Final Report

Real Time Bus Tracking App

University Of British Columbia Okanagan

COSC 499

Table of Contents

Identification

Name of Project: Bus Tracking App

Sponsor: Fremtid Media

Project Stakeholders

Stakeholder Names	Roles
Fremtid Media	Sponsor/Client
Reza Afzali	Client
Simranpal Bains	Client
Scott Fazackerley	Instructor/Supervisor
UBCO Students/ Residents of Kelowna	Users
COSC Capstone Team 12	Android App Developers
Engineering Capstone Team	Hardware/Infrastructure Team
University of British Columbia	Academic Institution
Kontakt.io	Beacon Provider
Google	Android OS Provider
Here	Maps API Provider for Showing Bus Location on the App

Team

Team Members	Roles	Responsibilities
Wasek Habib	Product Manager	- Primarily responsible for the project workflow, requirements engineering, software architecture, and ensuring SDLC. - Involved in design and development process.
Ini Oladosu	Technical Lead	- Responsible for the non coding documents - Responsible for UI Design of application - Involved in development process
Matthew De Leeuw	Integration Lead	- Responsible for checking code before integration into master branch (aka) responsible for what gets deployed - Responsible for programming with team on seperate features
Trevor Gallicano	DevOps Lead	- Responsible for continuous integration/deployment - Responsible for automated testing - Involved in development process
Kyle Rennie	Client Liaison	- Responsible for continuous client contact and logging conversation - Involved in development process

Authors

Document Owner(s)	Contact
Wasek Habib	wasek.edu@gmail.com
Matthew De Leeuw	mattdeleeuw@hotmail.com
Trevor Gallicano	trevorg@hotmail.ca
Ini Oladosu	inioladosu@gmail.com
Kyle Rennie	kyrenzie@gmail.com

Document History

Version	Date	Document Revision Description
1.0	April 14, 2019	Initial document

Terminology

Item	Description
Beacon	A low-energy Bluetooth device used to determine a device’s location. Developed by Kontakt.io
Continuous Integration (CI)	Development strategy used to automate integration, testing, and deployment.
Espresso	Java based testing API used to test UI of Android applications
Integration Test	A test that verifies interfaces (interactions) between two or more modules.
Module	A specific function or piece of the software system.
System Test	A test that verifies functionality of the completed system.
Test-Driven Design (TDD)	Design strategy that has tests created before development of a module.
Unit Test	A test that verifies the functionality of the smallest unit of code. The smallest unit will be referred to as a module.
User Interface (UI)	The graphical layer which allows the user to interact with the software system.
REST API	(Application Programming Interface): Tool that helps to communicate between two programs using HTTP protocols.

Scope and Charter

Document History

Version	Date	Document Revision Description
1.0	October 23, 2018	Initial document
2.0	November 20, 2018	Updated to match changes
3.0	December 5, 2018	Changes to requirements
4.0	April 16, 2018	Revision for final report

Project Purpose

Currently there is a ridership of 4.9 million in Kelowna per year, a fair amount of riders (>1%) use the transit app, with this app we are looking to take a portion of the demand for maps and transit for this app. Current apps have issues with telling the user the real time of their bus arrival because they are unreliably crowdsourced, or just use the bus schedule for times. The Transit App has a “GO” function but this is not always available. Bus drivers also want to maximize efficiency but they don't have a way to know if passengers are at the upcoming stops. Another problem transit users have is that not every bus stop has audible notifications. The purpose of the project is to provide a reliable solution to the commuters for tracking their desired busses in real time, passengers waiting at the bus stop to be picked up with an android application, and an audio notification of the upcoming bus to the people waiting at the bus stop. If the project is not completed on time then the company overseeing it can chose to continue with a new or the same team. They would have prototypes and documentation to help them continue with the project, even with a new team.

Project Objectives

Create a solution for commuters to track their desired bus in real time using an android app. Passengers waiting at a bus stop should be confident that the bus they are waiting for will pick them up. Passengers waiting at a bus stop who are not paying attention or maybe are visually impaired can have audio notifications from a speaker on the bus stop. A light sensor on the bus will notify the bus driver, if there's a passenger waiting at the bus stop and all the passengers have boarded.

Project Description

The project has two parts: Hardware and Software. COSC team is responsible for anything related to the server, development and deployment of the android application and its connection to the beacons (software side). The android app (proof of concept) will show the user live location and arrival time of the bus (real-time bus schedule) in nearby bus stops. The user will get a push notification of the remaining arrival time of the subscribed bus number. The beacons will be installed and configured on a bus/car and at bus stops by the engineering team. The engineering team is also responsible for small single board computer (Raspberry Pi) on the bus. The cloud server will send information (bus stop location) of passenger tracking a bus to internet enabled small board computers. The computers will then communicate with the light sensors and speaker and make some notification. (done by the engineers). If the user clicks on the push notification, it will take him/her to the app and show the live location of the bus on the map. The users will not need an account to use the application. Personal information of the users will not be asked or stored. However, their travel time and distance will be stored in a database anonymously.

Success Criteria

- The app will have simple and elegant user experience following standard human computer interaction rules.
- User satisfaction will be measured by a standardized set of questions followed by the usability test and will measure 3 or higher on a 5 point scale.
- The user interface will follow the best UI practices and 'material design' standards.

Out of Scope

- The engineering team is responsible for setting up and configuring beacons and programming small single board computers.
- Receiving data from the server to the small board computers and communication between small computers and speakers and light sensors are also EE team’s responsibility.
- Range of beacon.
- Accuracy of location tracking will not meet any standard
- Data protection and transmission security of the beacons. Kontakt.io is responsible for secured data transmission of the beacons.
- The coverage area will not be outside of UBCO roundabout and EME bus stop.
- Setting up the beacons, checking physical conditions and health.
- Creating mesh network.
- People with strollers or wheelchairs (physically impaired).
- Setting up and working with speakers and displays on bus-stops.
- Any communication between speakers, beacons, and light sensors.
- Any kind of hardware implementation, security, and maintenance, and programming the small board computers are done by the engineering team.
- Cost analysis.
- Wayfinding (Navigation and in-between stops).

Functional Requirements

- Android Application will ask for user’s permission to access location, and to store information.
- Android Application will display a map with the user location.
- App will let the user to track a bus/car.
- Android Application will display a map with the tracked bus location.
- Application will display the expected arrival times of the selected bus.
- Application will receive information from the nearby beacons.
- Application will send nearby beacon information to the server.
- User will receive a push notification when the selected bus has arrived at the stop.
    - Notification sent to phone from AWS
- Application will be available to all users with no registration.
- Application will show a list of nearby bus/car that has/have LTE enabled small board computer with GPS tracker board.

- App will track travel distances and travel times of a user.
- Admins will be able to view data stored in the online database hosted in AWS.
- The backend (rest apis and database) will be deployed on AWS.
- Cloud server will get user location when he/she is in the beacon range of the bus location.
- Cloud server will receive beacon information from the application.
- Cloud server will receive user selected bus stop and time reminder from the application.
- Cloud server will perform some computations to check if the user is on the bus.
- Cloud server will perform some computations to check if the user is out of the bus.
- Cloud server will send tracked bus location to an LTE enabled computer at bus stop/bus.
- Cloud server will store user's travel time anonymously in database.
- Cloud server will store user's travel distance anonymously in database.
- Cloud server will send tracked bus location to the application.
- Cloud server will send bus stop location of the passenger to an LTE enabled computer on the bus.
- Cloud server will get data of the live position of busses from small board computers.

Incomplete Functional Requirements

- User will receive a push notification when the selected bus has arrived at the stop.
    - Push notification is not sent automatically, but push notifications manually sent work as intended
- App will track travel distances and travel times of a user.
    - App does not store travel data. Travel data is instead sent straight to the server

Non-Functional Requirements

Development

- Developed as an Android application written in java.
- Development process will be completed using the agile method.
- Application will use third-party API's.

- Costs for development (such as cloud service or third party API’s) will be covered by the client.

Performance

- User must be connected to the internet.
- Software will handle at least 2000 users.
- Software will be scalable up to the entire 97 bus route in Kelowna.
- Bus position will be updated within 3 seconds (including screen latency).
- Application and small board computers use http calls (REST APIs) to communicate with the server.
- Bus tracking will only show information related to the next bus(es).
- App will not track location in the background

Incomplete Non-Functional Requirements

- Software will handle at least 2000 users.
    - Requirement cannot be tested
- Application and small board computers use http calls (REST APIs) to communicate with the server.
    - HTTPS is used over HTTP (This is better than the requirement, but the requirements wording does make it incomplete)

Technical Requirements

- The application will connect to the beacons using Kontakt Beacon APIs. API key will be stored in the app.
- The application will display a map with the bus locations using Here maps API and caching the latest location. API key will be stored in the app.
- Application will ask for user’s permission to access to GPS, bluetooth, and internet using AndroidManifest.xml and java file to ask for user permission.
- System will track the live position of busses using GPS location from small embedded computer on the bus.
- Application will display the expected arrival times of bus using GPS location from small embedded computer on the bus and Here maps APIs. API keys will be stored in the app. Beacon UUIDs and geo-coordinates will be stored in the database.

- User will receive a push notification when the bus they are tracking is within the amount of distance a user specified.
- Application will show a list of nearby bus/car(s) that has/have beacon installed using Kontakt Beacon APIs and BLE. API keys will be stored in the app.
- App will track the current location of a user in the background using GPS.
- App will track travel distances and travel times of a user using Here maps APIs to calculate the distance and time. API keys will be stored in the app.
- Travel data will be sent to an online database when the device has an online connection using /POST, /PATCH, /GET and /DELETE APIs.

User Requirements

- Use application without an account.
- User receives push notification when bus is specified range from stop.
- Users can track bus with map.
- User can see expected bus arrival time.
- Able to track single bus.
- Able to use application on Android device.
- Users are Identified by Instance ID, or mac address. This ID does not change

User Groups

- General Bus Commuters - View map with bus locations, set a reminder for bus arrival, and track select buses.
- Server Admins - View travel data on an online database

Assumptions

- A beacon will be set up in one bus upon approval from BC transit. If not, the students will use their own transports. The engineering team is responsible for this.
- The engineering team will be responsible for installing and configuring beacons and will provide any help needed as soon as possible when COSC team test them. Both team will be present during the integration testing.
- The beacons given will work and respond as expected. The client and engineering team are responsible for any faulty beacons and hardware issues.
- The developers have knowledge of Git, Java and relational databases.
- Third party APIs (i.e. maps API, beacon API) will work properly.
- The team built APIs will be public and will not ask for any authentications.

- The cloud service (AWS/Firebase) will have at least 99.5% uptime.
- The beacon range will be 50-70 meters.
- The beacons will not be stolen or vandalized.
- The team will be agile, self-organized, and maintain synergy.
- Any expenditure including cloud service, third party APIs, transportation logistics will be provided by the client.
- The communication between stakeholders will be prompt, clear, and efficient.
- Each developer will be working a minimum of 8 hours during school weeks.
- There will not be any requirement changes from the client after October 16, 2018.
- The Beacon response time will be 1-2 seconds.

Environmental Constraints

- Requires approval from UBCO and BC transit to install beacons on campus and bus.
- The durability and performance of the beacons in Kelowna weather is unknown.
- Limited in-person meeting with the client.
- Field testing will be challenging during hostile weather.
- Development tool licenses and term of service.
- No control over third party APIs and cloud service.
- Some tasks are dependent on the task completion of engineering team and can not be done in parallel.
- Majority of the developers are not familiar with Android ecosystem, building REST APIs with node.js, and NoSQL databases.
- Lack of domain knowledge will lead to unrealistic estimations.
- Time constraint.

Risks

Risk	Likelihood	Severity	Mitigation Plan
Unfinished Project	Medium	High	Plan to stay ahead of deliverables.
Conflict with Engineering Team	Low	High	Frequent communication and transparency.
Losing members	Low	Medium	Involve all team members in development so we can pick up where other left off.
Unmotivated Team Members	Low	High	Constantly staying ahead of deliverable due dates
Faulty Beacons	Medium	High	Have as many as possible so we are not reliant on just a few.
Client company shut down	Low	High	Continue project with Scott.
No Control Over 3rd Party APIs	High	Medium	Study the APIs as much as we can so we call deal with any issues if they arise.
Environmental Constraints	Medium	High	Have backup plans, which will be discussed with the client.
Beacon Security	Medium	High	Change beacon provider or implement own security.
API Security	Low	High	Implement API key and authentication
Device Security	Low	Low	Delete device database as soon as possible.

Project Development Methodology

Scrumban methodology and Trello will be used for project management. The sprints will be one week long starting from every Tuesday. Weekly sync up, demo, planning meeting will be held on Monday. If Monday doesn't work out for some reason, the team will have meeting on Friday or Tuesday before the class (not recommended). Backlog will be created in each week's planning based on the progress and priority. Every week the developers will pick up the tasks from backlog they think they can finish in that week. If a task seems big, sub tasks can be created under that task. The tasks will have story points. 1 point represents 1 hour. Everyone is expected to finish at least 80% of their workload each week, although 100% is preferred. The rest can be carried forward to the next week's sprint. There is no need for daily scrum meeting as trello is synced to the scrum channel. So everything knows who is working on what.

The developers will create and push commits to the branches named with task numbers and create pull requests. Only integration lead or feature owners can review code and merge pull requests to the "Development" branch. The integration or devOps Lead will then update the master branch. The code will be reviewed after running automated test cases and a successful Jenkins build. Non-code documents will be merged by the technical lead.

Project Milestones

Milestones / Deliverables	Due Date	Complete (YES / NO)
Scope and Charter Document & Presentation	October 23th, 2018	YES
Design Requirements Document & Presentation	November 13th, 2019	YES
Integration Testing with the Engineering team	November 16th, 2019	YES
MVP Presentations	January 8th, 2019	YES
Testing Documentation and Presentation	March 2019	YES
Final Deliverable	April 17th, 2019	NO
Final Project Presentation	April 10th, 2019	YES

Use-Case Index

Use-Case ID	Use-Case Name	Primary Actor(s)	Complexity (1-5)	Priority (1-5)
1	Send Location Data to Device	Beacon	5	5
2	View Push Notification for Bus arrival	User	5	5
3	View Map of Current Bus Location	User	3	4
4	View Expected Arrival Time of Bus	User	5	5
5	View a List of Nearby Buses	User	3	3
6	Accept Permissions given to App	User	2	5
7	Start/Stop Tracking a Bus	User	4	5
8	View Travel Data on Online Database	Admin	4	5
9	Send Data to Single Onboard Computer	Server	5	5
10	Send Data to Application	Server	4	4
11	Receive Beacon Information	Server	5	5
12	Store Travel Information	Server	4	4
13	Provide User Location	Device	4	5
14	Provide Beacon Location	Device	4	4

Use Cases

ID	1
Name	Send Location Data to Device
Actor(s)	Beacon
Flow of Events	1. Collect Location Data 2. Send location to Device using Low-Energy Bluetooth
Pre-conditions	● Device is turned on ● Bluetooth is enabled on Device ● App is open ● Device is within range of the beacon
Descriptions	The beacon will send location data to the device and gateway so the user will be able to use the apps location ability.

ID	2
Name	View Push Notification for Bus Arrival
Actor(s)	User
Flow of Events	1. Server sends reminder notification to device 2. Device displays push notification of the reminder 3. User views push notification 4. User is brought to app
Pre-conditions	● User has set a reminder for a tracked bus ● App was connected to the internet when setting reminder ● App is connected to the internet when reminder is sent ● App has location services enabled
Description	A user will get a push notification sent to their phone even if the app isn't on screen. Interacting with the notification opens the app.

ID	3
Name	View Map of Current Bus Location
Actor(s)	User
Flow of Events	1. User location shown on map 2. User clicks on Bus icon 3. App requests bus locations for requested Bus from the server 4. App shows bus location
Pre-conditions	● App is open ● App is connected to the internet ● App has location services and enabled ● User is not currently viewing another bus’s location ● The server knows the location of at least one bus
Descriptions	The user will be able to view the current location of a selected bus on the map.

ID	4
Name	View Expected Arrival Time of Bus
Actor(s)	User
Flow of Events	1. User is shown own location and location of a selected bus 2. User selects a bus 3. App makes request to server 4. Server provides bus location 5. Application calculates time until bus’s arrival
Pre-conditions	● App is opened ● App is connected to the internet ● App has location services and Bluetooth enabled ● There is a bus in a valid location (It is possible to reach location by road)

Descriptions	The user will be able to view the expected arrival time of a bus shown on top of the map.

ID	5
Name	View a List of Nearby Buses
Actor(s)	User
Flow of Events	1. Open App 2. Application requests buses from the server 3. Server provides a list of known buses 4. User views nearby bus on bottom half of screen
Pre-conditions	● App is open ● App is connected to the internet ● The server knows the location of at least one bus
Descriptions	The user will be able to view a list of nearby buses.

ID	6
Name	Accept Permissions given to App
Actor(s)	User
Flow of Events	1. App checks user permissions 2. If permissions aren’t enabled App ask for User to allow access to permissions and User accepts permissions 3. Else Continues to App Home Screen
Pre-conditions	● App is open ● User has not accepted permission before OR User has deleted permissions
Descriptions	The user will have to enable location services and have Bluetooth turned on in order to use the application.

ID	8
Name	View Travel Data Online
Actor(s)	Admin
Flow of Events	1. Admin logs into the MongoDB account 2. Admin selects database storing travel information 3. Admin views information stored on database
Pre-conditions	● Admin is on a device connected to the internet
Descriptions	An admin will be able to view the collect travel data of users,

ID	7
Name	Start/Stop Tracking a Bus
Actor(s)	User
Flow of Events	1. Open App 2. View nearby bus on bottom half of screen 3. Select bus 4. Select “Track” 5. Select reminder time for notification 6. (Optional) User selects the “X” button to stop tracking the bus
Pre-conditions	● App is open ● App is connected to the internet ● The server knows the location of at least one bus ● User is not currently tracking the selected bus ○ Exception: User is doing step 6
Descriptions	The user will be able to track a bus live and see its location on the map. The user will be asked to set a reminder for the bus’s arrival

ID	9
Name	Send Data to Single Onboard Computer
Actor(s)	Server
Flow of Events	1. Collect Data 2. Send Data to Single Onboard Computer
Pre-conditions	Connection between server and on board computer is established
Descriptions	The onboard computer will be receiving data from the server.

ID	10
Name	Send Data to Application
Actor(s)	Server
Flow of Events	1. Receive request from application 2. Find requested data 3. Send data to requesting device
Pre-conditions	Application is connected to the internet User has performed an action that requests data from server
Descriptions	The application will be receiving data from the server.

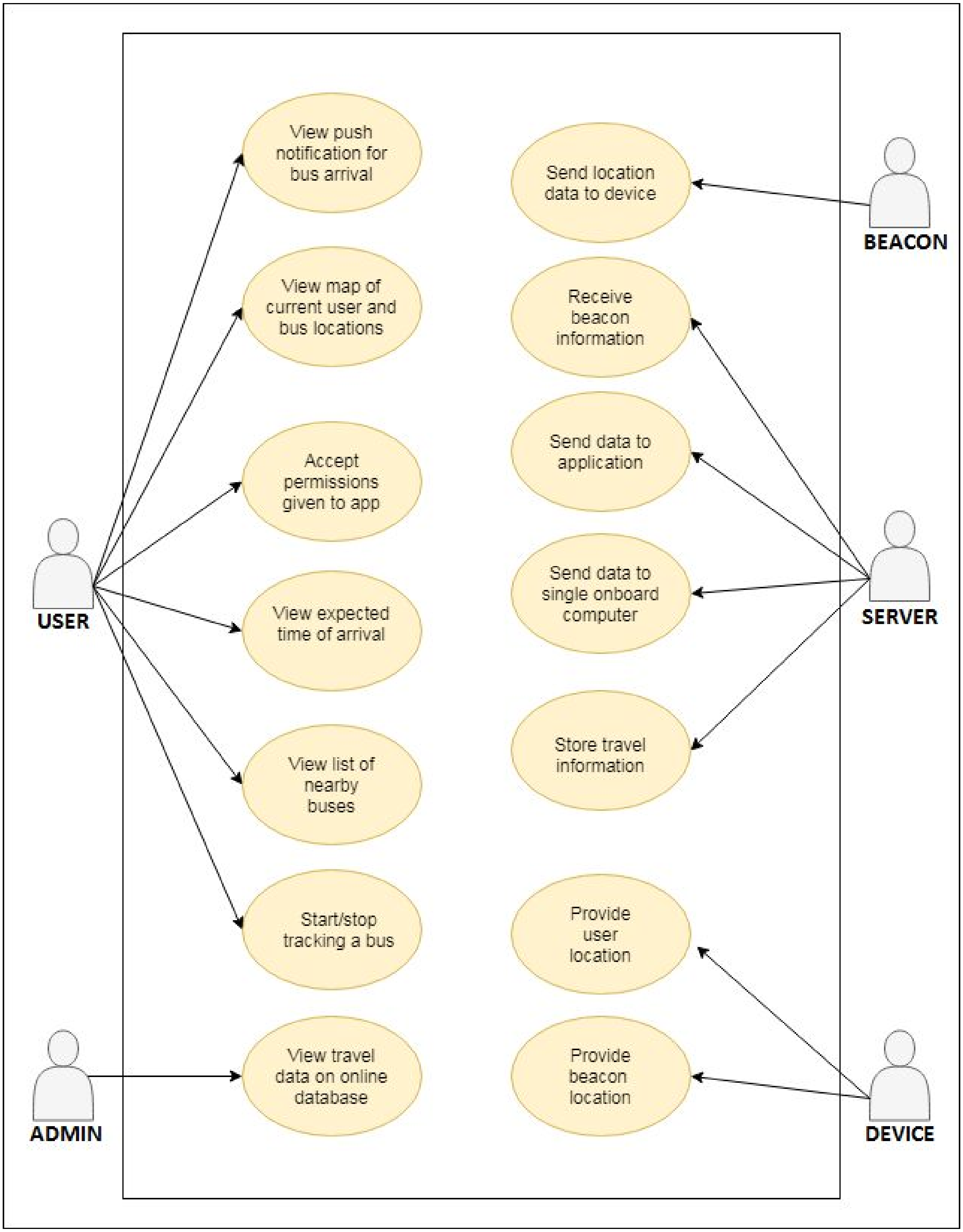
ID	11
Name	Receive Beacon Information
Actor(s)	Server
Flow of Events	1. Receive information from beacon
Pre-conditions	Connection between beacon and server is established
Descriptions	The server will be receiving information from the beacons.

ID	12
Name	Store Travel Information
Actor(s)	Server
Flow of Events	1. Server receives travel data from device a. When user enters a beacon’s range b. When user leaves a beacon’s range 2. Server stores data in database
Pre-conditions	● App is open ● Device is connected to the internet ● Device has Bluetooth enabled ● Device had entered range of the beacon of tracked bus ● Device had just left the range of the beacon ● Application tells server the user has left a bus
Descriptions	The server will anonymously store travel data of users

ID	13
Name	Provide User Location
Actor(s)	Device
Flow of Events	1. Application requests device location 2. Device determines its current location 3. Device informs application of current location
Pre-conditions	● App is open ● App is showing a map with the user’s location ● Device is connected to the internet ● Device has location services enabled ● Application has permissions for location
Descriptions	The device will provide the user’s location to the application

ID	14
Name	Provide Beacon Location
Actor(s)	Device
Flow of Events	1. Device receives location data from beacon 2. Application requests beacon location 3. Device provides beacon location
Pre-conditions	● App is open ● Device has location services and Bluetooth enabled ● Device is in range of a beacon ● Application has permissions for location and Bluetooth ○ Android requires location permissions to be given when using Bluetooth permissions
Descriptions	The device will provide the location of beacons to the application

Use-Case diagram


View push notification for bus arrival
View map of current user and bus locations
Accept permissions given to app
View expected time of arrival
View list of nearby buses
Start/stop tracking a bus
View travel data on online database
Send location data to device
Receive beacon information
Send data to application
Send data to single onboard computer
Store travel information
Provide user location
Provide beacon location
USER
ADMIN
BEACON
SERVER
DEVICE

Technical Specifications

Tools/Languages/Libraries	Type	Version
Android Studio	Development Environment	3.2.1
Oreo	Android OS Version	8
Jenkins/Travis	CI Solution	2.150
Git	Version Control System	N/A
Github	Version Control Hosting Service	N/A
MongoDB	NoSQL database	4.0
AWS	Cloud Platform	N/A
Java	Client side Programming Language	8
Node.js	Backend Programming Language	8.1.0
Postman	API debugging and testing tool	6.5.2
Here Maps Android SDK	Maps API	3.9
Firebase Cloud Messaging	Cloud Push Notification Service	7.00

Client Side:

- The client side (android app) will be built with Java and XML using Android studio.
- Here Maps Android SDK will be used for map and showing user and bus location.
- The build tool will be gradle.
- The android app gets UUId from firebase cloud messaging service, beacon id from AWS hosted api
- JUnit and Espresso will be used for the client side testing.

Server Side:

- The server side will be hosted on a cloud platform (AWS).
- Serverless framework will be used as deployment solution
- Node.js will be used primary backend language and writing REST APIs.
- The response of the REST APIs will be in JSON.
- The cloud database will be MongoDB.
- Mocha will be used for server side testing.
- IBM API Test Connect Test and Monitor will be used for server side automated testing

Work Breakdown

Work Breakdown Structure

Task List	Owner/ Completed	Ini	Kyle	Matt	Trevor	Wasek	Average Estimate (Per person)	Total Actual
Learning	All						41.6	143.5
Android	✔	8	13.5	20	3.5	10	14.8	55
Maps	✔		7.5			5	3.2	12.5
Kontakt.io	✔		4	12		5	3.8	21
CI Tools	✔				5.5	12	10	17.5
AWS/Firebase	✔			18	5	3	3.2	26
Databases	✔					3	2.6	3
Node.js	✔				6.5	2	4	8.5
Meetings		13	24.5	22	19	36	36	114.5
Client Communication			31.5				36	31.5
Design	All	Ini	Kye	Trevor	Matt	Wasek	13.6	107.25
Documentation	✔	10	16	20.75	12	16	9	74.75
Paper Prototype	✔	1		2	1	1	2	5
WireFraming	✔	4	0.5	1	1	2	1.8	8.5
MockUp	✔	15		2	1	1	1.8	19
Development		Ini	Kye	Trevor	Matt	Wasek	79.5	387.5

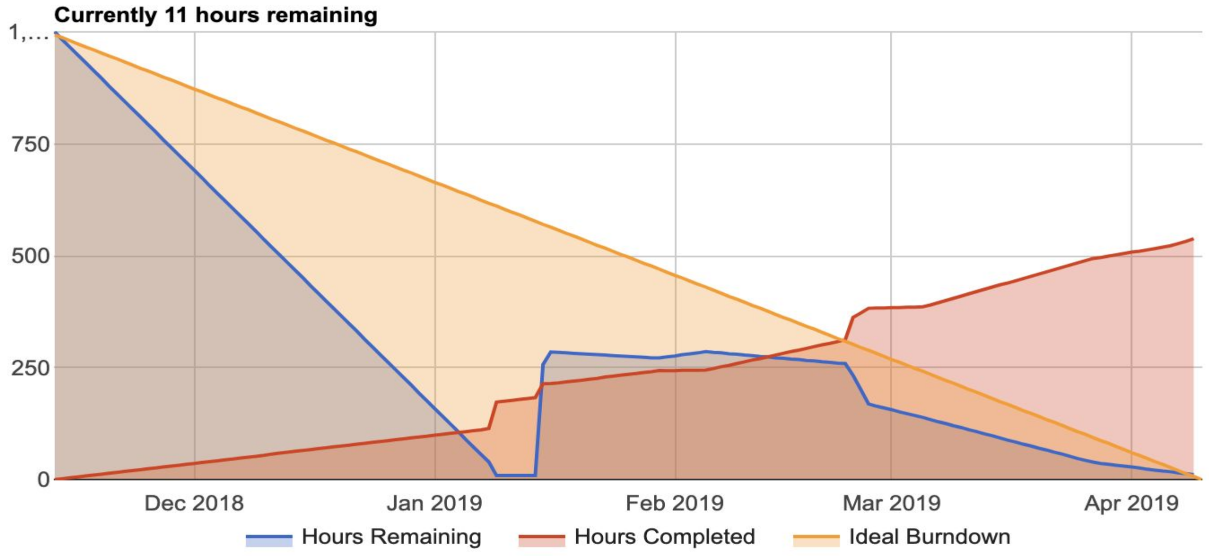
1. Maps API	Kyle, Matt						10.8	58.5
1.1 Integrate Maps Into GUI	✔		22			1	4.8	23
1.2 Show Where Bus is Located	✔		11.5			5	3.8	16.5
1.3 Create backend representation of data on AWS	✔			8			8	8
1.4 User location & Map	✔		11				0	11
2. Kontakt Beacon API	Matt	Ini	Kyle	Trevor	Matt	Wasek	3	4.5
2.1get location data from beacons, or arduinos.	✔			3.5		1	3	4.5
3. Backend - Firebase	Matt	Ini	Kyle	Trevor	Matt	Wasek	7.5	7.5
3.1 Send trip request to aws	✔			2			1	2
3.2 Send trip request to firebase	✔			1.5				1.5
3.3 Send bus location to app	✔			2	2		2.5	4
4. Android SDK	Tbd	Ini	Kyle	Trevor	Matt	Wasek	8.4	16
4.1 Collect GPS Data	✔		2			2	2.4	4
4.2 Store GPS Data	✔					8	4	8
4.3 Permissions	✔		1				1	1
4.4 Push Notification	✔			3			2	3
5. Backend - AWS	Wasek	Ini	Kyle	Trevor	Matt	Wasek	13.4	99
5.1 /POST /buslocation - sends bus location	✔					7	4	7
5.2 /PATCH	✔					5	5	5

/buslocation/:id - updates bus location once the first record is created.								
	Wasek	Ini	Kyle	Trevor	Matt	Wasek		
5.3 /POST /triprequest - sends a passenger trip request.	✔			0.5		4	4	4.5
5.4 /DELETE /triprequest/:id - cancels a passenger trip request.	✔			0.5		4	1	4.5
5.5 /GET /buslocation - returns latest bus location	✔					4	4	4
5.6 /GET /triprequest - returns info about bus stops if an user has planned a trip in last 100 mins.	✔					4	4	4
5.7 /GET /triprequest/:busstop - returns individual bus stop info, if a passenger has planned a trip in last 100 mins.	✔					4	2	4
5.8 /POST /triggers - sends user location and timestamp to server when they get on the bus	✔			8		5.5	4	13.5
5.9 /PATCH /triggers/id - sends user location and timestamp to server when they get off the bus.	✔			11		1.5	4	12.5
5.10 /GET /bustime/:busstop- returns latest bus ETA in minutes	✔					8	4	8

5.11 Configure serverless and deploy rest apis on AWS.Receive user and bus locations	✔					7	6	7
5.12 Deploy mongodb on AWS	✔					2	2	2
5.13 Setup MongoDB and connect with node	✔					4	4	4
5.14 Update existing APIs	✔					12	2	12
5.15 Setup test environment and Test /POST/Triggers API	✔					3	3	3
5.16 Set-up/Security	✔				4			4
6. User Interface	Ini	Ini	Kyle	Trevor	Matt	Wasek	6	37.5
6.1 Bus Number Button - when clicked it shows the bus to be tracked and changes the UI. Also triggers and the exit button to be displayed	✔	3		1			1.5	2
6.2 Track Button - triggers "Bus Tracker Activated" to be displayed	✔	2		1			1.5	1
6.3 ETA selection -	✔	2	7.5					7.5
6.4 Bus Selection Button	✔	3		1			1.5	2
6.5 Bus List uses recycler list	✔	3		1			1.5	2
6.6 Display expected arrival time of bus	✔		10		3			13

7. Integration and Testing	Trevor	Ini	Kyle	Trevor	Matt	Wasek	32.4	164.5
Ci Tools Setup					15	1	3	16
Usability Testing				1	4	5	4.8	10
Documentation	✔			4	10	16	6.6	30
Unit Testing		8	1	8	9	10	6.4	32
Integration Testing				4	4.5	8	6.6	16.5
Acceptance Testing	✔		2	1	1	5	5	9
Code Review			8.5	38.5			0	47
Final Documentation and Presentation	✔		16	10	19	16	16	61
Total Hours		72	190	208.25	126	249	186.7 (Per person)	738 (147.6 per person)

Burndown Chart



Team Hours

	Matthew De Leeuw	Kyle Rennie	Trevor Gallicano	Wasek Habib	Ini Oladosu
Week 1	6	6	6	14.5	4.5
Week 2	6.5	3.5	4.5	8	2.5
Week 3	7.75	7	6	9.5	6.5
Week 4	9	7	8.5	11	6.5
Week 5	7	6	7	9	8
Week 6	15.5	8	11	17	6
Week 7	11	8.5	9.5	12	7.75
Week 8	11	7	8	10	8
Week 9	5	7.5		8	
Winter Break	19.5	22	15	28	20
Week 1-2 (New Term)	11	16	6	12.5	11
Week 3-4	11	10	7	2	5
Week 5-6				24	18.5
Reading break	41.5	24	15.5	31.5	11
Week 8-9		13.5	6.5	16	6.75
Week 10-End	47.5	42	14.5	46	7.5
TOTAL	208.25	190	126	249	129.5

Current Implementation

Test Report

Type of Test Legend:

U-Positive Unit Test	I-Integration Test	A-API Test (Integration)
S-System Test	M-Manual/User Test	

Requirements	Type of Test	Pass (P) or Fail (F)	Developer	Tester
Functional Requirements				
User				
User is able to use application on Android device with Android 8 or later.	M	P	N.A.	MD,KR, TG
Users are Identified by Instance ID, or mac address. This ID does not change.	U,M	P	N.A.	KR
Android Application will ask for user's permission to access location, and to store information.	M	P	KR	KR
Application will be available to all users with no registration.	S,M	P	MD,KR	MD,KR
App will let the user track a bus/car.	M	P	KR	KR
App will track travel distances and travel times of a user.	U	F	MD	MD
Track, Bus Number & Exit buttons are pressable and maintain positions	U	P	IO	IO
UI				
Bus Number Button - when clicked it shows the bus to be tracked and changes the UI. Also triggers and the exit button to be displayed	U	P	IO	IO
GO button triggers ETA Time Picker	U	P	IO	IO

Requirements	Type of Test	Pass (P) or Fail (F)	Developer	Tester
ETA Time Picker allows user to select a time for push notification reminder	U,M	P	IO	IO
Time Picker is an option wheel	M	P	IO	IO
Track button allows user to start tracking a bus	U,M	P	IO	IO
Location Services				
Android Application will display a map with the user location.	M	P	KR	KR
App will only track the current location of a user while they are viewing the app.	U	P	KR	KR
Android Application will display a map with the tracked bus location.	M,I	P	KR	MD
Application will display the expected arrival times of the selected bus.	M	P	KR	MD
Application will show the 97 bus/car with LTE enabled small board computer with GPS tracker board.	U	F	MD	MD
Notifications				
User will be notified when the bus arrives at the selected stop	U	F	MD	MD
User will be notified about the wait time for the next arriving bus.	U	P	KR	KR
Server				
App will send user location asynchronously to the cloud server if they are tracking a bus/car.	A	P	MD	MD
Admins will be able to view data stored in the online database hosted in AWS.	M,U	P	MD	MD
Cloud server will perform some computations to check if the user is on the bus.	U	P	MD	MD
Cloud server will perform some computations to check if the user is out of the bus.	U	P	MD	MD

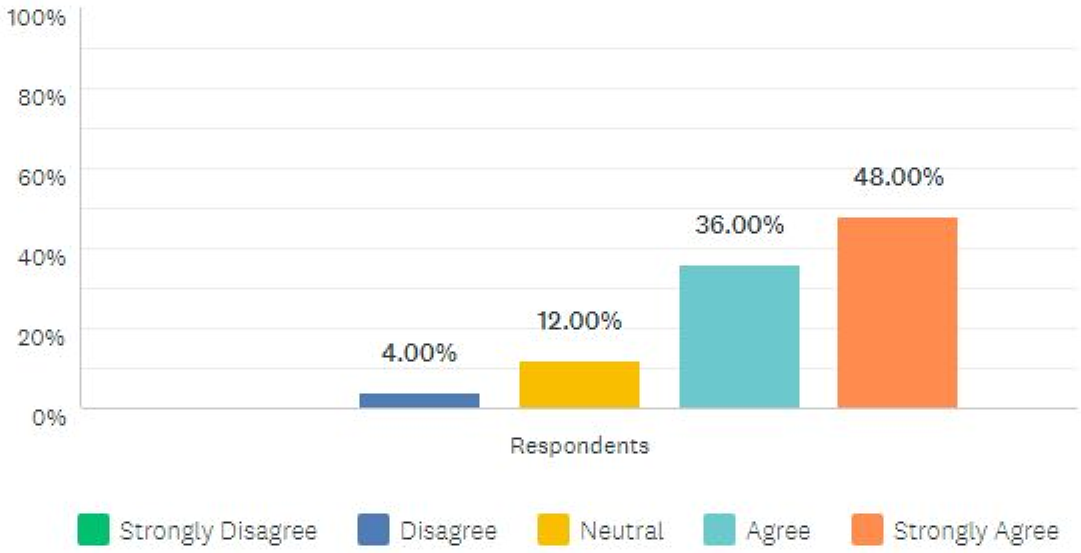
Requirements	Type of Test	Pass (P) or Fail (F)	Developer	Tester
PATCH - Cloud server will store user’s travel distance anonymously in database.	U	P	WH	WH,TG
PATCH - Cloud server will store user’s travel time anonymously in database.	U	P	WH	WH,TG
DELETE /triprequest/:id so trip request is removed when the user does not wish to track a bus anymore	U	P1	WH	WH
Rest APIs				
Application will send beacon information to the cloud server.	A	P	WH	TG
Application will send nearby beacon notification to the server when the user is on the bus.	A	P	WH	WH,TG
Application will send user selected bus stop and time reminder to the cloud server.	A	P	WH	TG
Application will send travel data to the server	A	P	WH	TG
Cloud server will send tracked bus location to the application.	A	P	WH	TG
Cloud server will get user location when he/she is in the beacon range of the bus location.	A	P	WH	TG
Cloud server will send tracked bus location to an LTE enabled computer at bus stop/bus.	A	P	WH	TG
Cloud server will send real time ETA to the android app	A	P	WH	TG
GET /triprequest/:busstop Cloud server will send requested bus stop location of the passenger to an LTE enabled computer on the bus.	A	P	WH	TG

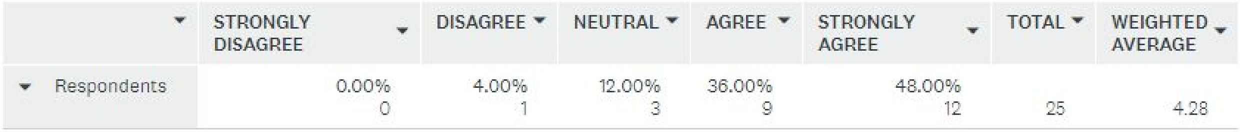
Requirements	Type of Test	Pass (P) or Fail (F)	Notes
Non-Functional Requirements			
Development			
Developed as an Android application written in java.	M,U	P	
Development process will be completed using the agile method.	M	P	
Application will use third-party API’s.	M	P	
Costs for development (such as cloud service or third party API’s) will be covered by the client.	M	P	Total cost of project remains $0.00
Performance			
User must be connected to the internet.	M	P	
Software will handle at least 2000 users.	S	F	Test out of scope. See Appendix C-7.3: Out of scope and exceptions
Software will be scalable up to the entire 97 bus route in Kelowna.	M	P	Requires adding locations of other bus stops
Bus position will be updated within 3 seconds (including screen latency).	M,U	P	Position update time determined by implementation of beacons
Application and small board computers use http calls (REST APIs) to communicate with the server.	M	F	See test below
Application and small board computers use https calls (REST APIs) to communicate with the server.*	M	P	Added security over original requirement.
Bus tracking will only show information related to the next bus(es).	M	P	
App will not track location in the background	M	P	

User Test Results

We have conducted one round of usability testing. A test version of the app was installed onto 2 Android devices. The application was tested by 25 users; however, it should be noted the proportion of testers who are part of the target audience cannot be determined. The testers were given a walkthrough of the app and were then asked to answer a set of six questions. Our goal was to receive a positive average score for each question (A positive score must be greater than 3: Neutral).

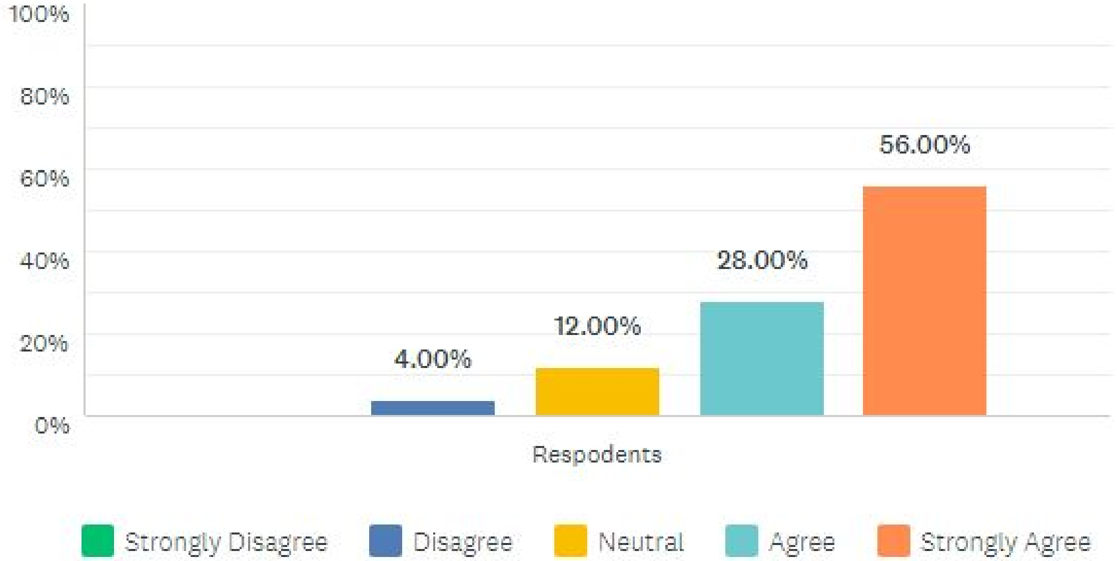
Q1: I am satisfied with the look and feel of the app.

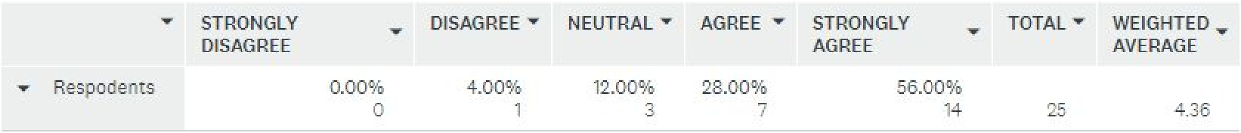




	STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE	TOTAL	WEIGHTED AVERAGE
Respondents	0.00% 0	4.00% 1	12.00% 3	36.00% 9	48.00% 12	25	4.28

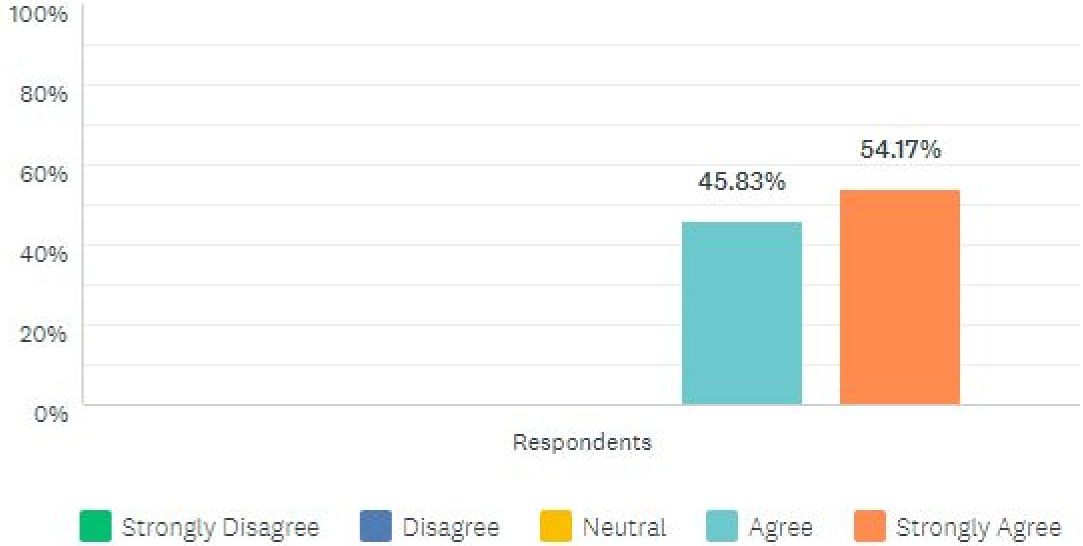
Q2: The bus location updates in an appropriate time.

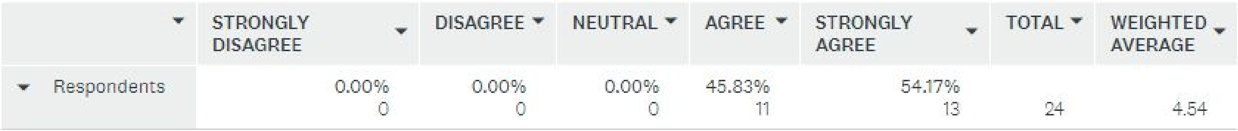




	STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE	TOTAL	WEIGHTED AVERAGE
Respodents	0.00% 0	4.00% 1	12.00% 3	28.00% 7	56.00% 14	25	4.36

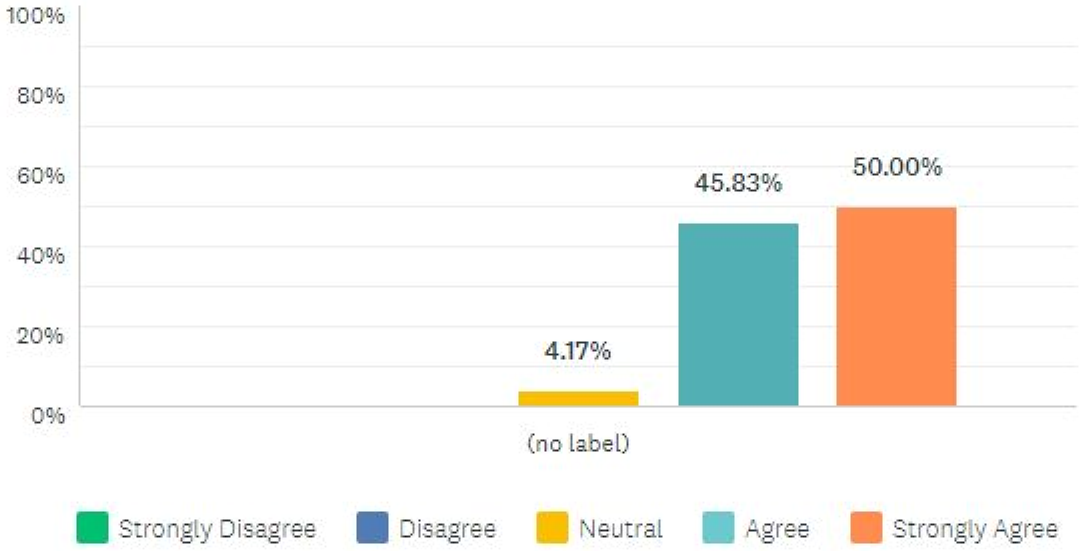
Q3: The interface is responsive to my actions.

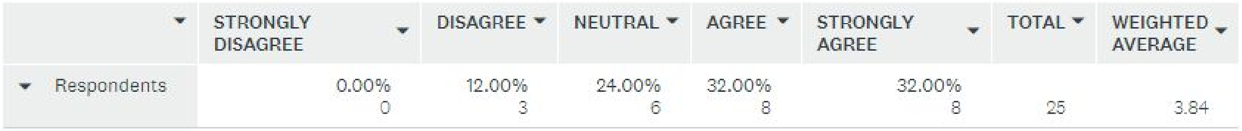




	STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE	TOTAL	WEIGHTED AVERAGE
Respondents	0.00% 0	0.00% 0	0.00% 0	45.83% 11	54.17% 13	24	4.54

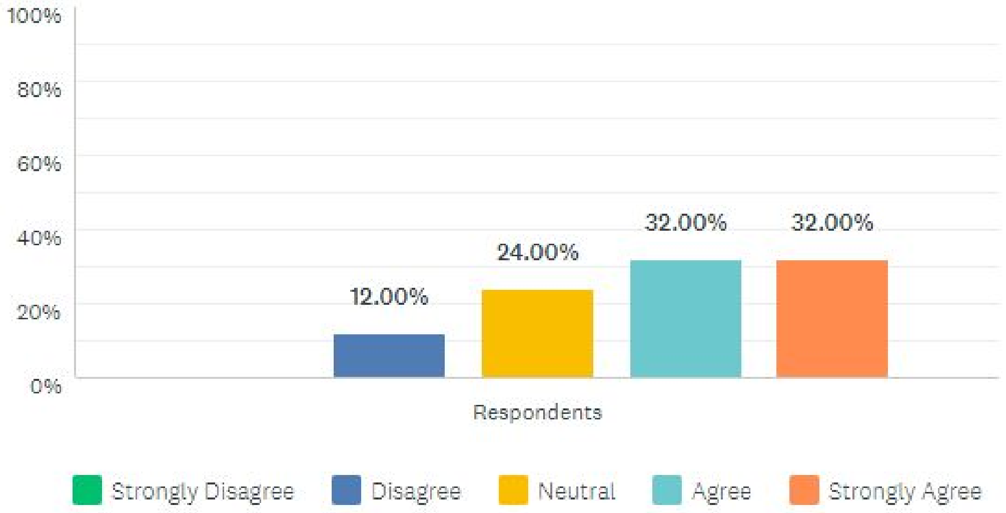
Q4: All the features are necessary and useful.

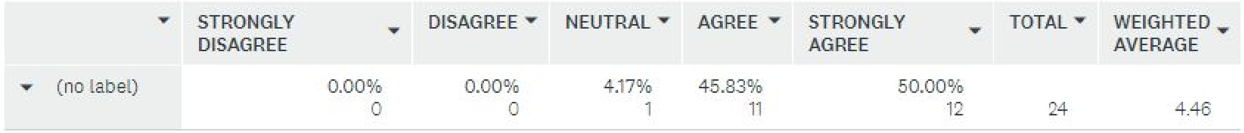




	STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE	TOTAL	WEIGHTED AVERAGE
Respondents	0.00% 0	12.00% 3	24.00% 6	32.00% 8	32.00% 8	25	3.84

Q5: I would recommend this app to a friend or colleague.





	STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE	TOTAL	WEIGHTED AVERAGE
(no label)	0.00% 0	0.00% 0	4.17% 1	45.83% 11	50.00% 12	24	4.46

Q6: Are there any additional features you would like to see added to the app?

- Help boxes or indication of what is happening
- Stylize the UI and design features a bit more
- Add the word minutes to the timer
- Have an option to select a location and see all the buses in the area
- The text for bus number looks [weird] on the side with the curved glass
- A little more UI for easier info
- Making buttons and status more clear. allow trip planning without centering on the bus.
- There isn't much affordance in the interface. Frequently wasn't sure where to go next.
- I would like to see a route when I plan a trip
- One of the testing devices has a screen that is curved on the left side. The application’s UI does not account for such a device.

User Testing Summary

The application received an average rating above 3 for all questions. This satisfies the minimum requirements we had set out to achieve. While most testers liked the look of the app, some respondents mentioned they found the UI to be unclear and confusing. In response to these concerns we implemented a new UI for the current version of the app. The current version has not undergone usability testing, and a second round of testing would need to be completed before the UI can be deemed deployment ready.

Design and Testing

Bus Tracking
University Of British Columbia Okanagan
COSC 499

Authors:

Document Owner(s)	Role	Contact
Wasek Habib	Product Manager	wasek.edu@gmail.com
Matthew De Leeuw	Integration Lead	mattdeleeuw@hotmail.com
Trevor Gallicano	DevOps Lead	trevorg@hotmail.ca
Ini Oladosu	Technical Lead	inioladosu@gmail.com
Kyle Rennie	Client Liaison	kyrenzie@gmail.com

Document History:

Version	Date	Document Revision Description
1.0	November 13, 2018	Initial document
2.0	November 20, 2018	Changes reflecting new project requirements.
3.0	April 16, 2019	Revision for final report

Usage Scenarios

User Groups

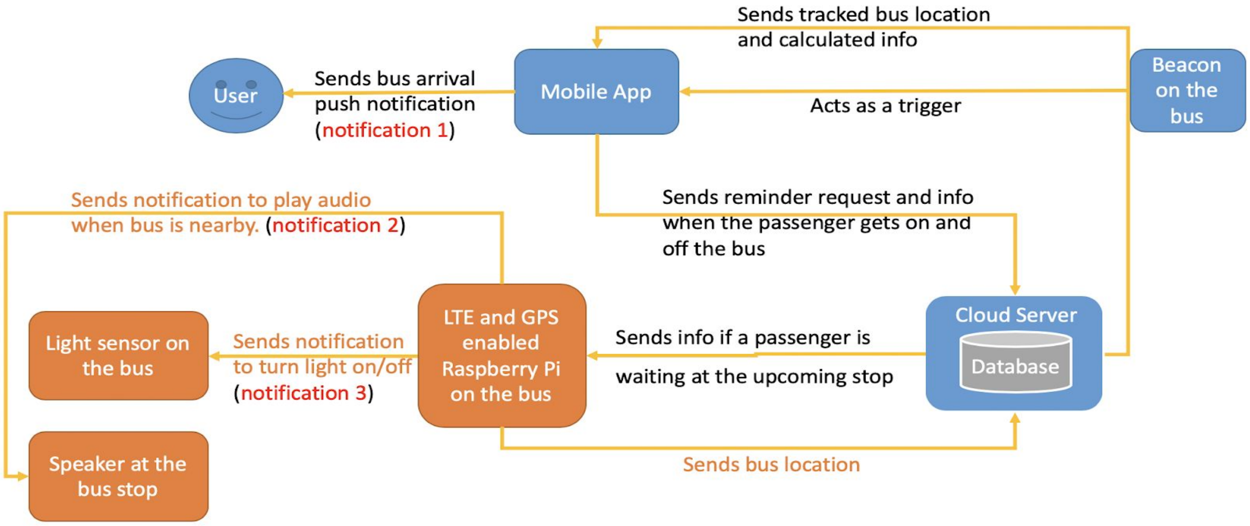
- University Students
- Retirees
- UBC Staff
- Commuters
- Physically impaired people

Actors

Actors	Goals
User	Select buses to track & receive notifications of the bus’ whereabouts
Beacon	Sends information to App
Admin	Views the travel data online
Server	Stores user travel data, beacon info, and sends and receives notifications

System Architecture Diagrams

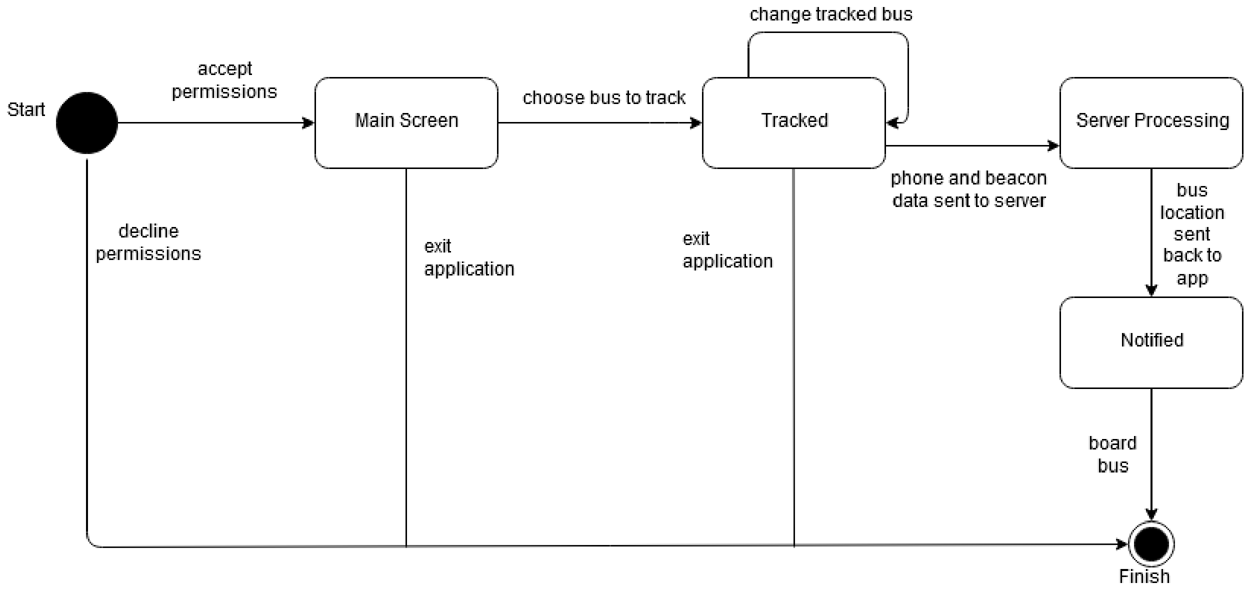
Software Architecture



The orange labeled Raspberry Pi and Light Sensor on the bus and Speaker at the bus stop are under engineering teams scope. We are responsible for the blue items, which are the mobile app, server, and beacon on the bus.

Raspberry pi sends real time bus location to the server and the app gets it when needed. The app sends user requested reminder to the server and server sends that request to the raspberry pi. The request contains passenger’s bus stop info. Raspberry pi processes it and passes the info to the light sensor and speaker when required. The server also does some calculations and send that info to the app. The app then sends a push notification to the user. Last but not least, the app receives a trigger from the beacons when the passengers are on and off the bus and sends that info to the server.

State Diagram



The state is very straightforward. There are two paths you can follow, one where you interact with the app and one where you are exiting the app. The top path, which is the intended path to use the app, follows the basic functionality of the app and what the user would experience as they used the app to track their desired bus. The user accepts permissions, choses which bus they would like to track, then the amount of time before bus arrival, then they are notified when the bus is close to their stop.

NoSQL Data Model

triprequests:

{"_id":{"$oid":"5cb3c9cb887a69ae074ffc2d"},"stopLocation":{"coordinates":[{"$numberDouble":"-122.5"},{"$numberDouble":"31.7"}],"type":"Point"},"userId":"fes08rweu08we09jdia","tripId":{"$numberInt":"3"},"busId":"fdsf","busStop":"UBCO A","requestedTime":{"$date":{"$numberLong":"1555286475610"}},"reminderTime":{"$numberInt":"12"},"__v":{"$numberInt":"0"}}

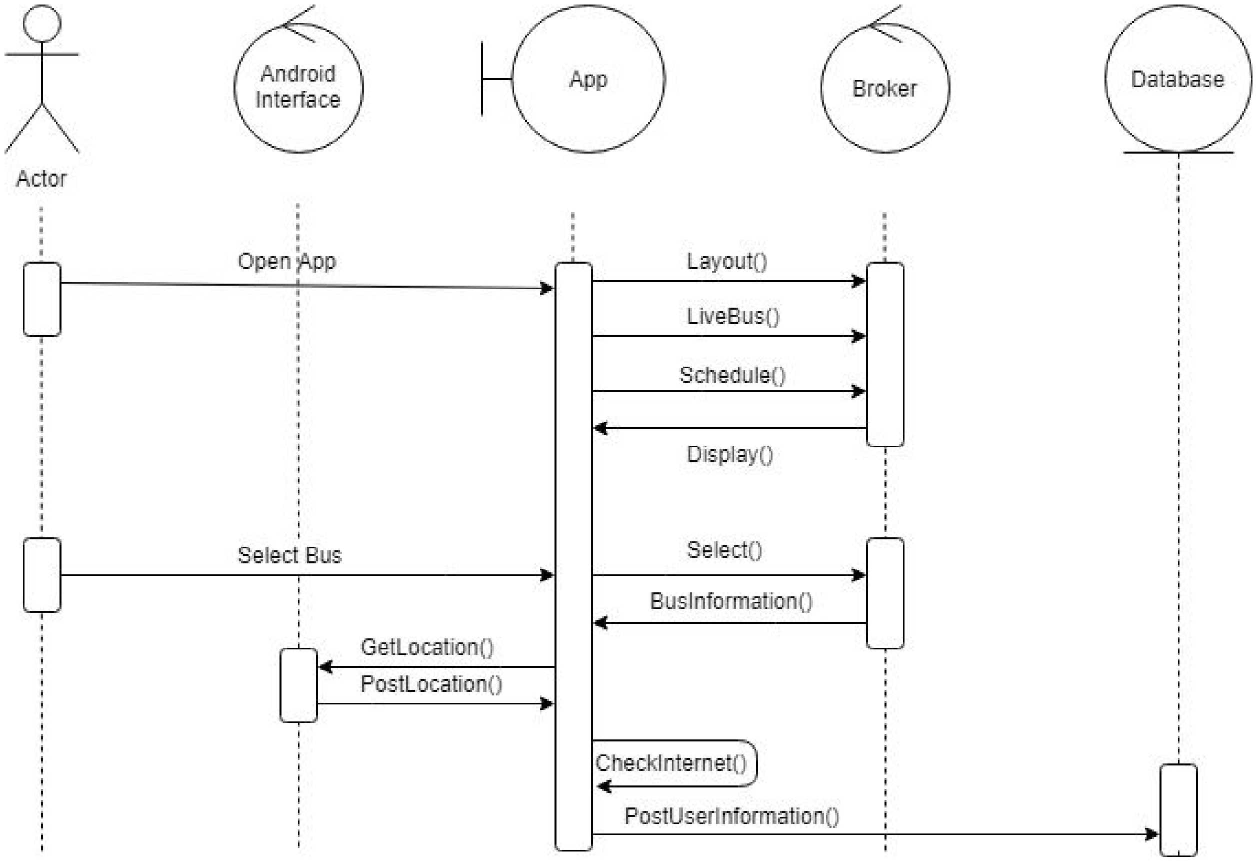
triggers:

{"_id":{"$oid":"5cb3c9cc887a6906254ffc2f"},"startLocation":{"coordinates":[{"$numberDouble":"-122.5"},{"$numberDouble":"31.7"}],"type":"Point"},"endLocation":{"type":"Point","c oordinates":[]},"completedAt":null,"userId":"fes08rweu08we09jdia","tripId":{"$numberInt":"3"},"beaconTrigger":"testdb6","busId":"97","startedAt":{"$date":{"$numberLong":"1555286476038"}},"__v":{"$numberInt":"0"}}

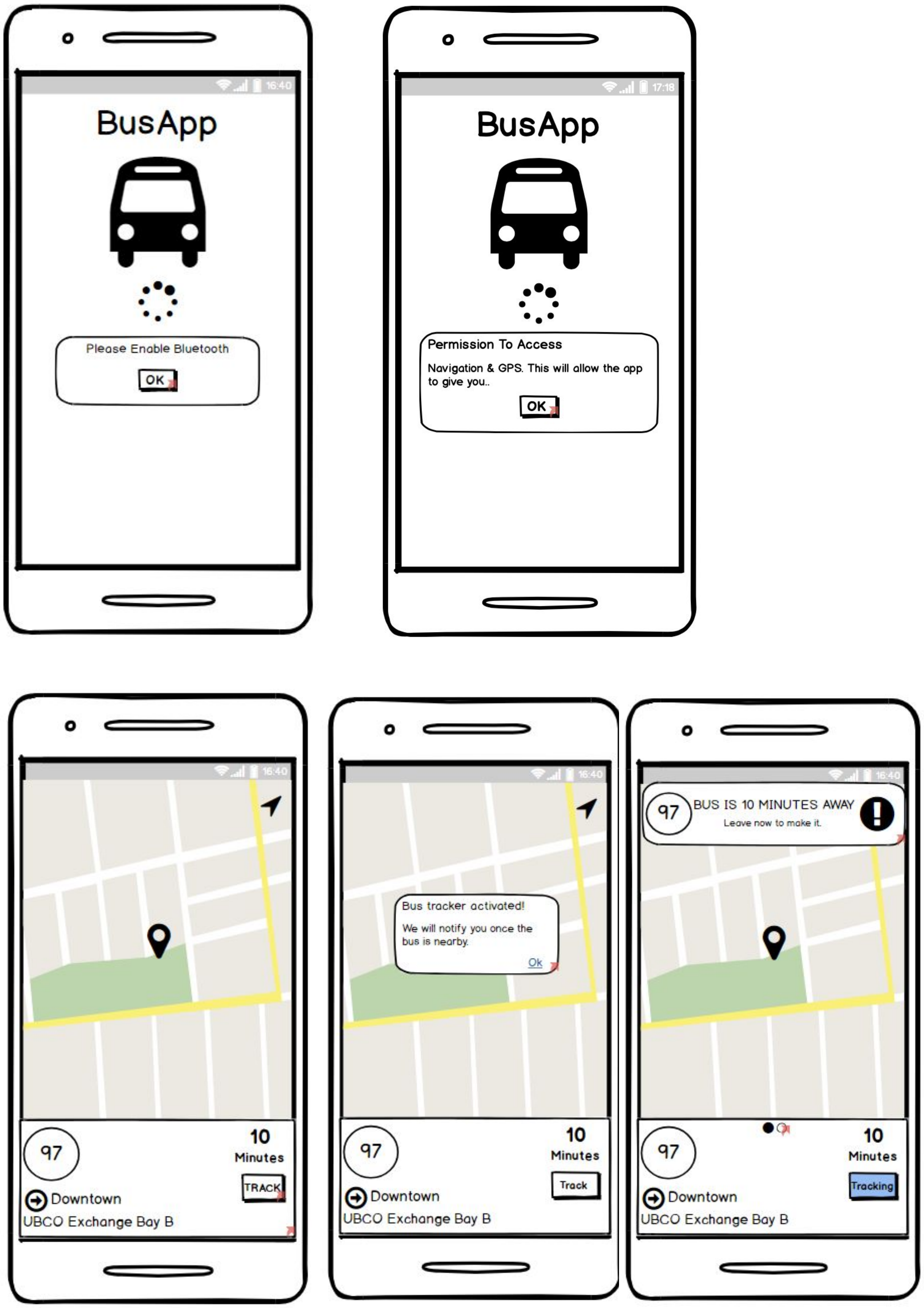
buslocations:

{"_id":{"$oid":"5c9d578bce7a1b3311c87ef7"},"location":{"coordinates":[{"$numberDouble":"-119.44254166666667"},{"$numberDouble":"49.94468166666667"}],"type":"Point"},"busId":"fdsf","timestamp":{"$date":{"$numberLong":"1553815435882"}},"__v":{"$numberInt":"0"}}

User Sequence Diagram



UI Mockup



Please Enable Bluetooth

Permission To Access

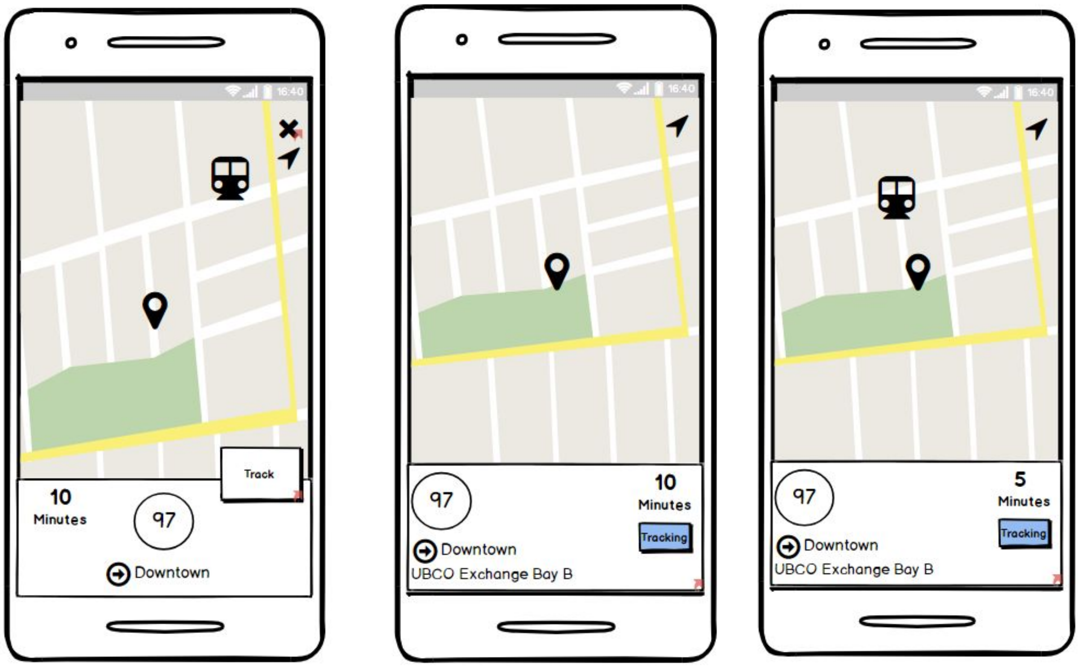
Navigation & GPS. This will allow the app to give you..

Bus tracker activated!

We will notify you once the bus is nearby.

BUS IS 10 MINUTES AWAY

Leave now to make it.



Detailed Test Plan

This test plan will be used to help the development team meet all requirements in an effective and efficient manner. The plan will outline the test-driven design strategy to be used by the team. Test scope, schedule, examples, and environment will be covered.

Test Scope

The following tables outline the scope of testing. Functional and non-functional requirements have been separated. Each component has been split into different types of tests. Positive system testing will involve all components, so it has been ignored.

Functional Scope

Item	Component	Description	Type of Test
FRT_01	App UI	Verify app displays information to the user. Includes bus location, all screens displayed to the user, and displaying the push notification.	Positive Unit Tests
FRT_02	App UI	Verify information being displayed is correct according to the other components.	Positive Integration Tests
FRT_11	Bus Tracking	Verify app receives information from nearby beacons.	Positive Unit Test
FRT_21	Cloud Server	Verify travel information is stored on the online server.	Positive Unit Test
FRT_22	Cloud Server	Verify server can send data to additional hardware created by the Engineering team.	Positive Unit Test
FRT_31	Data Transfer API	Verify travel information is correctly sent to the cloud database using an api.	Positive Integration
FRT_41	Push Notification	Verify push notification provides correct time estimate.	Positive Unit Test
FRT_42	Push Notification	Verify push notification is only given when the bus has arrived at the stop.	Positive Integration
FRT_51	Regression Testing	Verify app functions after the addition of a new module/feature.	Regression Test

FRT_61	Travel Information Database	Verify travel information is stored on the cloud server.	Positive Unit Test
FRT_62	Travel Information Database	Verify travel information is removed from the device after it has been sent to the server’s database.	Positive Integration Test
FRT_63	Travel Information Database	Verify that faked/broken travel information is excluded from the database.	Negative Unit Test

Non-Functional Scope

Item	Component	Description	Type of Test
NRT_01	Beacon/App Interaction	Verify app receives information from beacon when it is within the required range.	Positive Stress Test
NRT_02	Beacon/App Interaction	Verify app is able to receive information from a different beacon as it moves between beacon ranges.	Positive Integration Test
NRT_03	Beacon/App Interaction	Verify app is stable while out of range of beacons. App does not show false information and does not crash.	Negative System Test
NRT_11	Bus Location Tracking	Verify bus location is accurate within 100m.	Positive Integration Test

NRT_12	Bus Location Tracking	Verify bus location is updated at least once every 3 seconds.	Positive Unit Test
NRT_21	App UI	Verify buses updated location is displayed within the same 3 second interval as test NRT_12.	Positive Integration Test
NRT_22	App UI	Verify app displays information of the selected bus.	Positive Integration Test
NRT_22	App UI	Verify the app’s UI is user friendly.	User Testing

Out of Scope and Exclusions

Item	Description
Functionality of Additional Hardware	Functionality of any additional hardware outside of beacons and android device will not be involved in integration testing. Such devices are the responsibility of the Engineering team.
Number of Users	Stress testing of high number of users is excluded due to logistical limitations. Number of available devices for testing does not approach the required amount for proper testing.
Security of Beacon Data	Security for data transfer between beacons is out of scope. Responsibility of testing falls upon Kontakt.io.
Performance Req. as Regression Tests	Performance tests will be excluded from regression tests executed by the CI solution. These tests require connection to the physical beacons, which is not possible on an automated testing platform. (Note: Such tests will still be conducted manually)

Test Schedule

To accommodate a TDD strategy, three levels of testing will be implemented as follows:

- Unit tests will be used to verify the functionality of single modules. Unit tests will be written as black box tests. Unit tests will be prepared prior to their respective module's development and are executed upon completion of the module. Defects must be fixed prior to any integration of the module.
- Integration tests will be used to verify functionality between multiple modules. Integration tests will be prepared prior to the integration of two or more modules (and before the development of such modules if possible). Integration tests will be executed after the modules are integrated into the system. Once passed, integration tests will be added to the regression tests. Integration tests will be prepared and executed iteratively to ensure quick location of defects.
- System tests will be prepared prior to development of the application. System tests will begin being executed upon completion of the minimum viable product. Passed system tests will be added to regression tests.

Additional restrictions will be used to accommodate special requirements. These restrictions will be implemented as follows:

- Regression tests contain all previously passed integration and system tests. Key unit tests may also be included. These tests will be continually executed to ensure new features do not degrade code functionality. Regression tests will be executed by the CI solution prior to code deployment. Failed regression tests will stop the deployment of code.
- Stress tests will be used to verify minimum distance requirements are being met. Stress tests will be written and executed as system tests.
- User tests will be conducted manually by allowing users to interact with a working version of the system. Feedback will be gathered about their experience, and the feedback will be used to determine the result of the test. These tests will be created and conducted once a working version of the app is complete and may be redone after a major change to the App's UI.
- API tests would be used to verify data is being exchanged between cloud server and other devices using standard HTTP protocols. These tests will be created and updated once the REST APIs are written.

Negative Unit Test

Negative unit tests will be used to test the stored travel data. The user will not directly input any data; however, travel data will be validated after its transfer to cloud database to ensure there is no fake data. The tests will verify that data will be added to the database if and only if it is possible data. As an example, one test will verify that the module can detect that the given bus beacon id is not a valid beacon id.

Positive Unit Test

An example positive unit test would verify the UI displays the correct screen after the user selects a bus to track.

Positive Integration Test

An example integration test would have the app send travel information to the server but never receive a reply before the device goes offline. In a successful test the device should retain the information and attempt to send it again the next time the device connects to the server. The app should also not delete the record until it receives a positive response from the server. Success also requires the server to behave appropriately when the data is sent again (i.e. should not insert repeated data into the database).

Positive System Test

An example system test would have the user select a bus that is about to arrive at their stop, but have the user arrive to the stop after the bus has left. Success for this test would require the app to provide a push notification explaining that the bus was missed, and any other result would be a failure.

Negative System Test

The app should still function while it not receiving valid data from any beacons. One test would verify the app displaying a push notification while not receiving data from a beacon.

Regression Test

All regression tests are copies of other previously run tests. An example regression test would verify a UI element is still displayed correctly even after an update to the code.

User Test

An example UI test would have a test user use the basic functions of the app and have the user rate the usability of the app's UI. Success would require the user to have no negative opinions of the UI, while the contrary results in a failure.

Positive Stress Test

Stress tests would be used to verify the requirement for minimum distance to a beacon for the device to receive information. An example test would place the device at required range to the beacon, and verify the device is receiving all of the beacon's sent data over a long period of time.

API Test

API tests would be used to verify data is being sent and received from the cloud server to other devices. An example would be verifying bus location data is being sent to the user device and single board computers.

Test Data

Automated tests that require beacon information will use dummy data as a real connection would not exist to provide real data. This dummy data may include real beacon data taken from previous tests. Additional dummy data may be created to expand allow further testing. Additional data would include impossible/incorrect data and possible data which adds variety to the real collected data. Success of tests should take into account the correctness of the data being used. The data should be maintained in files so that regression tests can be run with consistent inputs.

Test Environment

To ensure efficiency of testing all developers are expected to be using Android Studio and should be writing with Java 8. In addition, tests should be written in such a way to be used by the following testing tools:

- JUnit – Framework used for constructing unit tests.
- Espresso – Framework used for testing the UI of Android applications.
- Jenkins- CI solution (Tests must be written in Java 8 only).
- Mocha - Test framework for node.js used for server side testing.
- Postman - Tool for verifying API responses.
- IBM Connect Test and Monitor - Tool for automated API tests.

Future of Project and Summary

Environment Description

Workflow

Developers on the project created Trello tasks based on what needed to be done before beginning work. Everyone would then assign the task to themselves and estimate the hours it would take after discussing what needed to be done in a meeting. Developers would create their own branch based on what they were working on. For large features developers would name their branch after the trello task they were assigned. They were encouraged to commit to the branch often so work would be secure and rolled back if required. Once a developer finished work on the feature they would create check to see if their were merge conflicts with the branch being merged into. If there were conflicts they would attempt to solve them by themselves. Afterward a pull request would be created and the integration lead would safely merge the branch into the master branch.

Testing

On the client-side unit tests would be written before work was done when possible. After a feature would be implemented it would be unit tested to see if it's working properly. Otherwise, it would be manually tested. For UI components they would be implemented first and then tested using Espresso testing. Jenkins was used to test that the build would compile every night on the master branch. On the server-side API responses were tested using postman. After all API's were determined to be working they were tested regularly using IBM Connect testing to ensure continued service.

Deployment

Any additional bus stop locations should be added to the application prior to deployment. This requires a name for each stop as well as the stop's longitude and latitude. An Android APK can be built from the master branch. The application can then be distributed to users/testers through any desired means (Google Play store, file sharing, ect). Following installation, users can immediately use the app, and the app will automatically connect to the necessary servers. Users will not need to download any additional applications or services in order to use the app. Deployment and use of the app does require the continuous operation of the AWS backend server, Firebase server, and MongoDB database (unless an alternative is developed). These servers run automatically and do not require extensive maintenance.

Known Bugs

- Each night the bus location will reset to the middle of the ocean as API testing is done
- The application can't deal with rotation of the phone, it will crash
- Movement of the map to center can be jerky due to lack of animation from HERE maps
- If map is scrolling and the user tries to center on a location it will continue scrolling after being centered (keeps its momentum)

Key Lessons Learned

- It's good to be flexible to change in a project
    - Specifically to changes in scope and scope creep
- Automation tools are very useful when used properly
    - Our automation tools weren't implemented early enough to be as helpful as they could have been
- Learning new technologies takes a large amount of the time
    - A large amount of time in this project was spent learning the technologies we chose to use

User Manual

System Overview

System Configuration

The bus tracking app runs on any Android device with Android 8 or later. The application requires connection to Internet in order to save data to database. Bluetooth must be enabled in order to obtain information from the beacons. Data saved in the database can be seen using any major Internet browser. After installation on the device, the bus tracking app can be used immediately without any further configuration.

User-Access Levels

All users are able to use the application and do not require any account. Travel data can only be viewed by admins.

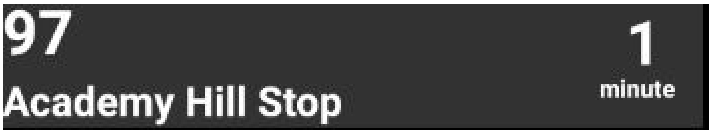
Contingencies

In case of loss of internet connection it is not possible to use the application. Data will be lost in the case of no internet connection, and notifications will not be received. Bus tracking and notifications will still work if only Bluetooth is disabled; however, travel data will not be properly collected if Bluetooth is disabled at any point prior to or during app use.

Getting Started

The Bottom Black Bar

(text view)



97

Academy Hill Stop

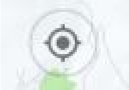
1

minute

This bar shows the bus being tracked in the top left, the stop which you are closest to, and how long till the bus arrives at that stop on the right. Clicking this bar without anything else will display the busses position on the map. After the bus is displayed on the map clicking this bar again will center the map view on the bus location.

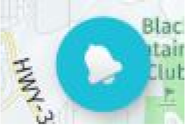
Navigation Button

(top right)



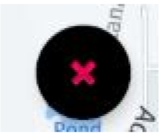
Pressing this button at any time will center the map view on the users position.

Notification Button



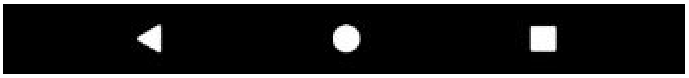
If the bus isn’t being tracked the button on the bottom right will look like this. Clicking this will show the bus on the map if it isn’t already showing and bring up a dialogue box asking you how long before your bus arrives at your stop do you want to be notified.

Cancel Tracking Button



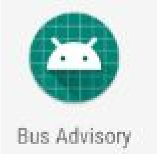
The button on the bottom right looks like this if you have requested to be notified of a bus arrival time. If you want to cancel the notification you can press this button and the bus will stop being displayed.

Android Button Bar



To leave the app you may use any of these three, but they all have seperate uses. After you have left the app you may close it by bringing up active apps with the box on the right and then dragging the bus app off the right side of the screen.

App Icon



To open the app locate this icon in your app folders and press it.

Using the System

A user would begin by opening the app with the Application Icon shortcut. Once the app opens for the first time it will ask for permissions. When the permissions are accepted a map will display their current location. A user can use this map to see where they are and to locate nearby bus stops indicated by small empty circles. The app assumes the closest stop to them is the one they would want to use for a bus stop so it will show the estimated time of the bus to there. When the user would like to be notified of the bus arrival they can click either the text view bar or the notification button, which will display the bus. If they click the notification button a window will pop up asking them how long before the bus arrives would they like to be notified. They may choose a time and click ok. Now the bus is being tracked and when the bus is in range they will receive a push notification to their phone telling them the bus is arriving in the specified time. They can click the notification or bring the app up and catch their bus.

Reporting

Report Capabilities

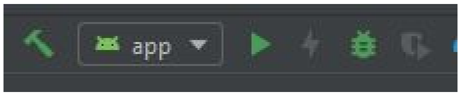
The application sends all necessary information to the server. The server stores the current location of each bus (including all test data); current, canceled trip requests; and all travel data from users. Each entry of travel data contains information for time, location, bus id, and whether the user was entering or exiting the bus. A anonymous ID is used to distinguish data between users.

Report Procedure

The data described above can be accessed by admins through MongoDB (See Resources for url and login information). An admin must log-in to view the data, and can view/filter all collected data. Data can be found by selecting ClusterDB followed by Collections.

Application Installation Guide

1. You will need Git installed (we recommend using GitHub Desktop for ease of use) and have a GitHub account. https://github.com/
2. Install Android studio, specifically version 3.2.1.
3. Clone the repository to your local machine using Git. (https://github.com/UBCO-COSC499-Winter-2018-Term-1-2/project-12-bus-advisory-offline-real-time-bus-location-tracker)
4. Open Android studio and select “Open an existing Android Studio project”, then locate where your repository was created and select development in the repository. Click OK.
5. This is where all future client-side work can be done. All code can be found under MapActivity.java which is located in app>src>java
6. To start the emulator and see the application in use click the run button



7. Click “Create New Virtual Device”, you may choose any device you wish, but for the sake of ease we recommend the Nexus 5 or Pixel 2. Click Next after having the device selected
8. For a system image click download on Oreo with API level 26 and on Android 8.0. When the download is complete click next.
9. You may now choose to name your emulator under AVD Name and make sure the orientation is in portrait.
10. Now when you press run it will display the emulator you created, so select it and hit OK
11. The emulator will work as a real phone would, however it will not get correct GPS location from an emulator.
12. To change location in the emulator click the … on the right side white bar. Under Location in the box that comes up you can change the values of latitude and longitude.
13. The ETA and closest bus stop only work when the bus is in a location on land so it will likely have to be moved as the testing resets its position every night.
14. To install the application on a real device you will have to turn on USB Debugging under developer options on the device. Connect the device to your computer with Debugging enabled and when you click the run button the device should be shown in the list to install to.

Server-side Installation and Deployment Guide

The server side is built with node.js and uses npm and serverless framework.

1. Install serverless globally- $ npm install serverless -g
2. Configure AWS credentials
3. Go to development/rest-apis using terminal
4. Install all packages- $ npm install
5. Run the server locally- sls offline -c
6. Deploy - sls deploy

Unfinished Testing

Although all core requirements have been tested there is still some testing that has not been completed:

- Delete API has been tested using Postman but has not been added to the automated tests, so it should be tested again prior to deployment
    - API tests should be updated to stop moving the test bus to the ocean
- Jenkins builds the master branch daily. This should be updated to build and test after a pull request is made to master.
- A second run of user testing should be done to verify the usability of the updated UI.

Core Requirements Delivered

- Application is available to all users without registration.
- Track a bus/car.
- Display a map with user location.
- Create a server to handle processing and data
- Share information between Raspberry Pi and Android app.

Unimplemented Features

- User will receive a push notification when the selected bus has arrived at the stop.
  - Push notification is not sent automatically, but push notifications manually sent work as intended
- App will track travel distances and travel times of a user.
  - App does not store travel data. Travel data is instead sent straight to the server
- Software will handle at least 2000 users.
  - Requirement cannot be tested

Resources

GitHub Repository

Location:
https://github.com/UBCO-COSC499-Winter-2018-Term-1-2/project-12-bus-advisory-offline-real-time-bus-location-tracker
Description: Private GitHub repository containing all up-to date code and documentation.

API Documentation

Location:
https://docs.google.com/document/d/12QO34SncpFbj08Qc4KUrUzaNMPt20GCNRn9pyykwJ5E/edit?usp=sharing
Description: Documentation for design of API’s as well as example requests and responses.

MongoDB

Location
https://cloud.mongodb.com/user?_ga=2.89485019.1106248769.1553027188-1298984703.1553027188#/atlas/login
ID ubco.bus.advisory@fremtidmedia.com
Password UBCObusadvisory1!

Description Non-relational database storing bus locations, trip requests, and travel data. After logging in select ClusterDB in the ‘Sandbox’ window then select Collections in the top ribbon to view the stored data. Travel data is labeled as ‘triggers’.

Firebase Server

Location: https://firebase.google.com/
ID: ubco.bus.advisory@fremtidmedia.com
Password: UBCObusadvisory1
Description: Server used for sending out push notifications/reminders

AWS Backend Server

Location: https://us-east-1.console.aws.amazon.com/console/home?region=us-east-1
ID: ubco.bus.advisory@fremtidmedia.com
Password: UBCObusadvisory1
Description: Server where the core APIs are hosted.

AWS Jenkins Server

Location:
https://us-east-2.console.aws.amazon.com/console/home?region=us-east-2
ID ubco.bus.advisory@fremtidmedia.com
Password UBCObusadvisory1
Description: AWS EC2 instance hosting jenkins and the fake bus node application (for testing). The instance can be accessed by using ssh to reach:

e2-user@ec2-18-218-198-68.us-east-2.compute.amazonaws.com

With the ppk file (if used on a windows machine). The .ppk file must be converted from the .pem key available in the repository: docs/AWS.
This server can be terminated with no ill-effect on the rest of the project.

IBM Connect Test and Monitor

Location: https://ibm-apiconnect.github.io/test-and-monitor/
ID ubco.bus.advisory@fremtidmedia.com
Password: UBCObusadvisory1
Description: Automated API testing tool currently set to test the APIs once per day.

Approval

Product Manager:____________________________________

Signature: ____________________________________

Project Sponsor:____________________________________

Signature: ____________________________________