage to check if key is valid
KeyStorage	Valid Key Container	Contains the set of valid keys
KeyRetriever	Retrieve key	Text message service (for example, Twilio) to send a text message to manager, containing the valid key

Use Case 8: SendMessageToDriver

Includes Concept Definitions, Association Definitions, and Attribute Definitions from UC-7:ManagerAuthernticator

- Concept Definitions:

Responsibility Description	Type	Concept Name
Shows a menu/user interface with functions available to manager	D	AppInterface
MessageInterface	K	A part of the AppInterface: User Interface to select recipients and type a message to send to bus drivers
MessageSender	D	A part of the MessageInterface, connects to text message service (such as Twilio) to send text message to the bus drivers

- Association Definitions:

Concept Pair Name	Associated Definition	Associated Name
AppInterface↔MessageInterface	AppInterface shows MessageInterface as a function available to the manager	Display function
MessageInterface↔MessageSender	MessageInterface passes off the message typed by the manager to the MessageSender which connects to a text message service to send the message	Send message

- Attribute Definitions:

Concept	Attributes	Attribute Description
AppInterface	Available Functions	Displays a menu or user interface with functions available to the manager

MessageInterface	Select Recipients USING HARDWARE	Gives the option of either selecting a specific bus driver to send a message to, or broadcast a message to all the bus drivers
MessageSender (Using hardware by default)	N/A	N/A

Domain Traceability Matrix

	Application	**MapService**	**Database**	**ManagerApplication**	**TextMessage Service**
UC-1	Display interface	Display requested bus route and location			
UC-2	Receive user's information and link to other pages		Check user's input		
UC-3	Receive user's demand and link to other pages		Remember user's demand		
UC-4	Waiting time	Display	Check the		

	is shown on Application	information of selected route	NextBus API		
UC-5	Reminding user about the arriving bus		Collecting completed and failure request for future analyzing		
UC-6	Application requests transportation service status and sends a push notification		Checks status of bus routes		
UC-7				Provides the log in page for the manager to access the bus management system	Retrieve keys for the manager
UC-8				_Provides the user interface with all the functions that the manager can use	Sends message to the bus driver by hardware

B. System Operation Contracts

Name	UC-1 App interface
Preconditions	● For Request page and feedback page, it may require users to log in successfully by entering correct username and password.
Postconditions	● Display correct page when requested

Name	UC-2 UserAccess
Preconditions	● The system can recognize the user's identity ● System ask users for NetID and password
Postconditions	● system allow users who enter the correct information to another page ● system make users who enter the wrong information stay at this page

Name	UC-3 GetRequest
Preconditions	-Users know which buses they want to take and where they want to go. -App can correctly receive information and can specify information from different users.
Postconditions	-App receive users' information and specify their demands. -After users enter information, AppInterface will link users to next page, "GetResponse".

Name	UC-4 Get Response
Preconditions	Application is available Database is not empty
Postconditions	1.Students know the bus route they are waiting. 2.Students know the waiting time of selected bus to their current bus stop.

Name	UC-5 Feedback
Preconditions	A route of bus is requested by users
Postconditions	A bus has passed the requested station

Name	UC-6 GetNotification
Preconditions	Has set in the options what they want to receive notifications on and at what time intervals
Postconditions	The timer for notifications is reset

Name	UC-7 ManagerAuthenticator
Preconditions	The set of valid keys stored in the system
Postconditions	The manager is authenticated and logged into the web interface for managing the buses/drivers

Name:	UC-8 SendMessageToDriver
Preconditions	The system displays the menu of available functions to the manager
Postconditions	The message that the manager intended to send is received by the bus driver(s).

C. Mathematical Model

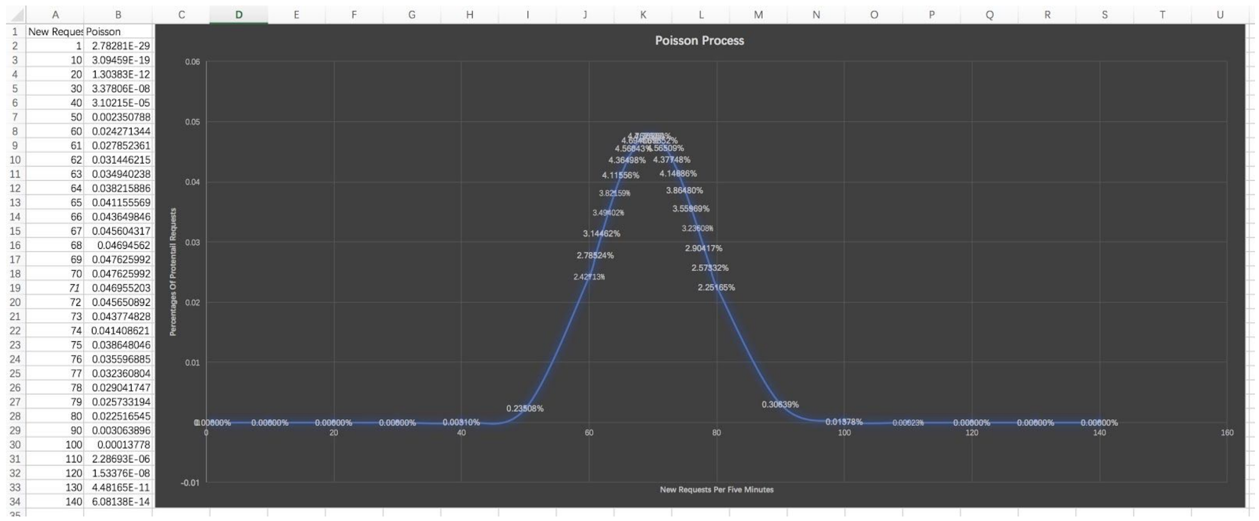
Since the whole system depends on user's request, it is important to make simulated requests correctly and reasonably. To achieve this, we will use Poisson process simulate many users' requests.

For a Poisson process with average rate λ, the probability of getting n requests in the time interval Δt equals:

$$\Pr(n) = \frac{e^{-\lambda\cdot\Delta t}(\lambda\cdot\Delta t)^n}{n!}$$

Since the request includes different types of data and the data are made by ourselves, we will intentionally make our data values so that it would be "reasonable" and "correct" for our simulation.

The Poisson process gives the result that can be analysed to determine whether the data is valid or not.
To better illustrate how this will work, using route B as an example. Based on B bus capacity and amount of buses, λ = 14 requests/minute. The time interval Δt = 5 mins since the highest dispatch rate is 5 minutes. By using different n values, the corresponding results would be symmetric impulse with peak value at n = λ*Δt, in this case is n = 70:

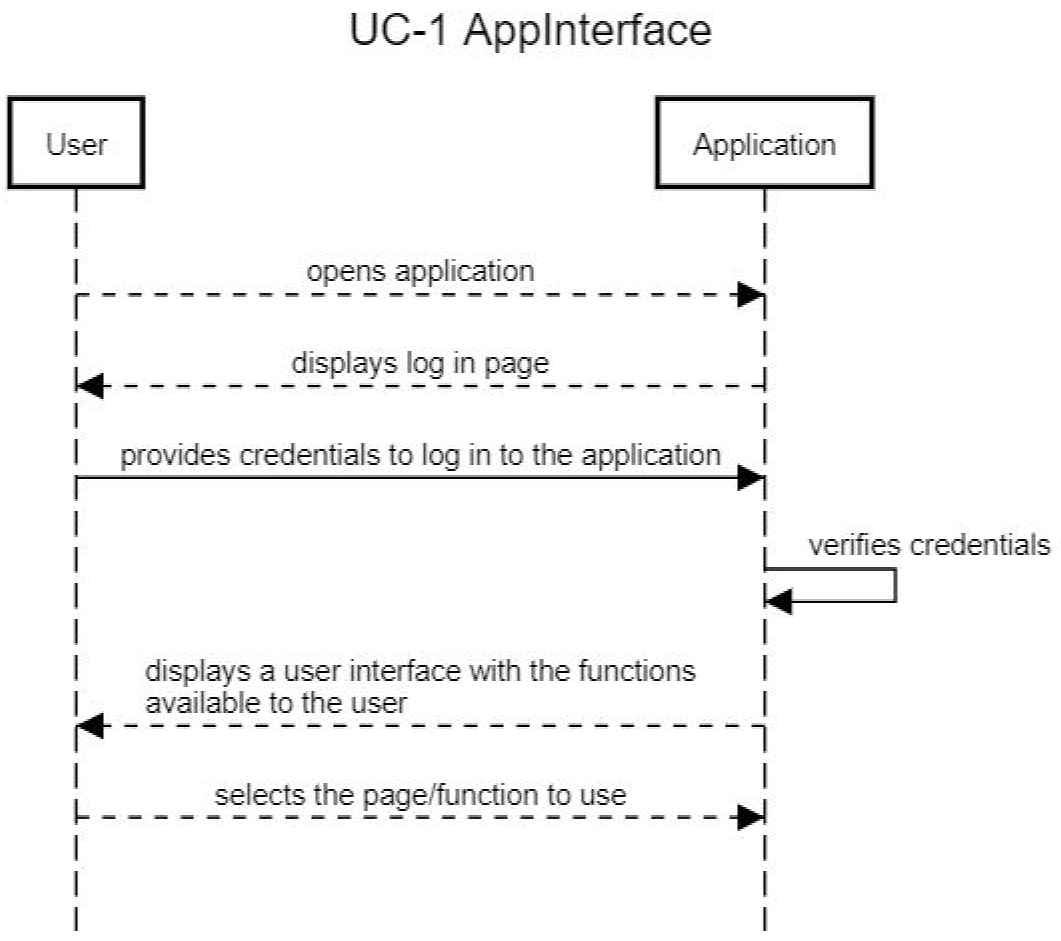

Poisson Process
Percentages Of Protentail Requests
New Requests Per Five Minutes

New Reques	Poisson
1	2.78281E-29
10	3.09459E-19
20	1.30383E-12
30	3.37806E-08
40	3.10215E-05
50	0.002350788
60	0.024271344
61	0.027852361
62	0.031446215
63	0.034940238
64	0.038215886
65	0.041155569
66	0.043649846
67	0.045604317
68	0.04694562
69	0.047625992
70	0.047625992
71	0.046955203
72	0.045650892
73	0.043774828
74	0.041408621
75	0.038648046
76	0.035596885
77	0.032360804
78	0.029041747
79	0.025733194
80	0.022516545
90	0.003063896
100	0.00013778
110	2.28693E-06
120	1.53376E-08
130	4.48165E-11
140	6.08138E-14

By defining valid values between 60 and 80, any data that inside this interval will be the valid values. Also, since the number of users may be very small during weekend, our app now only apply to the daily workdays. Special events & days may not be included so far.

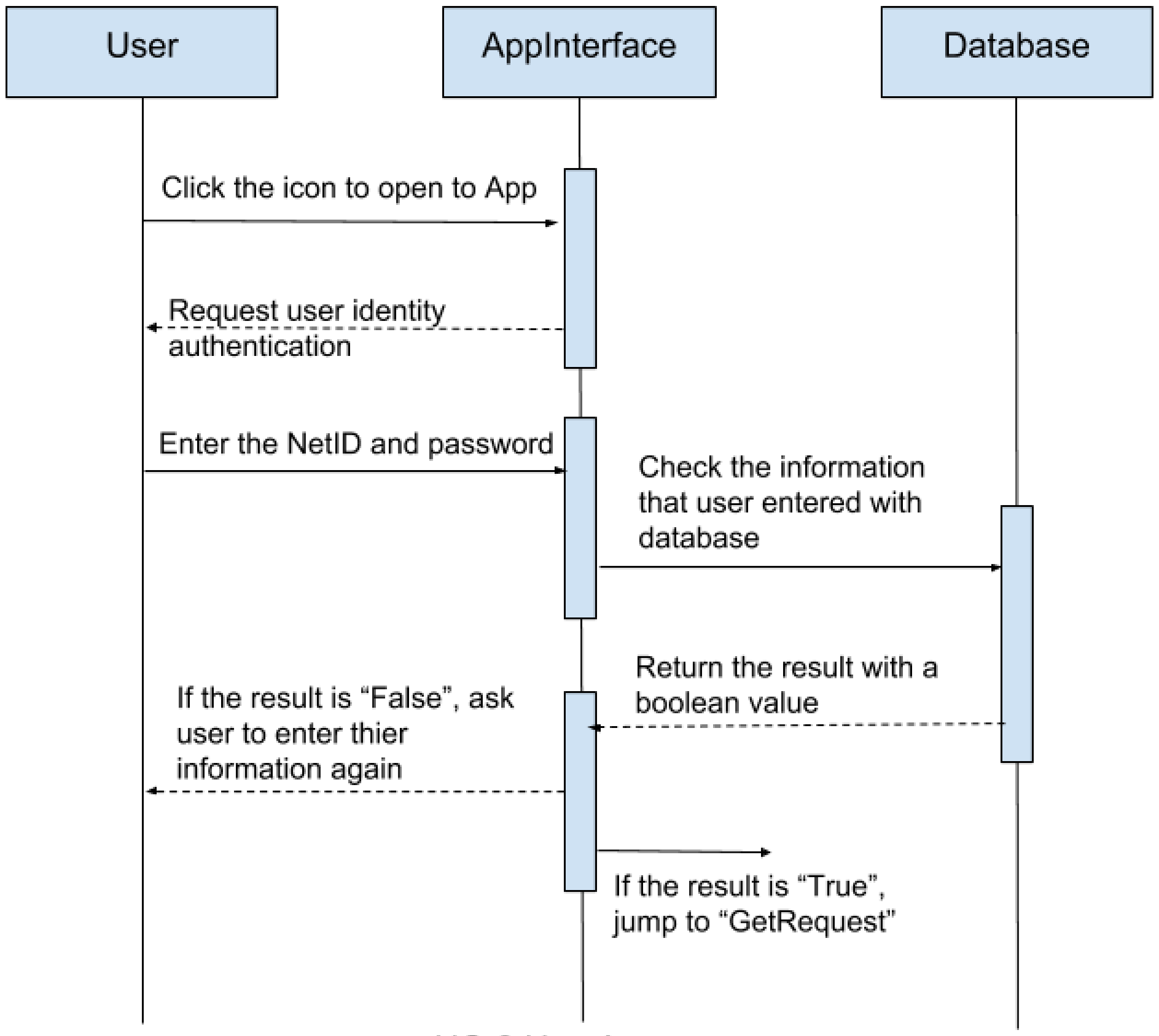
Section 7. Interaction Diagrams

UC-1


UC-1 AppInterface
User
Application
opens application
displays log in page
provides credentials to log in to the application
verifies credentials
displays a user interface with the functions available to the user
selects the page/function to use

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

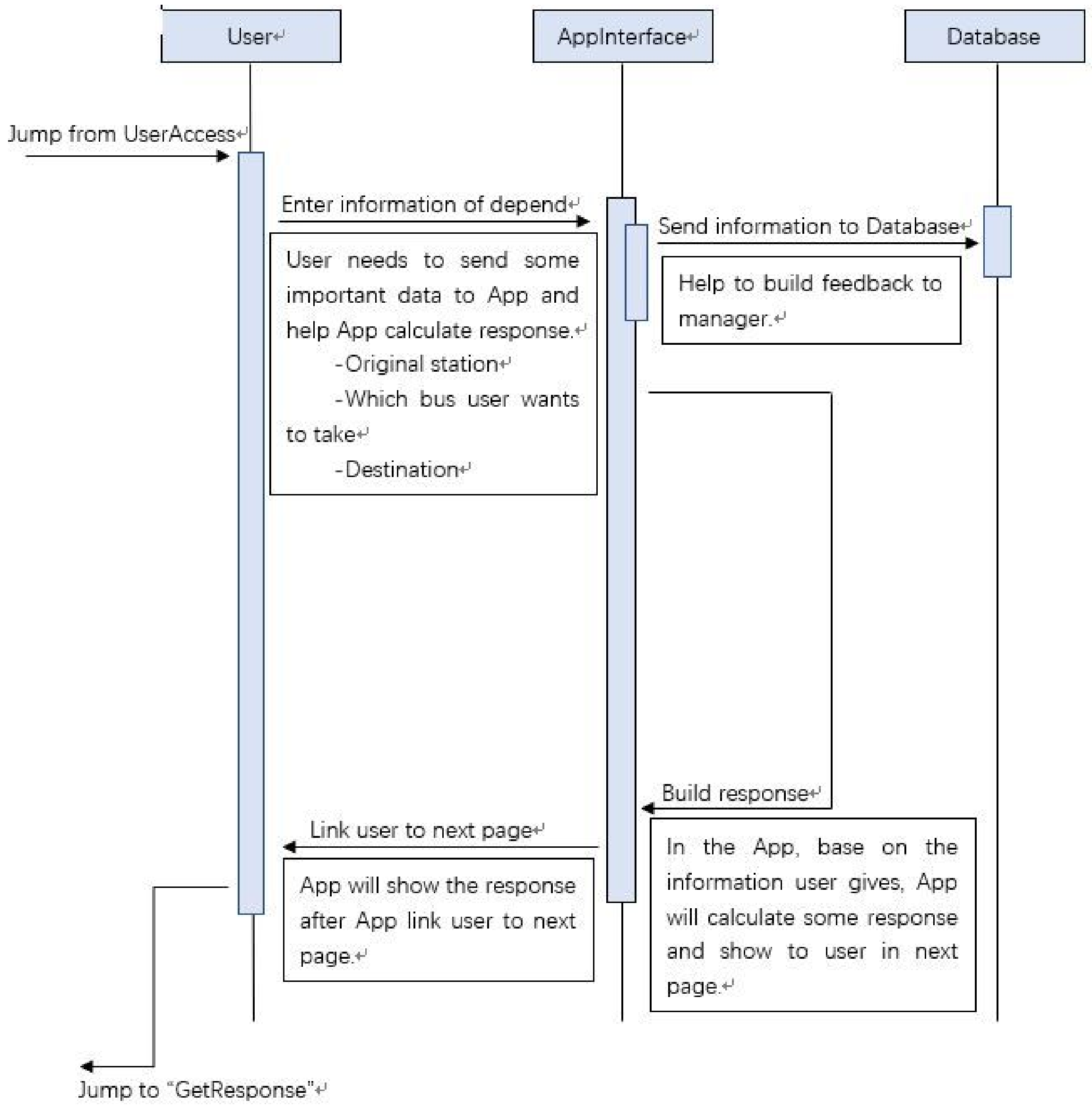

User
AppInterface
Database
Click the icon to open to App
Request user identity authentication
Enter the NetID and password
Check the information that user entered with database
Return the result with a boolean value
If the result is "False", ask user to enter thier information again
If the result is "True", jump to "GetRequest"

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.

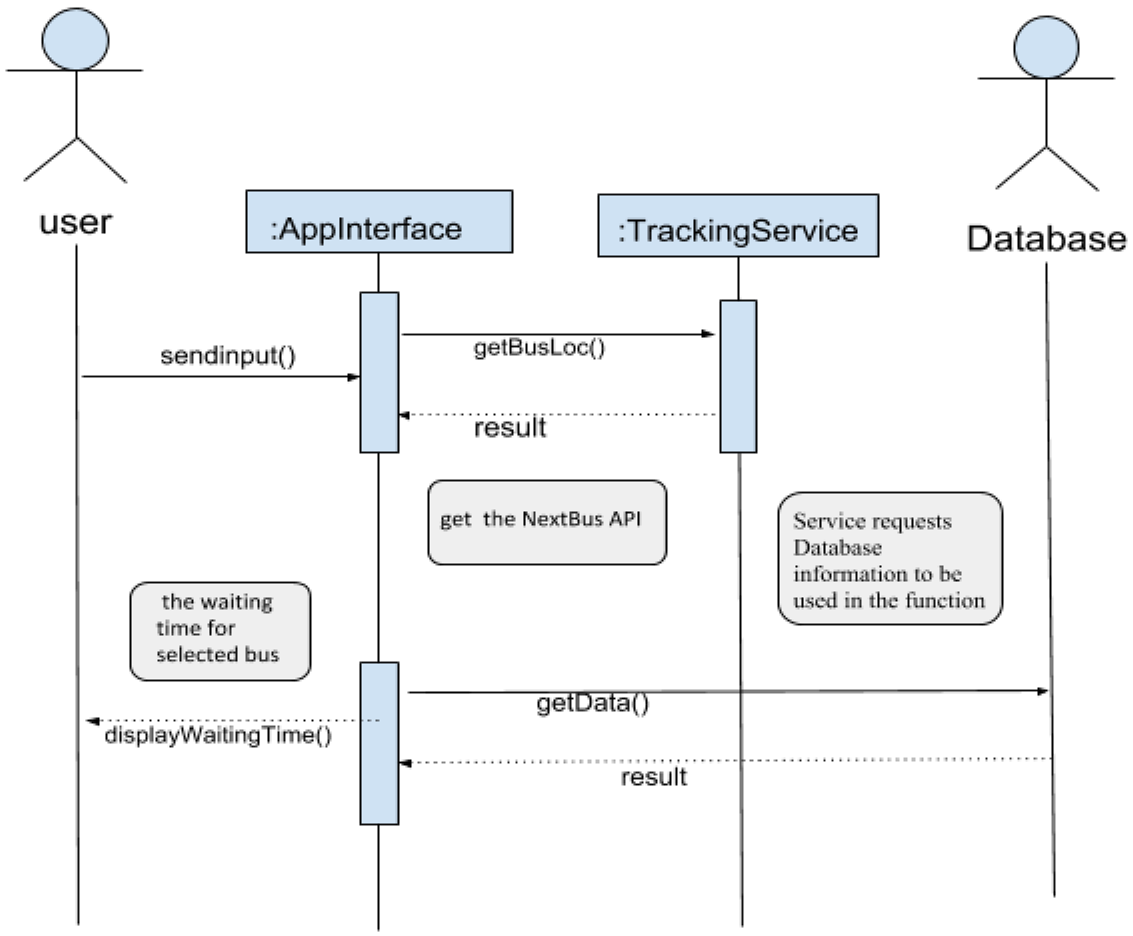
The high cohesion principle applied to this use case since the role of Database is checking the information only.

UC-3


User
AppInterface
Database
Jump from UserAccess
Enter information of depend
User needs to send some important data to App and help App calculate response.
-Original station
-Which bus user wants to take
-Destination
Send information to Database
Help to build feedback to manager.
Build response
In the App, base on the information user gives, App will calculate some response and show to user in next page.
Link user to next page
App will show the response after App link user to next page.
Jump to "GetResponse"

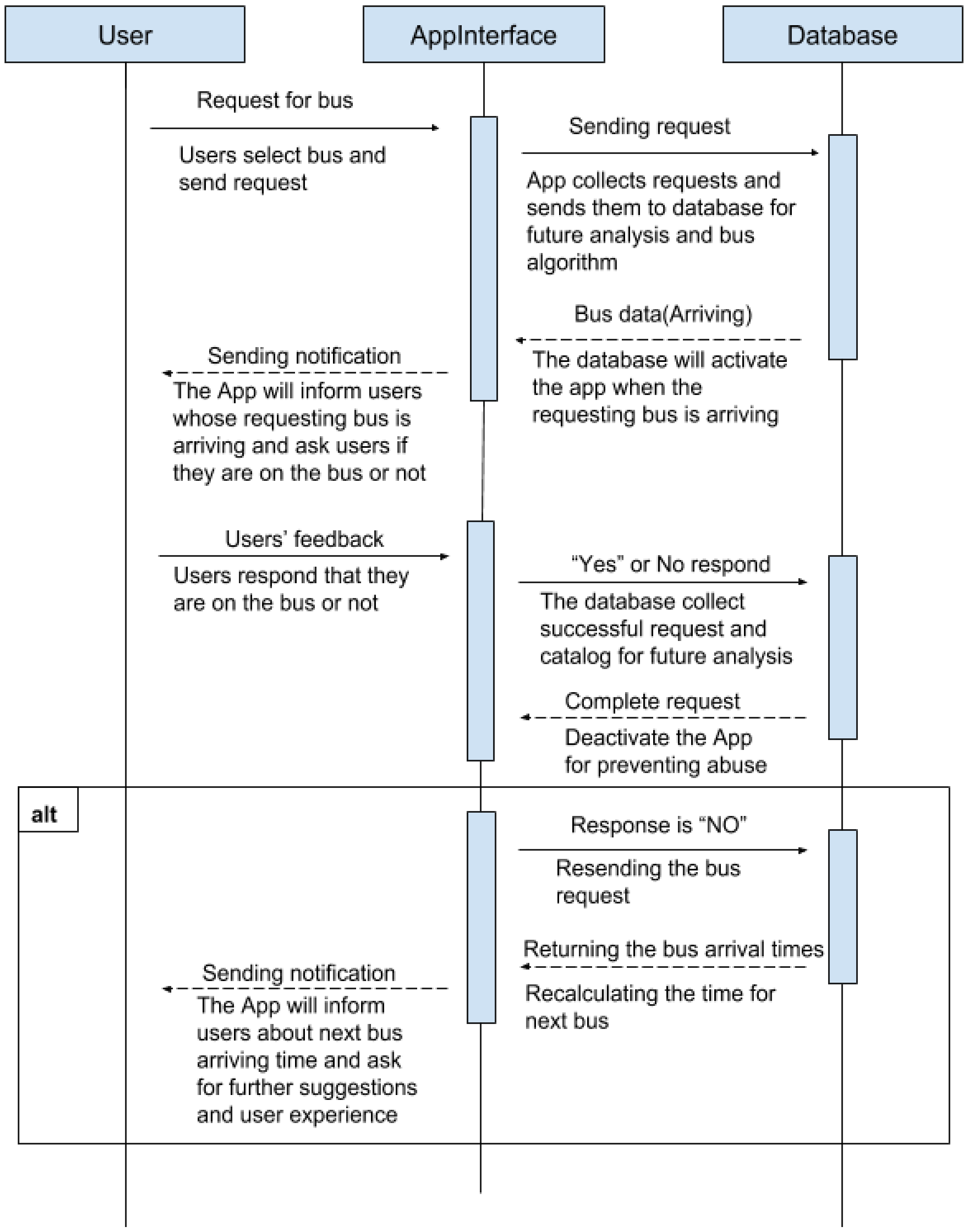
This use case is to call users to enter what their demands. In this page, users need to enter original bus station, destination and the bus they want to take. Appinterface will send information that is from users and will link users to next page to receive response.

UC-4



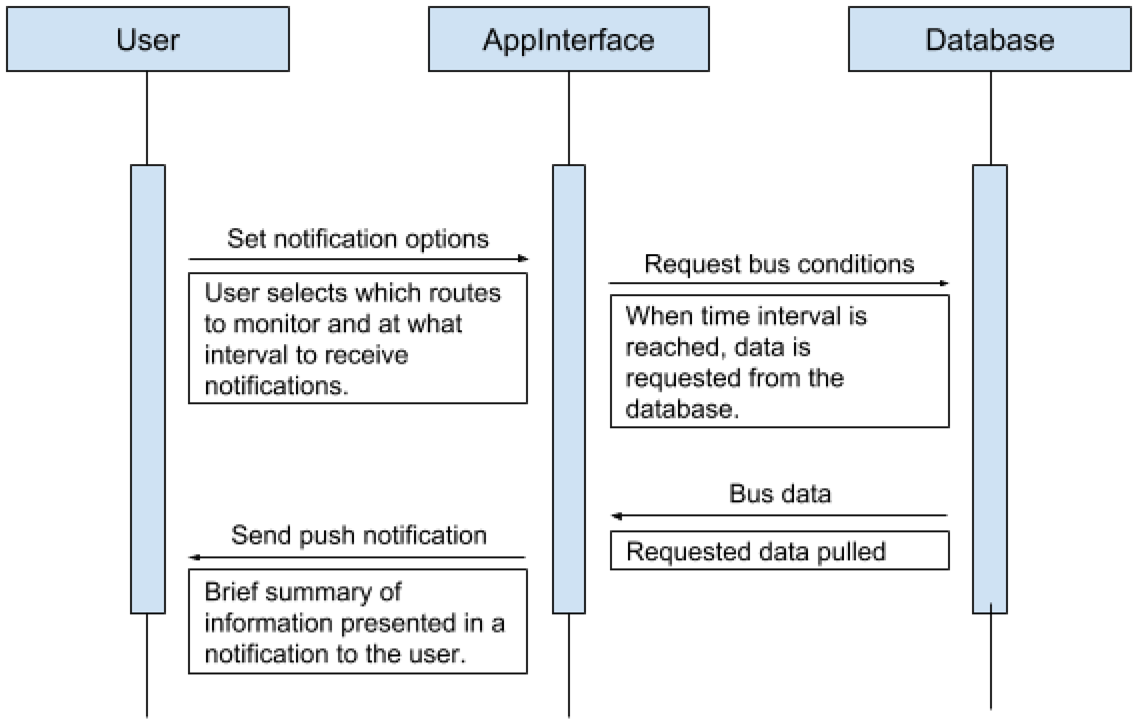
The use case has the responsibility to know the estimated time of arrival (ETA) for buses. user enters his or her data to the bus Application and sends user selected bus to the Tracking Service. The Tracking Service requests Database information to be used in the function and display the correct estimate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

UC-5


User
AppInterface
Database
Request for bus
Users select bus and send request
Sending request
App collects requests and sends them to database for future analysis and bus algorithm
Bus data(Arriving)
The database will activate the app when the requesting bus is arriving
Sending notification
The App will inform users whose requesting bus is arriving and ask users if they are on the bus or not
Users' feedback
Users respond that they are on the bus or not
"Yes" or No respond
The database collect successful request and catalog for future analysis
Complete request
Deactivate the App for preventing abuse
alt
Response is "NO"
Resending the bus request
Returning the bus arrival times
Recalculating the time for next bus
Sending notification
The App will inform users about next bus arriving time and ask for further suggestions and user experience

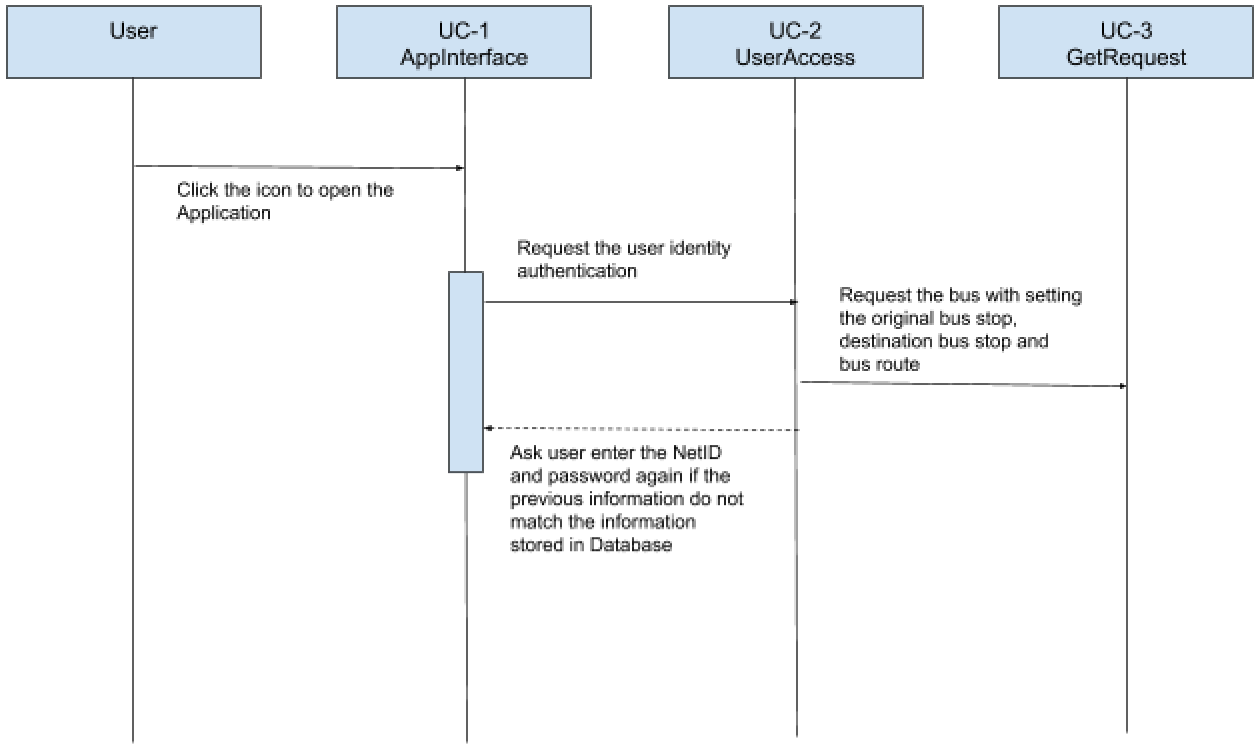
This diagram represents UML for UC5 which is responsible for check users' status and feedback. The Appinterface's duties are informing users about their requesting buses and classifying their request for database. The duties for database are analyzing users' requests and determine if users get on the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

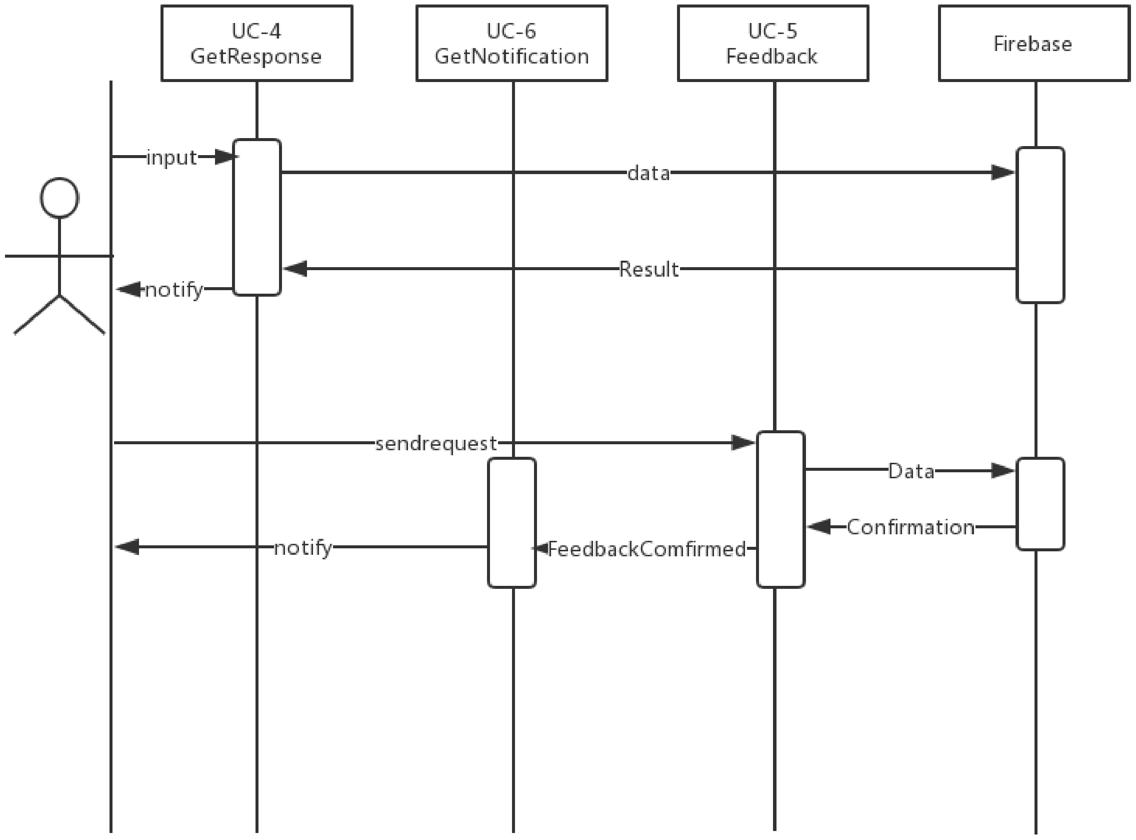

User
AppInterface
Database
Set notification options
User selects which routes to monitor and at what interval to receive notifications.
Request bus conditions
When time interval is reached, data is requested from the database.
Bus data
Requested data pulled
Send push notification
Brief summary of information presented in a notification to the user.

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.

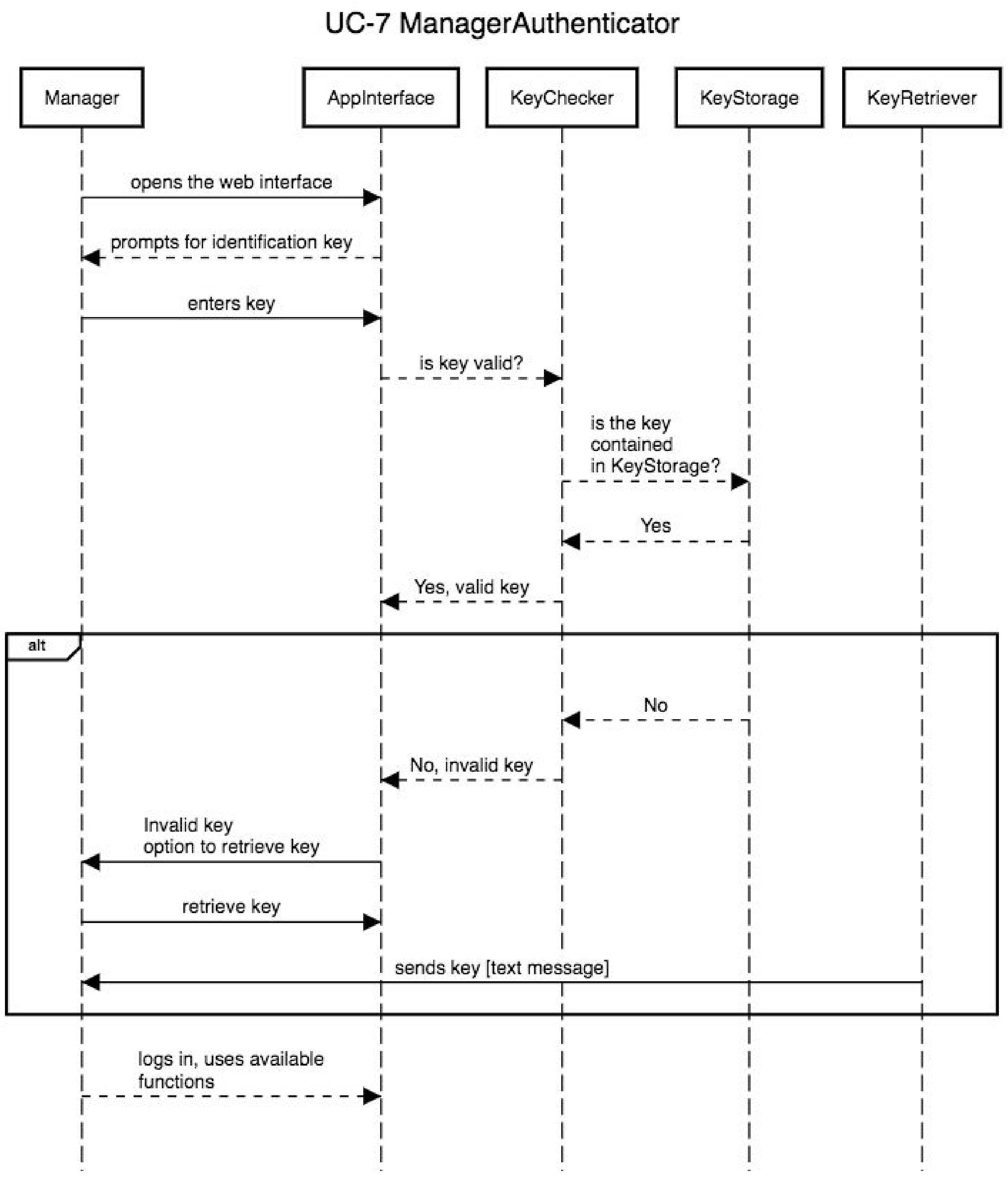
User Case 1-2-3


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

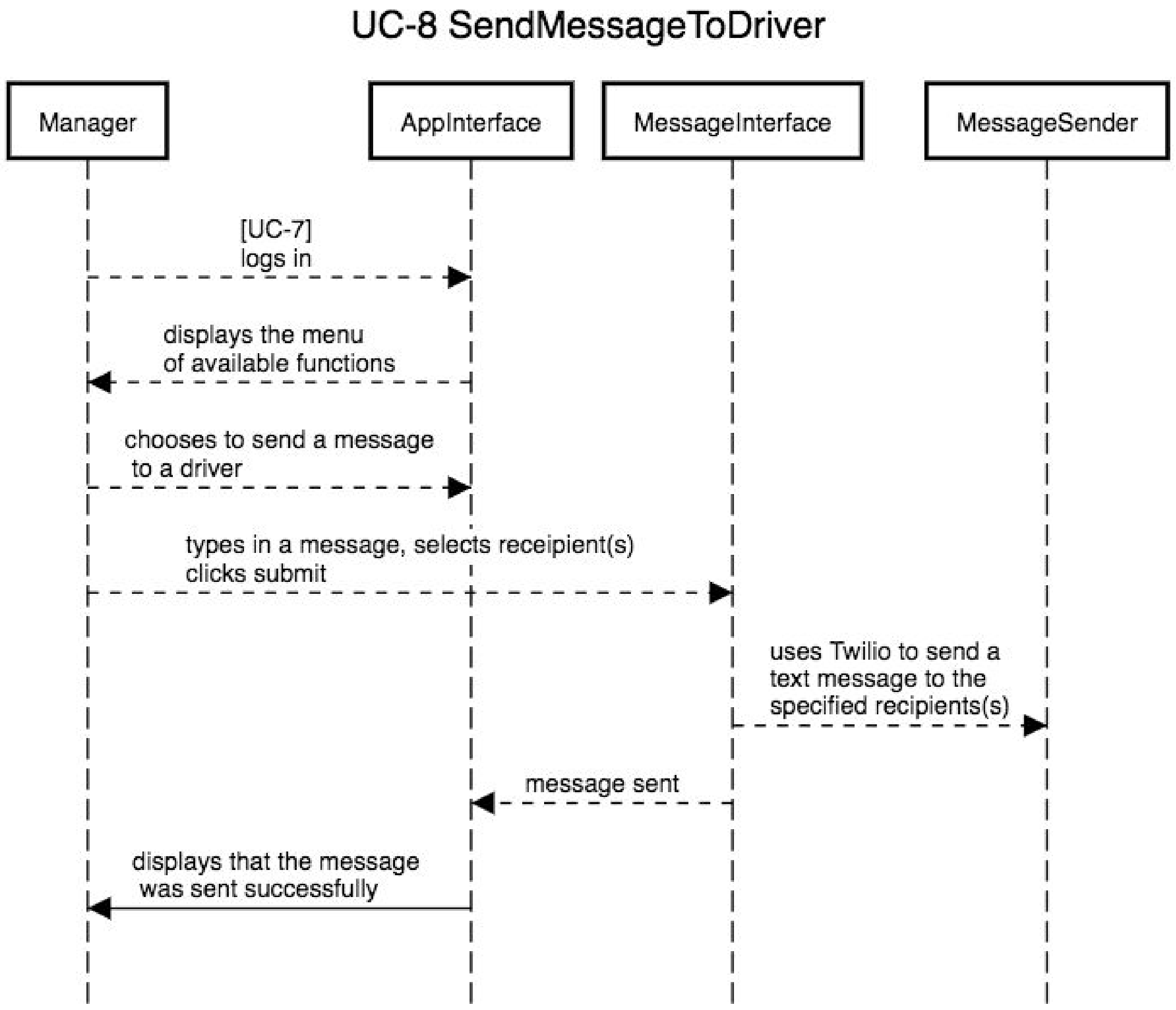


UC-7 ManagerAuthenticator



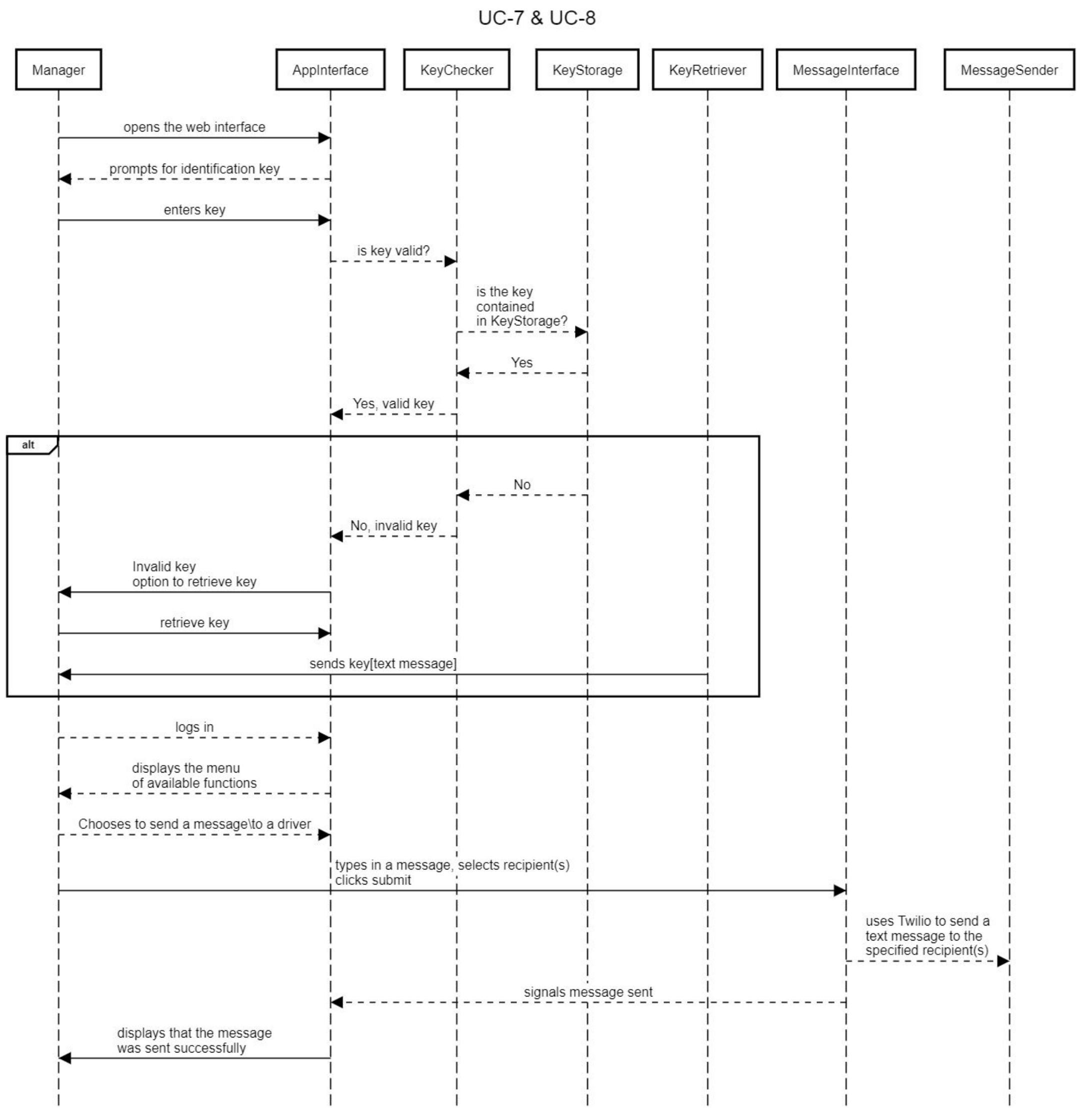
This web interface is to be used by the managers to log in to the bus management system in order to perform their day-to-day tasks. The manager plugs in the key into the AppInterface, which then uses the function KeyChecker() to check in the KeyStorage if the key is contained in the KeyStorage and if 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


UC-8 SendMessageToDriver
Manager
AppInterface
MessageInterface
MessageSender
[UC-7] logs in
displays the menu of available functions
chooses to send a message to a driver
types in a message, selects receipient(s) clicks submit
uses Twilio to send a text message to the specified recipients(s)
message sent
displays that the message was sent successfully

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

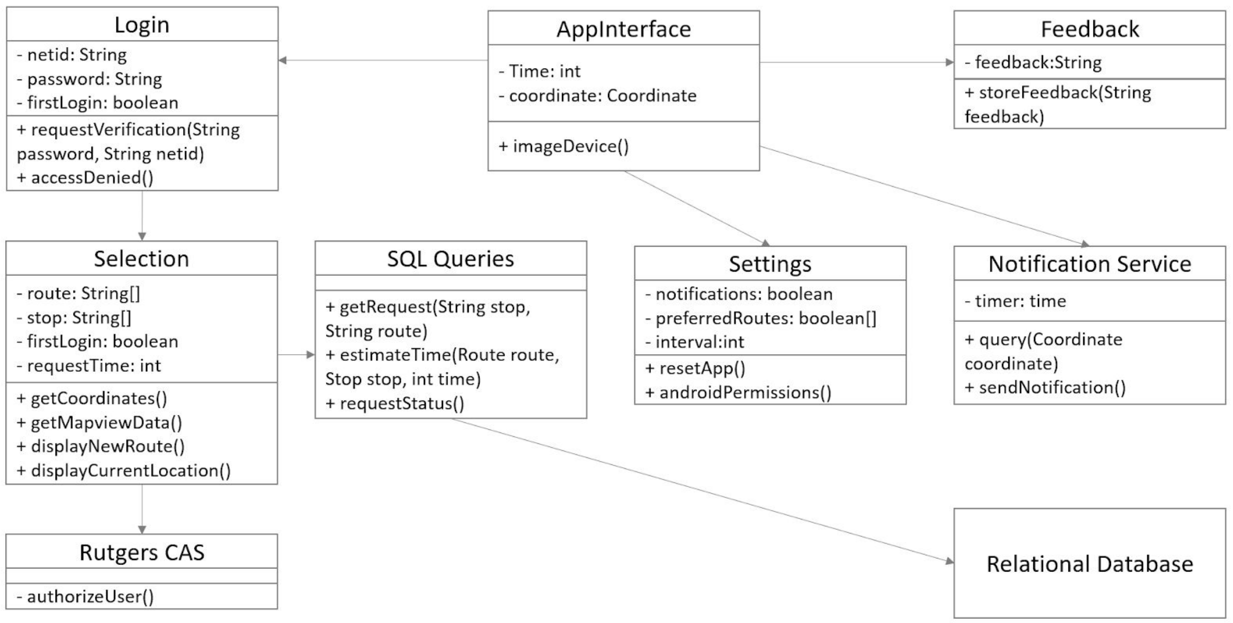

UC-7 & UC-8
Manager
AppInterface
KeyChecker
KeyStorage
KeyRetriever
MessageInterface
MessageSender
opens the web interface
prompts for identification key
enters key
is key valid?
is the key contained in KeyStorage?
Yes
Yes, valid key
alt
No
No, invalid key
Invalid key option to retrieve key
retrieve key
sends key[text message]
logs in
displays the menu of available functions
Chooses to send a message\to a driver
types in a message, selects recipient(s) clicks submit
uses Twilio to send a text message to the specified recipient(s)
signals message sent
displays that the message was sent successfully

Design Patterns

Described in Section 8.D

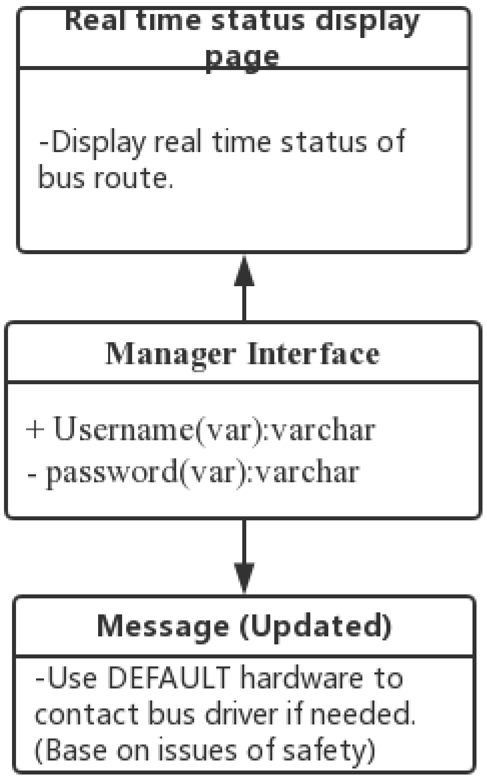
Section 8. Class Diagram and Interface Specification

A. Class diagram

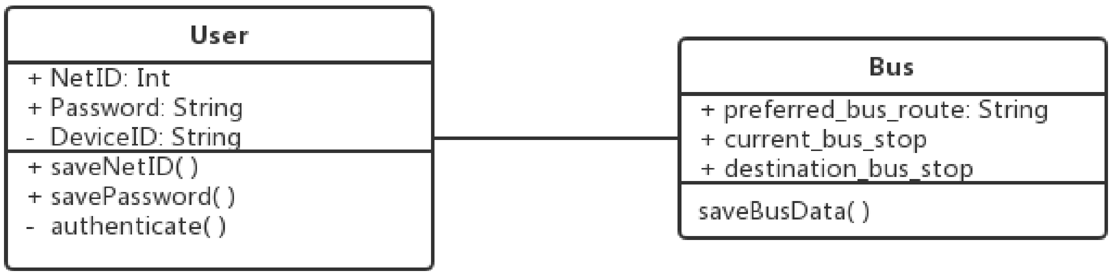


Use Case 2:
From the Login and Selection classes, the Rutgers CAS class can be reached. The user enters their NetID and password into the login screen prompt. Rutgers CAS authenticates the user's input information. The Relational Database can be reached by the Rutgers CAS. This system is used to count 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.

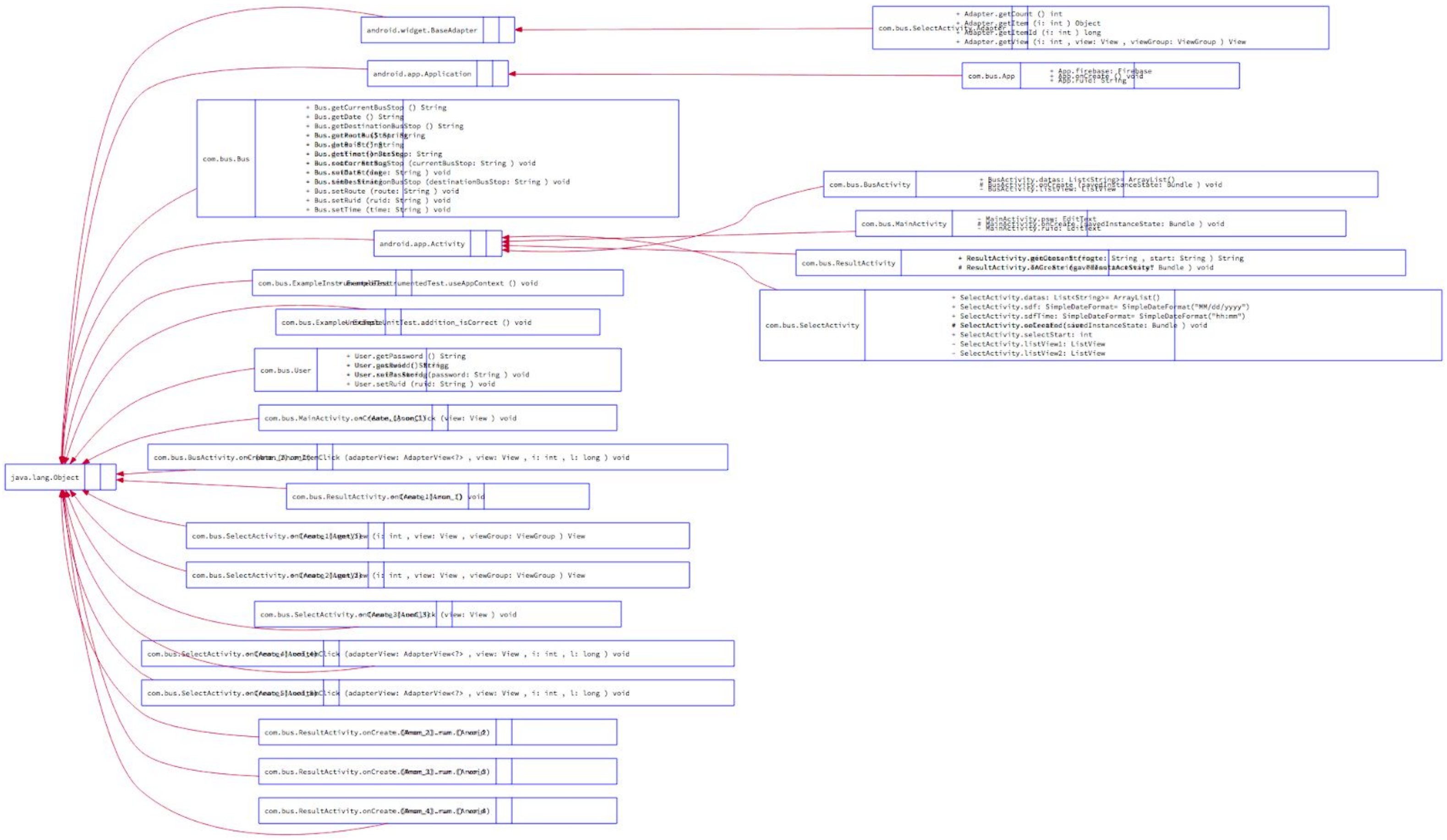
Manager Interface


Real time status display page
-Display real time status of bus route.
Manager Interface
+ Username(var):varchar
- password(var):varchar
Message (Updated)
-Use DEFAULT hardware to contact bus driver if needed. (Base on issues of safety)

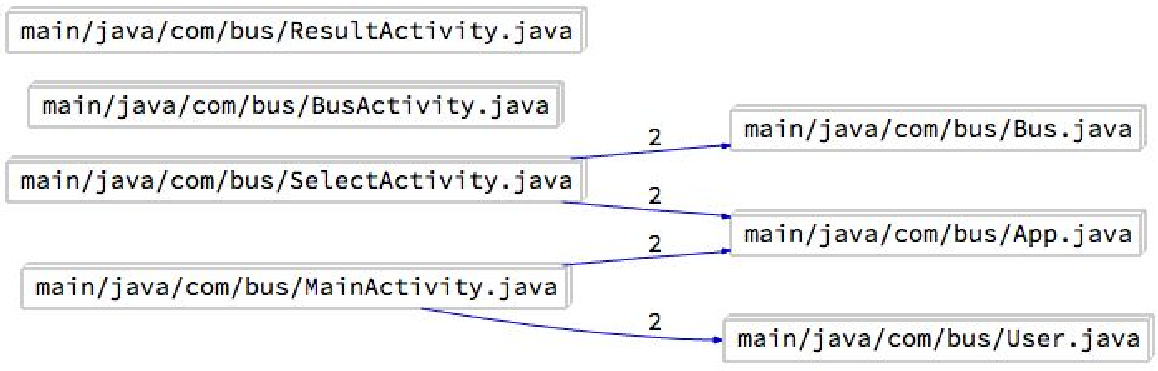
Use Case 7 & 8:
These two use cases will be implemented in the manager interface. It includes two main parts: The first part is login page which is going to identify the authenticity of the users, the second part is going to monitor the real time buses requests and send messages & signals to drivers if necessary.



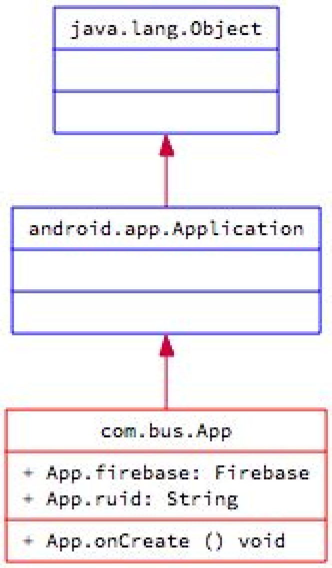
Overall android app UML diagram



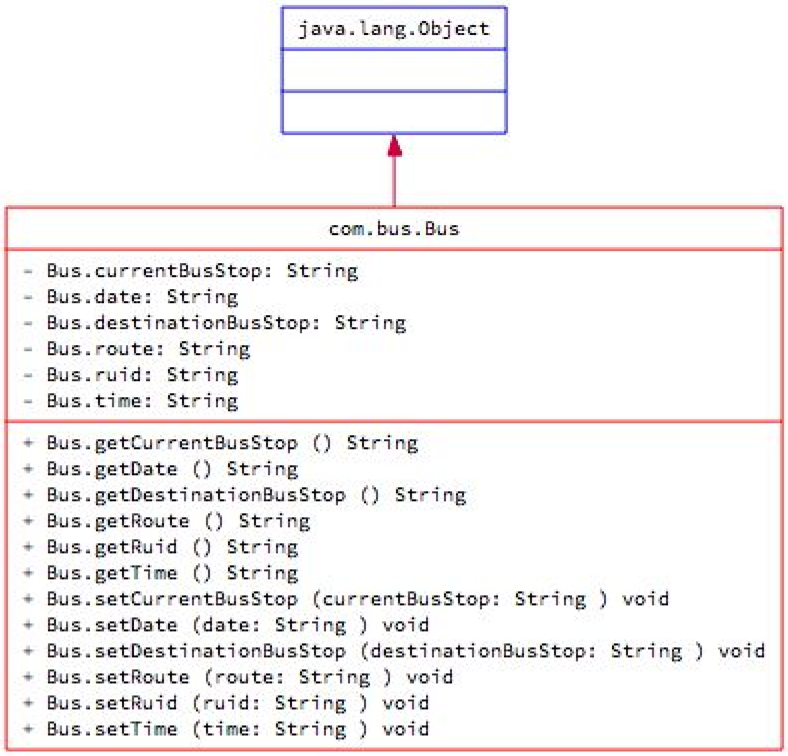
Arch Internal Dependencies


main/java/com/bus/ResultActivity.java
main/java/com/bus/BusActivity.java
main/java/com/bus/SelectActivity.java
main/java/com/bus/MainActivity.java
main/java/com/bus/Bus.java
main/java/com/bus/App.java
main/java/com/bus/User.java
2
2
2
2

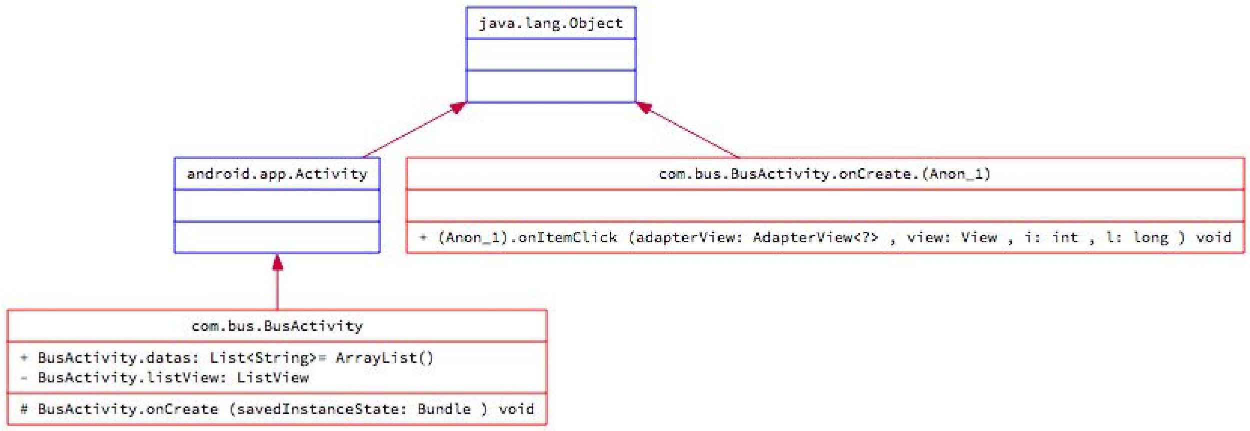
UML diagram for App.java



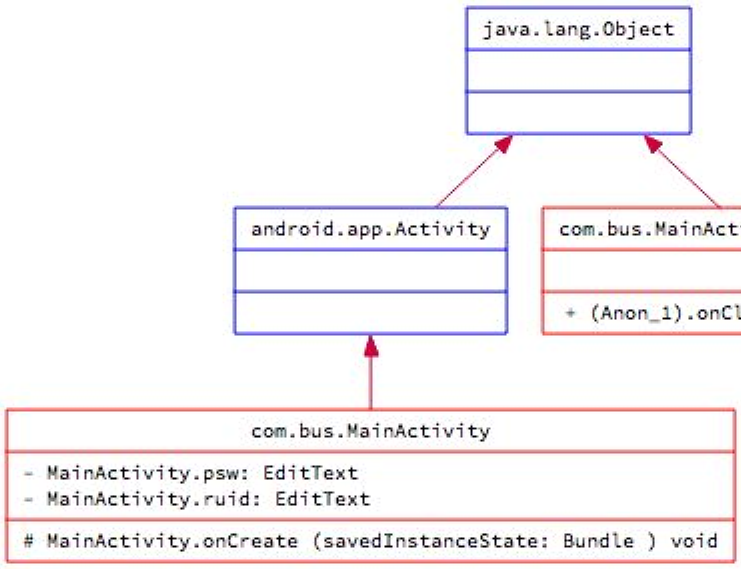
UML diagram for Bus.java


java.lang.Object
com.bus.Bus
- Bus.currentBusStop: String
- Bus.date: String
- Bus.destinationBusStop: String
- Bus.route: String
- Bus.ruid: String
- Bus.time: String
+ Bus.getCurrentBusStop () String
+ Bus.getDate () String
+ Bus.getDestinationBusStop () String
+ Bus.getRoute () String
+ Bus.getRuid () String
+ Bus.getTime () String
+ Bus.setCurrentBusStop (currentBusStop: String) void
+ Bus.setDate (date: String) void
+ Bus.setDestinationBusStop (destinationBusStop: String) void
+ Bus.setRoute (route: String) void
+ Bus.setRuid (ruid: String) void
+ Bus.setTime (time: String) void

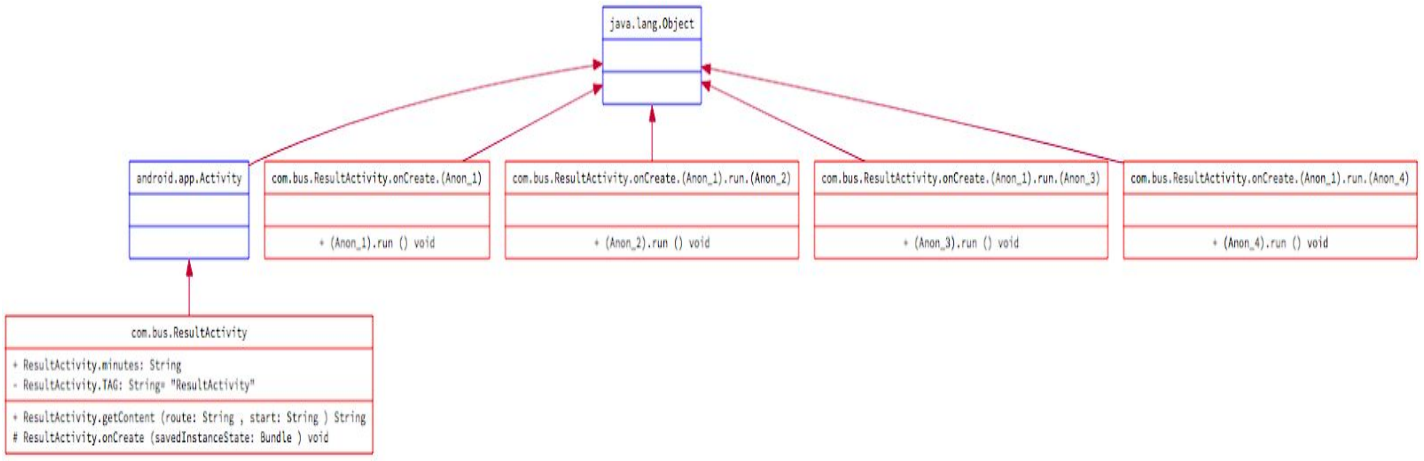
UML diagram for BusActivity.java


java.lang.Object
android.app.Activity
com.bus.BusActivity.onCreate.(Anon_1)
+ (Anon_1).onItemClick (adapterView: AdapterView<?> , view: View , i: int , l: long) void
com.bus.BusActivity
+ BusActivity.datas: List<String>= ArrayList()
- BusActivity.listView: ListView
# BusActivity.onCreate (savedInstanceState: Bundle) void

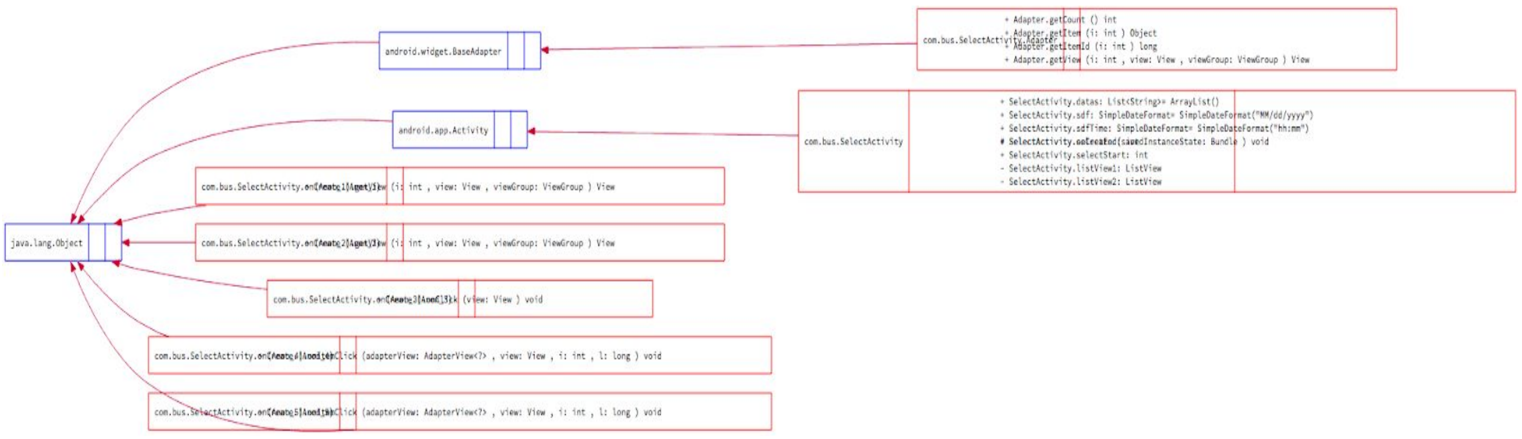
UML diagram for MainActivity.java


java.lang.Object
android.app.Activity
com.bus.MainActivity.onCreate.(Anon_1)
+ (Anon_1).onClick (view: View) void
com.bus.MainActivity
- MainActivity.psw: EditText
- MainActivity.ruid: EditText
# MainActivity.onCreate (savedInstanceState: Bundle) void

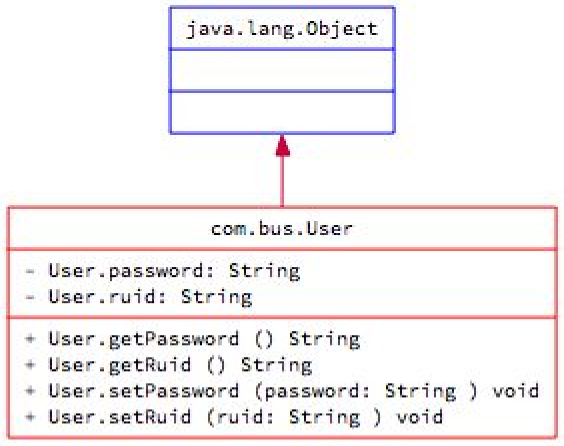
UML diagram for ResultActivity.java


java.lang.Object
android.app.Activity
com.bus.ResultActivity.onCreate.(Anon_1)
+ (Anon_1).run () void
com.bus.ResultActivity.onCreate.(Anon_1).run.(Anon_2)
+ (Anon_2).run () void
com.bus.ResultActivity.onCreate.(Anon_1).run.(Anon_3)
+ (Anon_3).run () void
com.bus.ResultActivity.onCreate.(Anon_1).run.(Anon_4)
+ (Anon_4).run () void
com.bus.ResultActivity
+ ResultActivity.minutes: String
- ResultActivity.TAG: String= "ResultActivity"
+ ResultActivity.getContent (route: String , start: String) String
# ResultActivity.onCreate (savedInstanceState: Bundle) void

UML diagram for SelectActivity.java


android.widget.BaseAdapter
com.bus.SelectActivity.Adapter
+ Adapter.getCount () int
+ Adapter.getItem (i: int) Object
+ Adapter.getItemId (i: int) long
+ Adapter.getView (i: int , view: View , viewGroup: ViewGroup) View
android.app.Activity
com.bus.SelectActivity
+ SelectActivity.datas: List<String>= ArrayList()
+ SelectActivity.sdf: SimpleDateFormat= SimpleDateFormat("MM/dd/yyyy")
+ SelectActivity.sdfTime: SimpleDateFormat= SimpleDateFormat("hh:mm")
# SelectActivity.onCreate (savedInstanceState: Bundle) void
+ SelectActivity.selectStart: int
- SelectActivity.listView1: ListView
- SelectActivity.listView2: ListView
java.lang.Object
com.bus.SelectActivity.onCreate.(Anon_1) (i: int , view: View , viewGroup: ViewGroup) View
com.bus.SelectActivity.onCreate.(Anon_2) (i: int , view: View , viewGroup: ViewGroup) View
com.bus.SelectActivity.onCreate.(Anon_3) (view: View) void
com.bus.SelectActivity.onCreate.(Anon_4) onItemClick (adapterView: AdapterView<?> , view: View , i: int , l: long) void
com.bus.SelectActivity.onCreate.(Anon_5) onItemClick (adapterView: AdapterView<?> , view: View , i: int , l: long) void

UML diagram for User.java


java.lang.Object
com.bus.User
- User.password: String
- User.ruid: String
+ User.getPassword () String
+ User.getRuid () String
+ User.setPassword (password: String) void
+ User.setRuid (ruid: String) void

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

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

D. Design Patterns

The design patterns we use in Interaction Diagrams include:

Creation patterns:

Abstract Factory: Provide interface for creating families

Builder: Separate the construction for complex objects

Structure patterns:

Adapter, Wrapper or Translator: Convert interface of a class into another interface clients expect

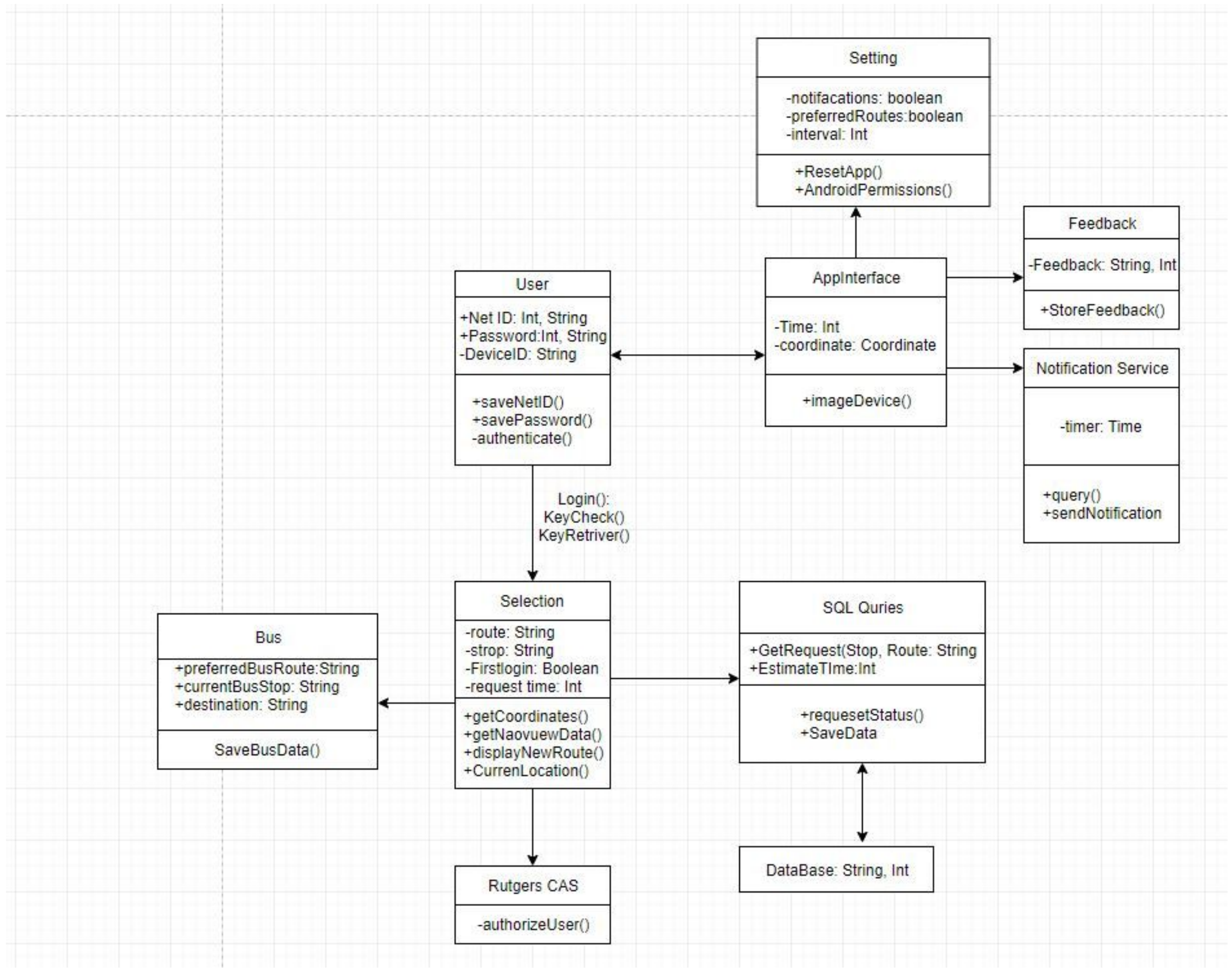
Bridge: Decouple an abstraction from its implementation

Front Controller: The design of web application

Behavioral patterns:

Chain of responsibility: Give more than one object to a chance to handle the request

E. Object Constraint Language (OCL) Contracts



Section 9. System Architecture and System Design

A. Architectural Styles

The main architectural style in this project is a 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 is the mobile application which design for the 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 the client-server model is a good fit in this case.

B. Identifying Subsystems

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

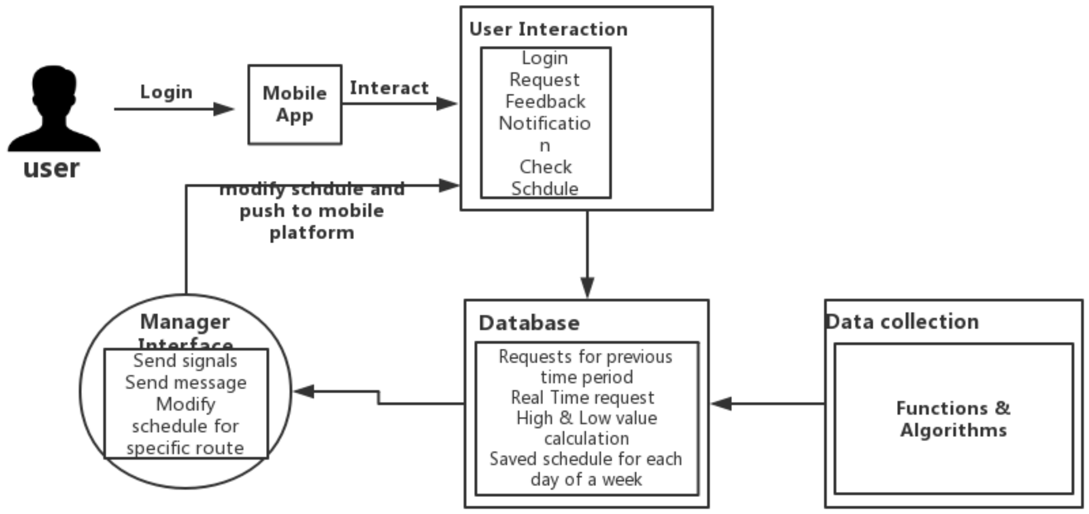
The mobile app will be the platform the users will interact with. When users approach the nearby bus-stop, they can request their preferred route. At the same time, the mobile app will collect the inputs and send feedback to manager website if any route reaches a peak value of the request. The mobile 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 a manager to decide which route may need more dispatch.

The data collection system involves the number of requests services. This subsystem also connects 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



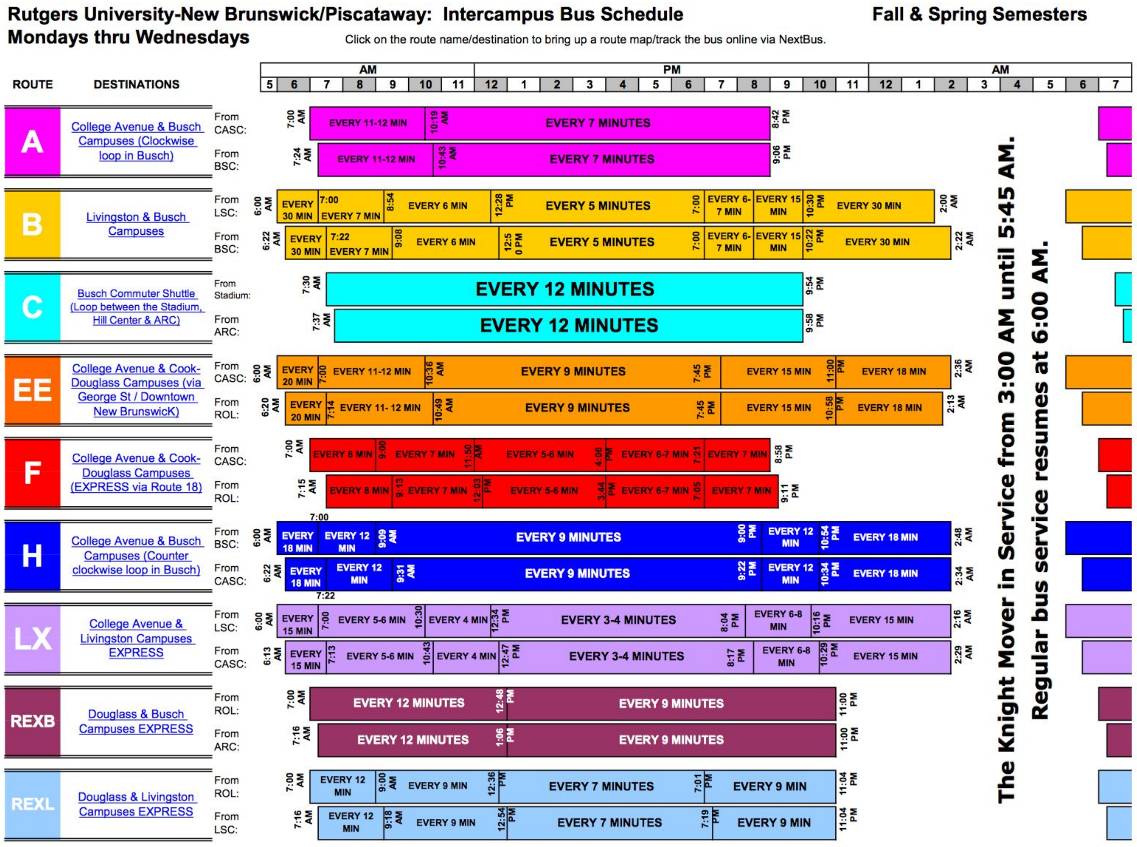
C. Mapping Subsystems to Hardware

We have three kinds of subsystem to hardware:

1. Android Applications - Run on mobile device.
2. Web Clients - run on web browser.
3. Google Firebase Service (cloud) - To storage data and implement algorithm.

D. Persistent Data Storage

Using Firebase(cloud) to store our data from users.
Bus routes and Bus schedule are stored in a relational database.

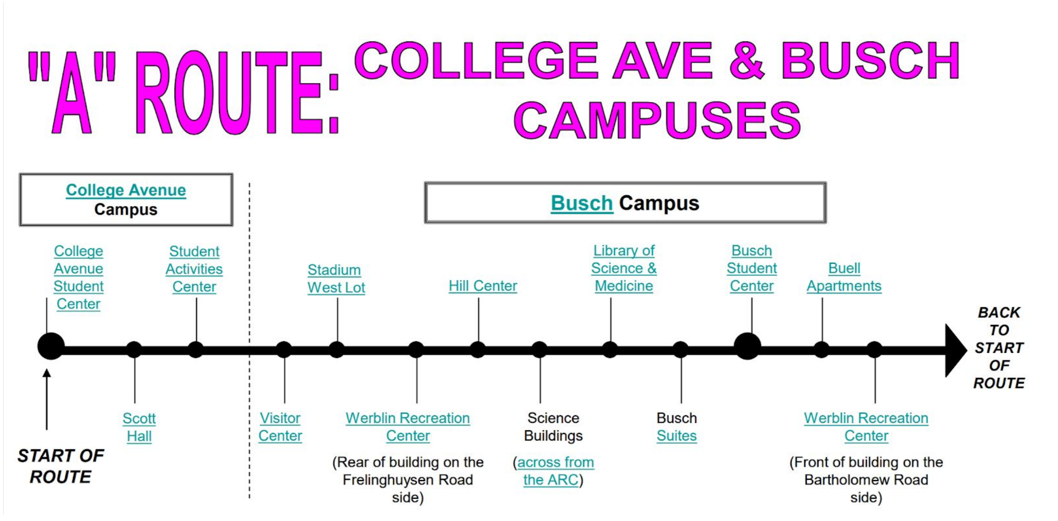


Rutgers University-New Brunswick/Piscataway: Intercampus Bus Schedule **Fall & Spring Semesters**
Mondays thru Wednesdays
Click on the route name/destination to bring up a route map/track the bus online via NextBus.

ROUTE	DESTINATIONS	From	Schedule
A	College Avenue & Busch Campuses (Clockwise loop in Busch)	From CASC:	7:00 AM · EVERY 11-12 MIN · 10:19 AM · EVERY 7 MINUTES · 8:42 PM
		From BSC:	7:24 AM · EVERY 11-12 MIN · 10:43 AM · EVERY 7 MINUTES · 9:06 PM
B	Livingston & Busch Campuses	From LSC:	6:00 AM · EVERY 30 MIN · 7:00 · EVERY 7 MIN · 8:54 · EVERY 6 MIN · 12:28 PM · EVERY 5 MINUTES · 7:00 · EVERY 6-7 MIN · EVERY 15 MIN · 10:30 PM · EVERY 30 MIN · 2:00 AM
		From BSC:	6:22 AM · EVERY 30 MIN · 7:22 · EVERY 7 MIN · 9:08 · EVERY 6 MIN · 12:50 PM · EVERY 5 MINUTES · 7:00 · EVERY 6-7 MIN · EVERY 15 MIN · 10:22 PM · EVERY 30 MIN · 2:22 AM
C	Busch Commuter Shuttle (Loop between the Stadium, Hill Center & ARC)	From Stadium:	7:30 AM · EVERY 12 MINUTES · 9:54 PM
		From ARC:	7:37 AM · EVERY 12 MINUTES · 9:58 PM
EE	College Avenue & Cook-Douglass Campuses (via George St / Downtown New BrunswicK)	From CASC:	6:00 AM · EVERY 20 MIN · 7:00 · EVERY 11-12 MIN · 10:36 AM · EVERY 9 MINUTES · 7:45 PM · EVERY 15 MIN · 11:00 PM · EVERY 18 MIN · 2:36 AM
		From ROL:	6:20 AM · EVERY 20 MIN · 7:14 · EVERY 11- 12 MIN · 10:49 AM · EVERY 9 MINUTES · 7:45 PM · EVERY 15 MIN · 10:58 PM · EVERY 18 MIN · 2:13 AM
F	College Avenue & Cook-Douglass Campuses (EXPRESS via Route 18)	From CASC:	7:00 AM · EVERY 8 MIN · 9:00 · EVERY 7 MIN · 11:50 AM · EVERY 5-6 MIN · 4:06 PM · EVERY 6-7 MIN · 7:21 · EVERY 7 MIN · 8:58 PM
		From ROL:	7:15 AM · EVERY 8 MIN · 9:13 · EVERY 7 MIN · 12:03 PM · EVERY 5-6 MIN · 3:44 PM · EVERY 6-7 MIN · 7:05 · EVERY 7 MIN · 9:11 PM
H	College Avenue & Busch Campuses (Counter clockwise loop in Busch)	From BSC:	6:00 AM · EVERY 18 MIN · 7:00 · EVERY 12 MIN · 9:09 AM · EVERY 9 MINUTES · 9:00 PM · EVERY 12 MIN · 10:54 PM · EVERY 18 MIN · 2:48 AM
		From CASC:	6:22 AM · EVERY 18 MIN · 7:22 · EVERY 12 MIN · 9:31 AM · EVERY 9 MINUTES · 9:22 PM · EVERY 12 MIN · 10:34 PM · EVERY 18 MIN · 2:34 AM
LX	College Avenue & Livingston Campuses EXPRESS	From LSC:	6:00 AM · EVERY 15 MIN · 7:00 · EVERY 5-6 MIN · 10:30 · EVERY 4 MIN · 12:34 PM · EVERY 3-4 MINUTES · 8:04 PM · EVERY 6-8 MIN · 10:16 PM · EVERY 15 MIN · 2:16 AM
		From CASC:	6:13 AM · EVERY 15 MIN · 7:13 · EVERY 5-6 MIN · 10:43 · EVERY 4 MIN · 12:47 PM · EVERY 3-4 MINUTES · 8:17 PM · EVERY 6-8 MIN · 10:29 PM · EVERY 15 MIN · 2:29 AM
REXB	Douglass & Busch Campuses EXPRESS	From ROL:	7:00 AM · EVERY 12 MINUTES · 12:48 PM · EVERY 9 MINUTES · 11:00 PM
		From ARC:	7:16 AM · EVERY 12 MINUTES · 1:06 PM · EVERY 9 MINUTES · 11:00 PM
REXL	Douglass & Livingston Campuses EXPRESS	From ROL:	7:00 AM · EVERY 12 MIN · 9:00 AM · EVERY 9 MIN · 12:36 PM · EVERY 7 MINUTES · 7:01 PM · EVERY 9 MIN · 11:04 PM
		From LSC:	7:16 AM · EVERY 12 MIN · 9:18 AM · EVERY 9 MIN · 12:54 PM · EVERY 7 MINUTES · 7:19 PM · EVERY 9 MIN · 11:04 PM

The Knight Mover in Service from 3:00 AM until 5:45 AM.
Regular bus service resumes at 6:00 AM.

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


"A" ROUTE: COLLEGE AVE & BUSCH CAMPUSES
College Avenue Campus
Busch Campus
College Avenue Student Center
Student Activities Center
Stadium West Lot
Hill Center
Library of Science & Medicine
Busch Student Center
Buell Apartments
BACK TO START OF ROUTE
START OF ROUTE
Scott Hall
Visitor Center
Werblin Recreation Center
(Rear of building on the Frelinghuysen Road side)
Science Buildings
(across from the ARC)
Busch Suites
Werblin Recreation Center
(Front of building on the Bartholomew Road side)

<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 into an application will immediately send to a database and retrieved. On the other hand, our system is also event-driven. According to the database, a 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 connects our application with service to share the information of the database.

G. Hardware and Software Requirements

- For mobile application:

User is required to have a smartphone with android 1.6 or higher, 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, 1GB disk space and access to the Internet.

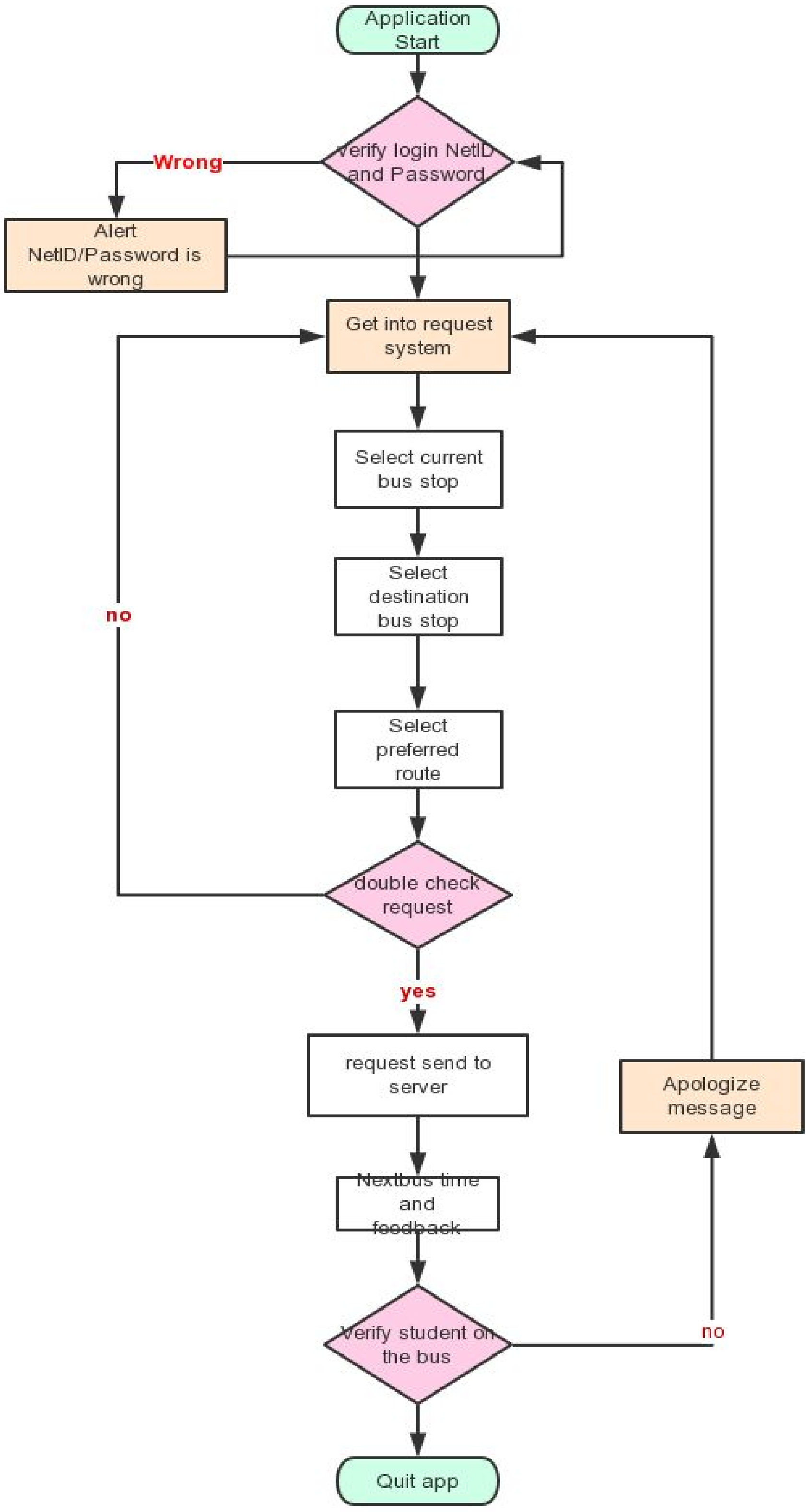
Section 10. Algorithms and Data Structures

A. Mobile App Algorithm and Data Structures

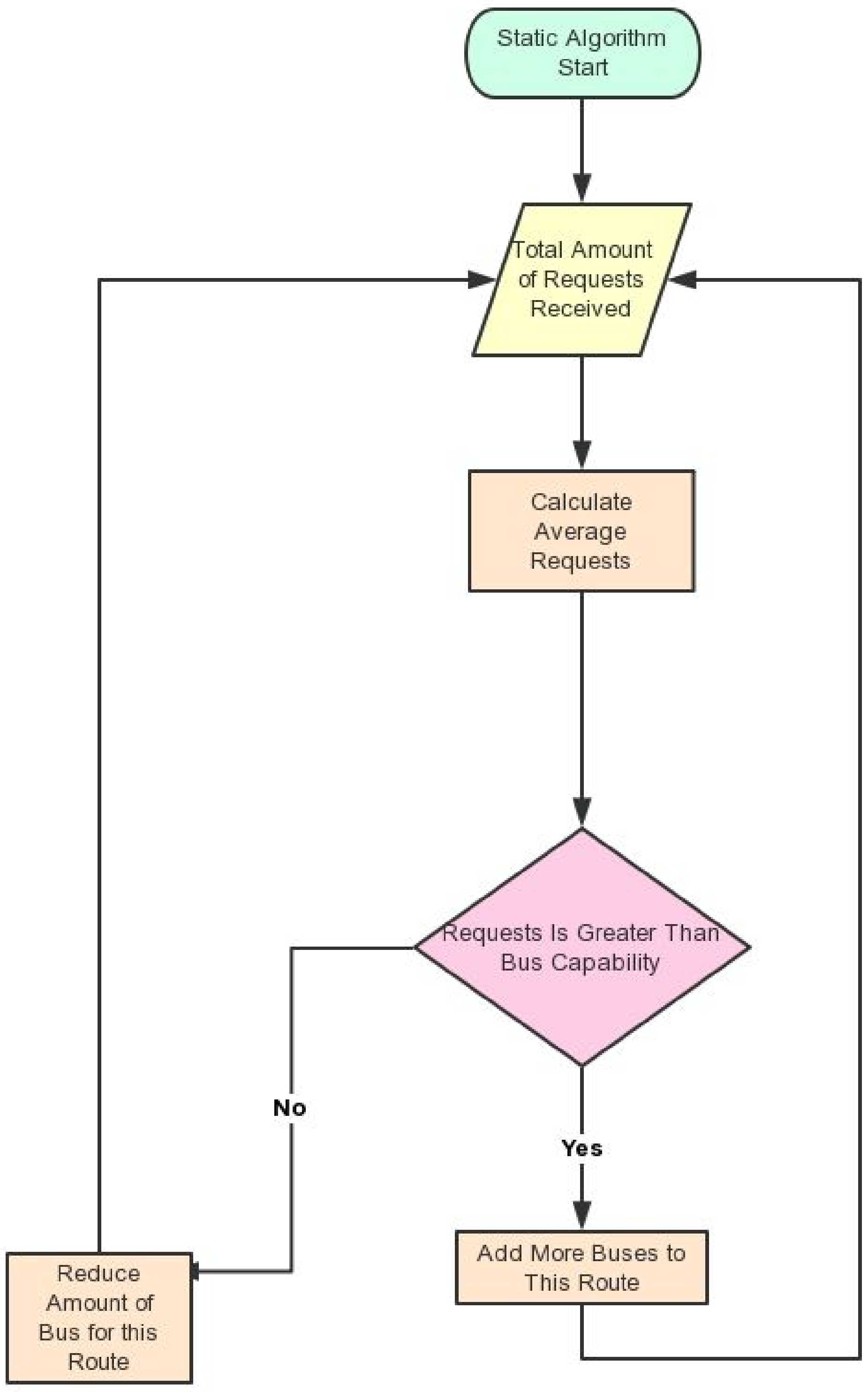
The mobile app itself does not require extensive use of many data structures since much of the data is pulled in from the server where all the data manipulation is done, however basic arrays are used. In the Values XML for the app, the list of routes and destinations are stored as hard coded string arrays 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.

Data Structure: JSON tree

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.

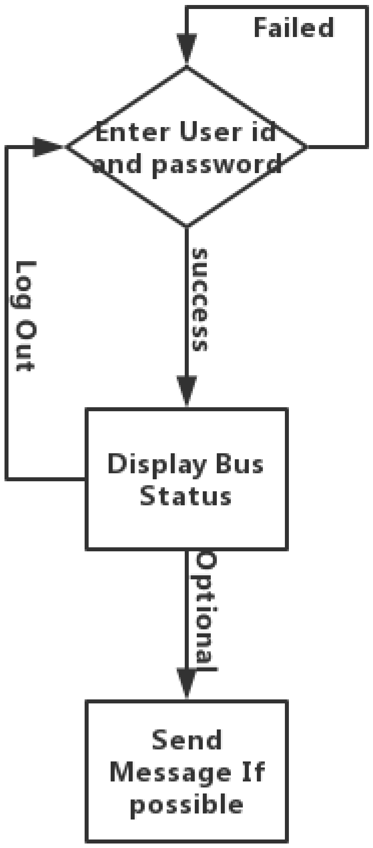


B. Server Side Algorithm and Data Structure


Static Algorithm Start
Total Amount of Requests Received
Calculate Average Requests
Requests Is Greater Than Bus Capability
No
Yes
Reduce Amount of Bus for this Route
Add More Buses to This Route

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. The web is constructed by HTML and CSS. The first part is only static web (demo 1), real time status will try to be implemented in the final project.



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

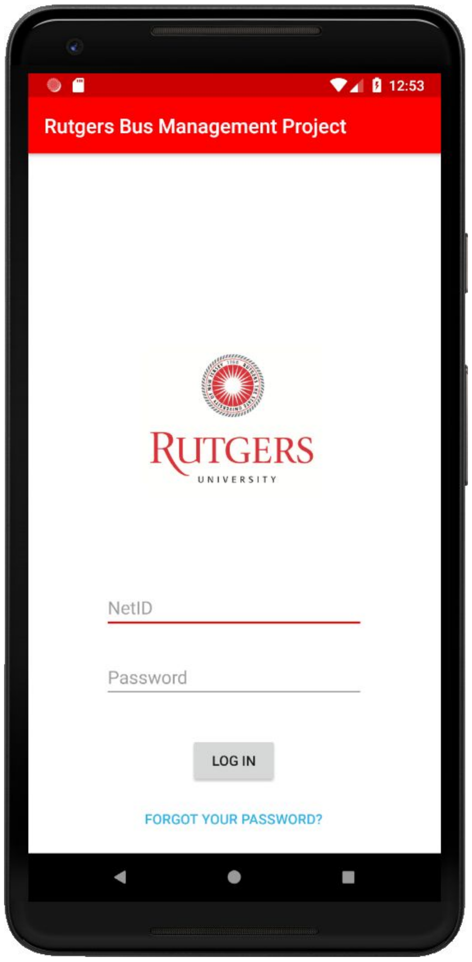
Part 2 Schedules

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

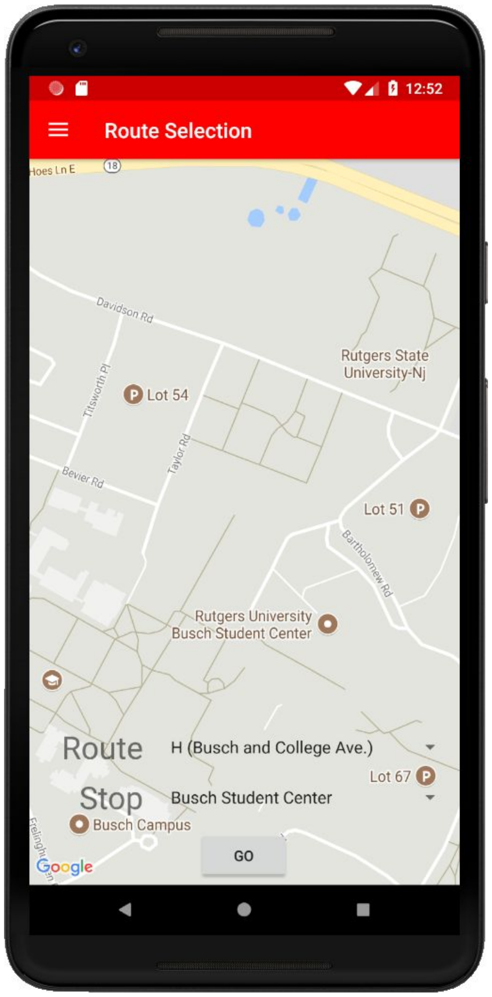
Section 11. User Interface Design and Implementation

A. Mobile App Interface

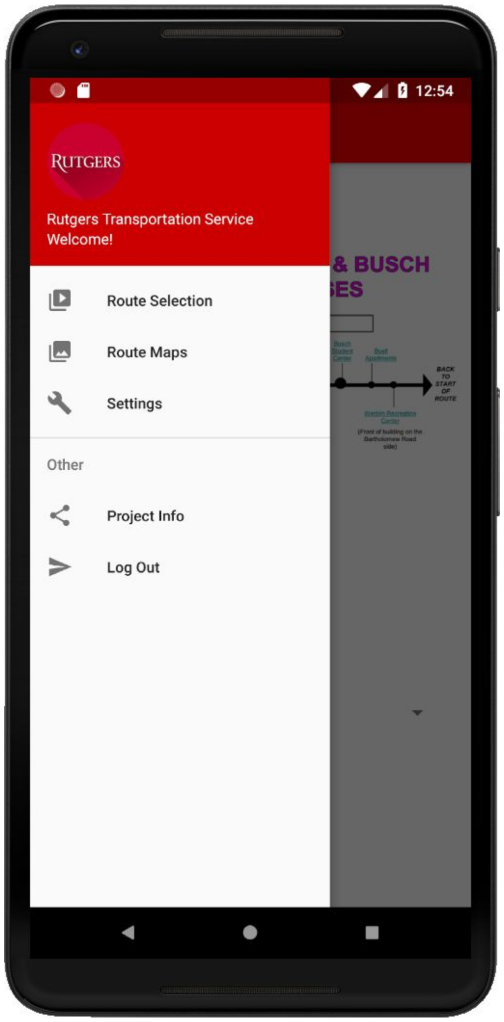
Since the mobile Android app is being developed through Android Studio (now SDK 27 for Android version 8.1), some alterations were made to the UI design in order to implement a few of Google's official Material Design guidelines. A basic guide of this can be found at the URL [https://material.io/guidelines/]. 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.

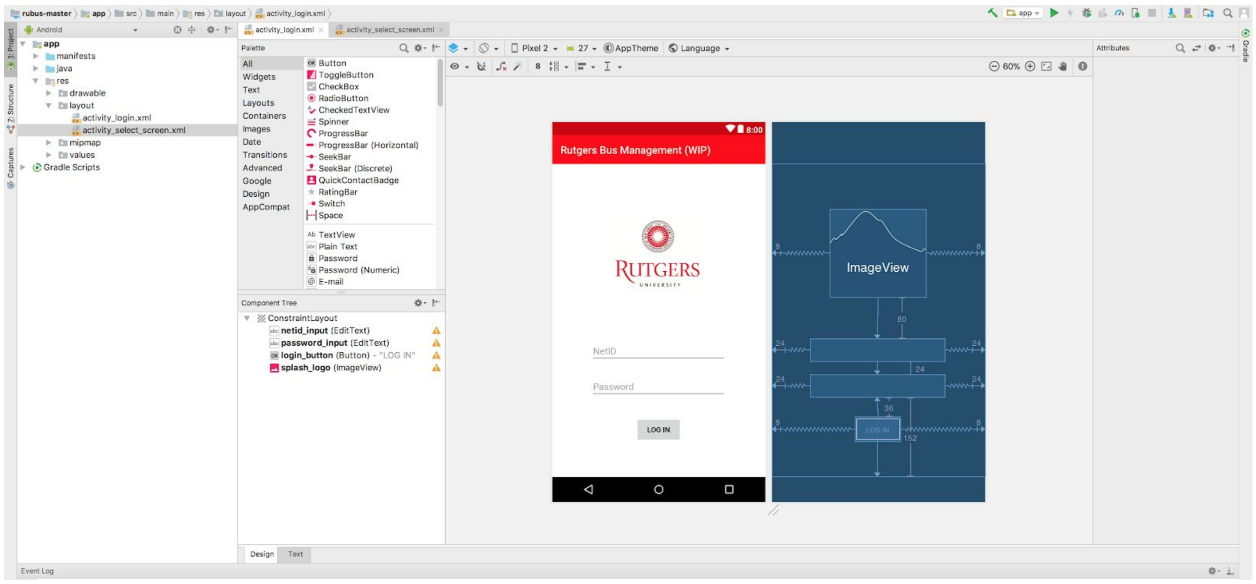


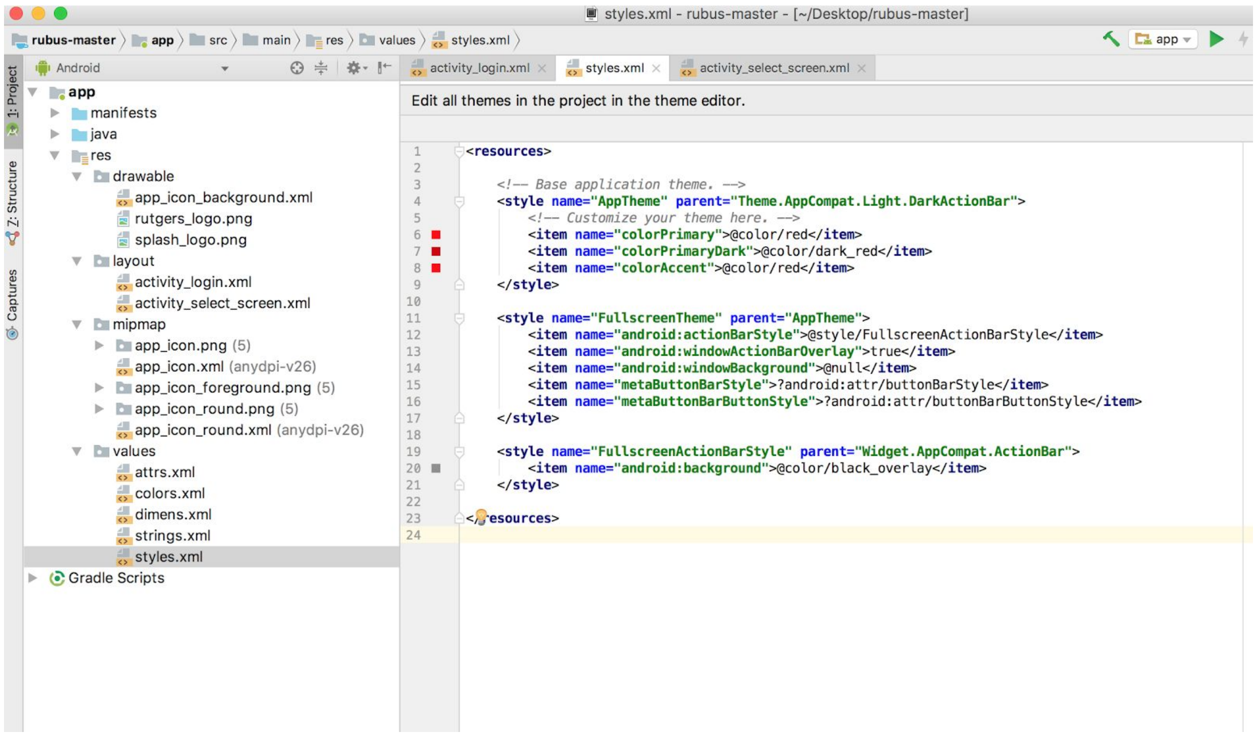
Another change that was made was for the placement of the settings activity. Initially, we were going to access the settings screen through a discrete button placed somewhere near the greeting screen, but this would mean a user would need to navigate back to the main screen whenever they wish to edit 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.

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.

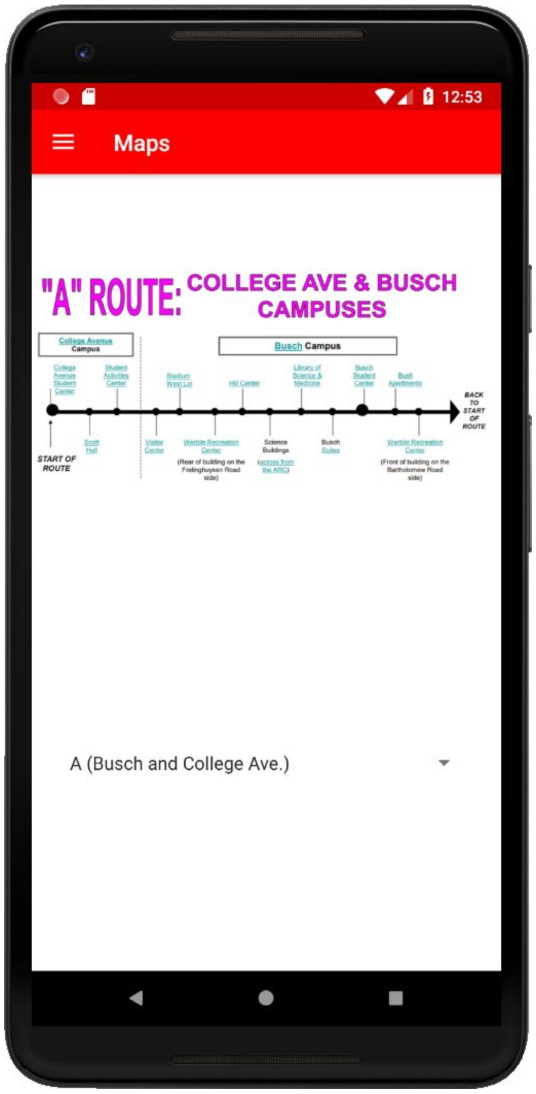

rubus-master › app › src › main › res › layout › activity_login.xml
activity_login.xml
activity_select_screen.xml
Palette
Component Tree
ConstraintLayout
netid_input (EditText)
password_input (EditText)
login_button (Button) - "LOG IN"
splash_logo (ImageView)
Rutgers Bus Management (WIP)
RUTGERS
NetID
Password
LOG IN
ImageView
Design
Text


styles.xml - rubus-master - [~/Desktop/rubus-master]
rubus-master › app › src › main › res › values › styles.xml
app
manifests
java
res
drawable
app_icon_background.xml
rutgers_logo.png
splash_logo.png
layout
activity_login.xml
activity_select_screen.xml
mipmap
app_icon.png (5)
app_icon.xml (anydpi-v26)
app_icon_foreground.png (5)
app_icon_round.png (5)
app_icon_round.xml (anydpi-v26)
values
attrs.xml
colors.xml
dimens.xml
strings.xml
styles.xml
Gradle Scripts

Edit all themes in the project in the theme editor.

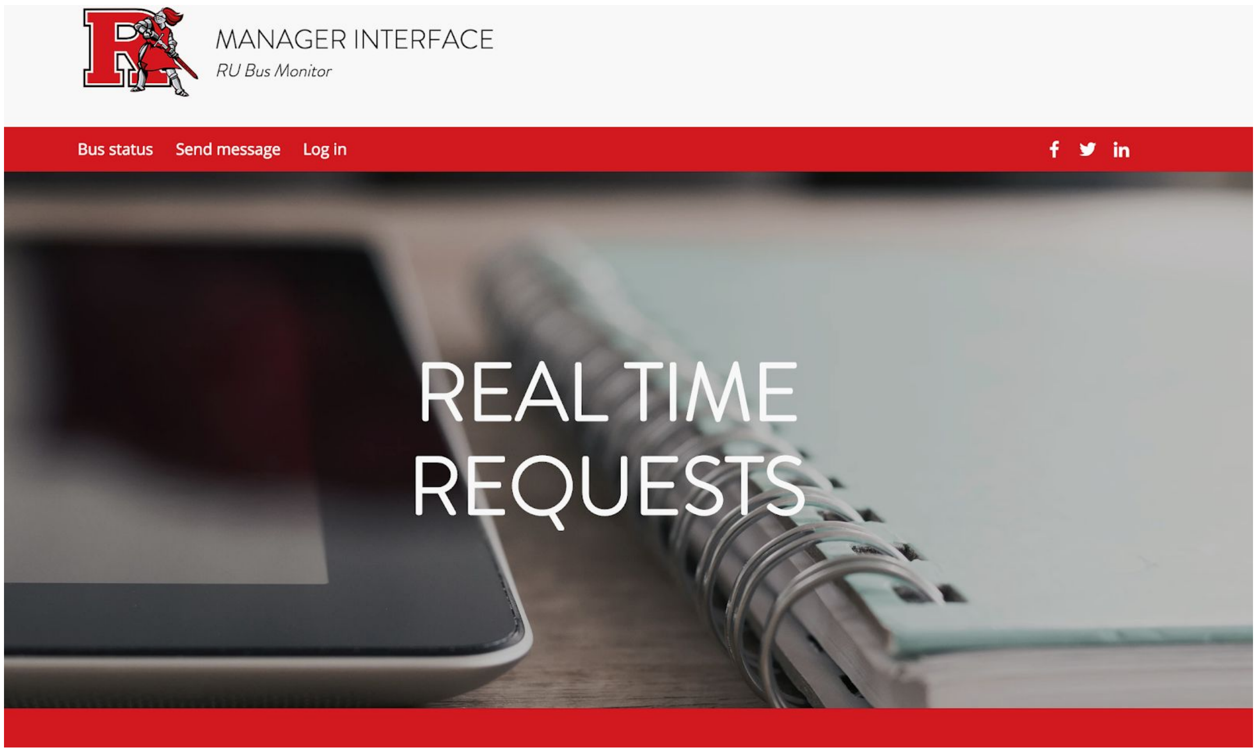
```xml
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/red</item>
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/red</item>
    </style>

    <style name="FullscreenTheme" parent="AppTheme">
        <item name="android:actionBarStyle">@style/FullscreenActionBarStyle</item>
        <item name="android:windowActionBarOverlay">true</item>
        <item name="android:windowBackground">@null</item>
        <item name="metaButtonBarStyle">?android:attr/buttonBarStyle</item>
        <item name="metaButtonBarButtonStyle">?android:attr/buttonBarButtonStyle</item>
    </style>

    <style name="FullscreenActionBarStyle" parent="Widget.AppCompat.ActionBar">
        <item name="android:background">@color/black_overlay</item>
    </style>

</resources>
```

The current version of our app now features pictures of the map layouts from the RUDOT website for basic reference of the routes for new students as well as a full screen variant of the Google Maps overlay instead of a small window.

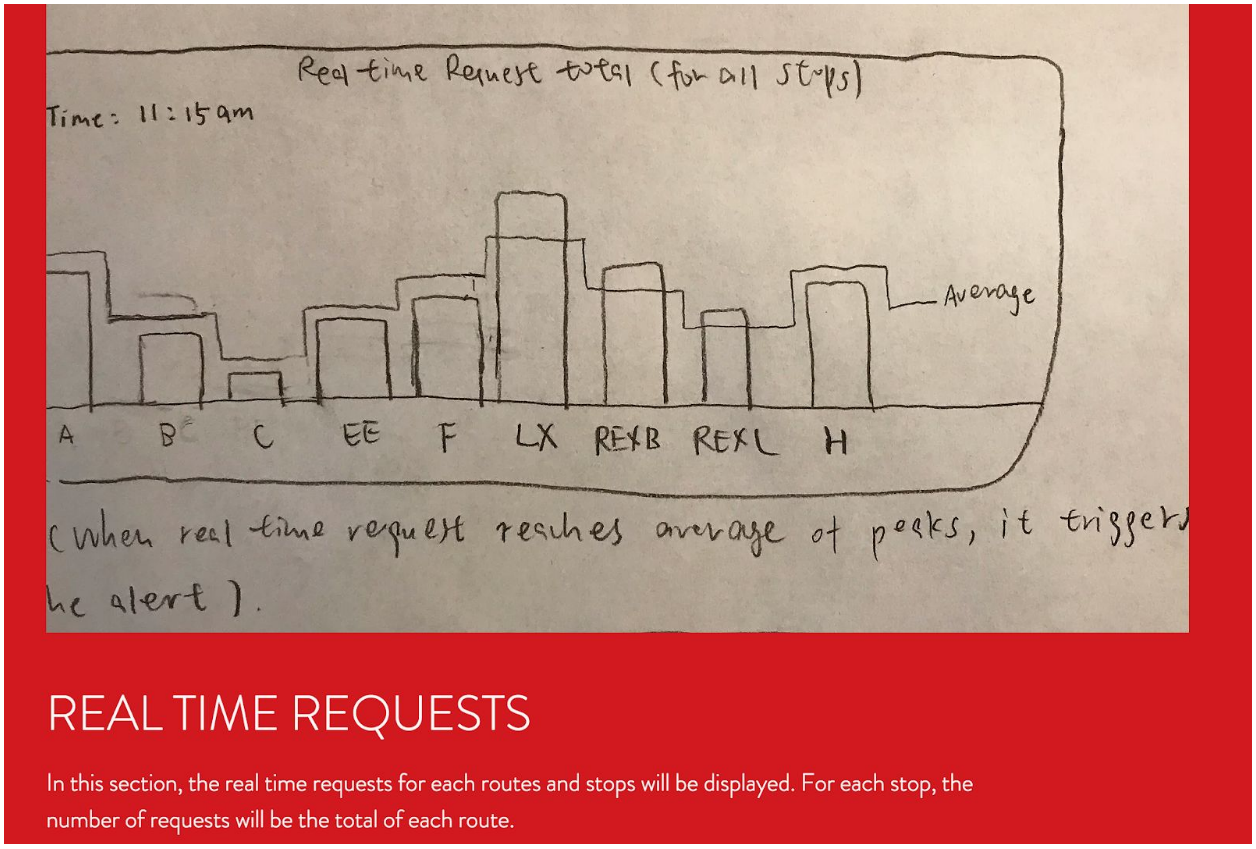

12:53
Maps
"A" ROUTE: COLLEGE AVE & BUSCH CAMPUSES
College Avenue Campus
Busch Campus
College Avenue Student Center
Student Activities Center
Stadium West Lot
Hill Center
Library of Science & Medicine
Busch Student Center
Buell Apartments
BACK TO START OF ROUTE
START OF ROUTE
Scott Hall
Visitor Center
Werblin Recreation Center
(Rear of building on the Frelinghuysen Road side)
Science Buildings
(across from the ARC)
Busch Suites
Werblin Recreation Center
(Front of building on the Bartholomew Road side)
A (Busch and College Ave.)

B. Manager Interface

The original design is shown below: (Final version will be shown on DEMO2)



* Hand Drawn Mockup



(when real time request reaches average of peaks, it triggers
he alert).

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.

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 12. 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 check real time status of bus routes.
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. Otherwise, the web should stay at the current page and notify users to input the correct data. Also, the test passes if manager can visualize the real time status of bus route. (Loading chart successfully)

Section 13. History of Work, Current Status, and Future 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.

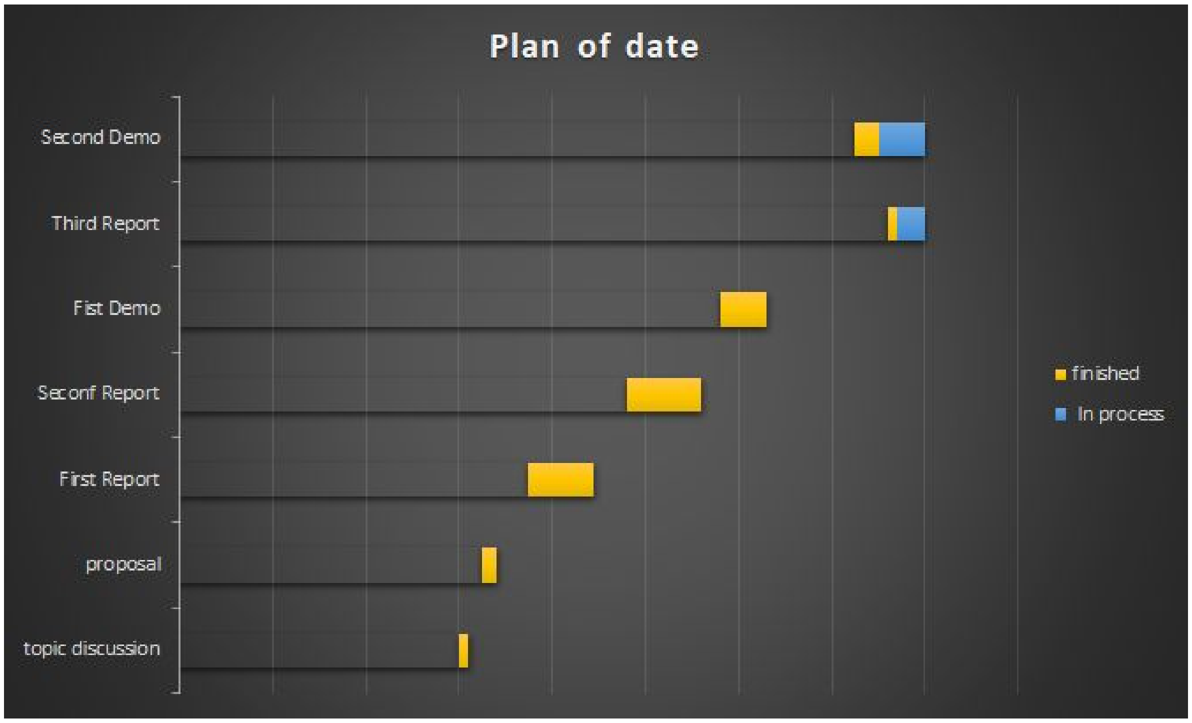
Problems	Solution
Procrastination	Changed the working due date of report on Saturday night and schedule voice meeting to debug our report on Sunday
Lack of communication (We have 75% native Chinese speaker; language issues)	It is required to discuss project in English and we work more hard on grammar 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. History of Work, Current Status, and Future Work

Previous goals → Initial Plan of work



A.Android Development:	Aaron Hu, Haowei Li
B.Manager Web Development:	Viraj Patel, Yufeng Lin
C.Backend Development (Bus interaction):	Yaocheng Tong, Lu Jin, Minghao Qin,Yuhai Zhang

Milestone	Original Deadline	Date Completed	Initial subroup
Mobile Application	3/1	3/2	Aaron Hu, Haowei Li
Web Application	3/3	3/3	Viraj Patel, Yufeng Lin

Database implementation	3/15	3/16	Minghao Qin,Yuhai Zhang
Integration	3/25	3/25	Yaocheng Tong, Lu Jin
Correction and debug	4/27	4/27	All member

Final history of work:

A.Android Development:	Aaron Hu, Haowei Li
B.Manager Web Development:	Yufeng Lin
C.Backend Development (Bus interaction):	Yaocheng Tong, Lu Jin, Minghao Qin,Yuhai Zhang

Milestone	**Date completed**	**Subgroup**
Web application	3/2	Yufeng Lin
UI Design	3/3	Aaron Hu, Haowei Li
The first algorithm: For number of users	3/20	Minghao Qin,Yuhai Zhang
The second algorithm: For static schedule	3/25	Yaocheng Tong, Lu Jin

Since we had a few days of bad weather, we couldn't meet face to face and got some delays, but we use online voice meeting to manage and complete our core objectives.

Current accomplishments

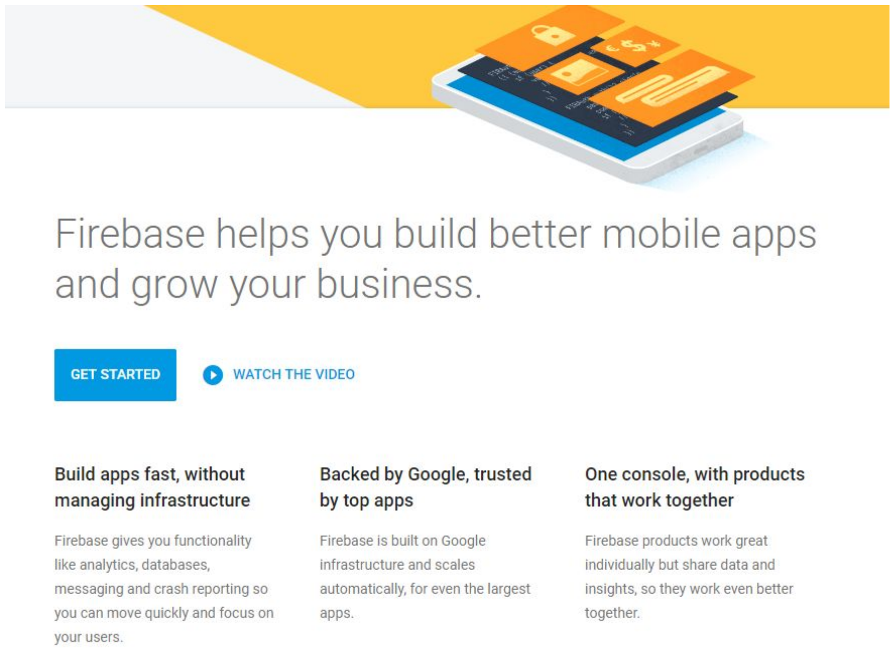
1. locations of bus stops' map display
2.Mobile bus request report
3 static Algorithm

Future work

In the first demo, we focus on the mobile application for request bus and finished with the process of figuring out what data will be required from which sources. Since the number of people in the team has changed, our web application has not been completely developed yet. Therefore, for the second demo, we will not only optimize app performance, but also pay more attention on web development. 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. Furthermore, we will try to use machine learning that explore the construction and study of algorithms by building a model from our input data.

D. 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.



Firebase helps you build better mobile apps and grow your business.

GET STARTED WATCH THE VIDEO

Build apps fast, without managing infrastructure

Firebase gives you functionality like analytics, databases, messaging and crash reporting so you can move quickly and focus on your users.

Backed by Google, trusted by top apps

Firebase is built on Google infrastructure and scales automatically, for even the largest apps.

One console, with products that work together

Firebase products work great individually but share data and insights, so they work even better together.

Section 14. 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
7. *Relationship between poisson and exponential distribution.* Apr 28, 2018. Retrieved from:https://stats.stackexchange.com/questions/2092/relationship-between-poisson-and-exponential-distribution
8. Poisson and exponential distributions. Apr 28, 2018 Retrieved from: http://www.kellogg.northwestern.edu/faculty/weber/decs-430/Notes%20on%20the%20Poisson%20and%20exponential%20distributions.pdf

Project Management

Described in Section 13.