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

1. Project Description

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. The project has two parts: Hardware and Software. COSC team is responsible for anything related to the server, location engine, development and deployment of the android application and its connection to the beacons (software side). The offline first 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 when he/she is in a certain range of the beacon at the bus stops. The beacons and gateway will be installed and configured on a bus/car and at bus stops by the engineering team. The engineering team is also responsible for programming speakers and light sensors (small single board computers) located in the bus stops and busses, respectively. The application and single board computers will receive the bus/car information and passenger bus stop location from the cloud server. 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 anonymously on device and uploaded to the cloud upon internet connection.

2. Terminology

Item	Description
Beacon	A low-energy Bluetooth device used to determine a device's location within a mesh network. 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.

3. 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
Gateway	Sends beacon data to server
Location Engine	Receives beacon data from gateway and send beacon data using kontakt api

Use-Case Index:

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

	Beacon Information			
12	Provide User Location	Device	4	5

Use Cases:

ID	1
Name	Send Beacon Data to Server
Actor(s)	Gateway
Flow of Events	1. Collects Data 2. Sends to Server
Pre-conditions	There is a connection between the server and the gateway
Descriptions	The gateway will send data from the beacon to the server.

ID	2
Name	Send Location Data to Gateway & Device
Actor(s)	Beacon
Flow of Events	1. Collect Data 2. Send to Gateway & Device
Pre-conditions	Connection between gateway and app to beacon is established
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	3
Name	View Push Notification for Bus Arrival
Actor(s)	User
Flow of Events	1. App scans for Beacon information 2. The App runs in the Android device background and detects time 3. The App uses the information from Beacon to send the notification at the Bus's time of Arrival
Pre-conditions	The user has the app open, is connected to the internet and has enabled location services, or bluetooth enabled
Description	A user will get a push notification sent to their phone even if the app isn't on screen.

ID	4
Name	View Map of Current Bus Location
Actor(s)	User
Flow of Events	1. User location from android device shows nearby busses 2. User clicks on Bus icon 3. App requests Beacon information for requested Bus 4. App translates Beacon information and shows bus location
Pre-conditions	The user has the App open, is connected to the internet and has enabled location services, or bluetooth enabled
Descriptions	The user will be ables to view the current location on the map.

ID	5
Name	View Expected Arrival Time of Bus
Actor(s)	User
Flow of Events	1. User location is used to show nearby bus icons, and requests Beacon and database information for them 2. User selects a bus 3. App translates Beacon information and shows selected bus' real time arrival time
Pre-conditions	The user has the App opened, is connected to the internet and has enabled location services, or bluetooth enabled
Descriptions	The user will be able to view the expected arrival time of a bus shown on top of the map.

ID	6
Name	View a List of Nearby Buses
Actor(s)	User
Flow of Events	1. Open App 2. View nearby bus on bottom half of screen
Pre-conditions	User has the app opened and is connected to a beacon via bluetooth
Descriptions	The user will be able to view a list of nearby buses.

ID	7
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	User has the app opened and is connected to a beacon via bluetooth
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. User location is used to show nearby bus icons, and requests Beacon and database information for them 2. User scrolls to selected bus 3. App translates Beacon information and shows selected bus' real time arrival time and scheduled arrival times
Pre-conditions	The user has the App opened, is connected to the internet and has enabled location services, or bluetooth enabled
Descriptions	An admin will be able to view the collect travel data of users,

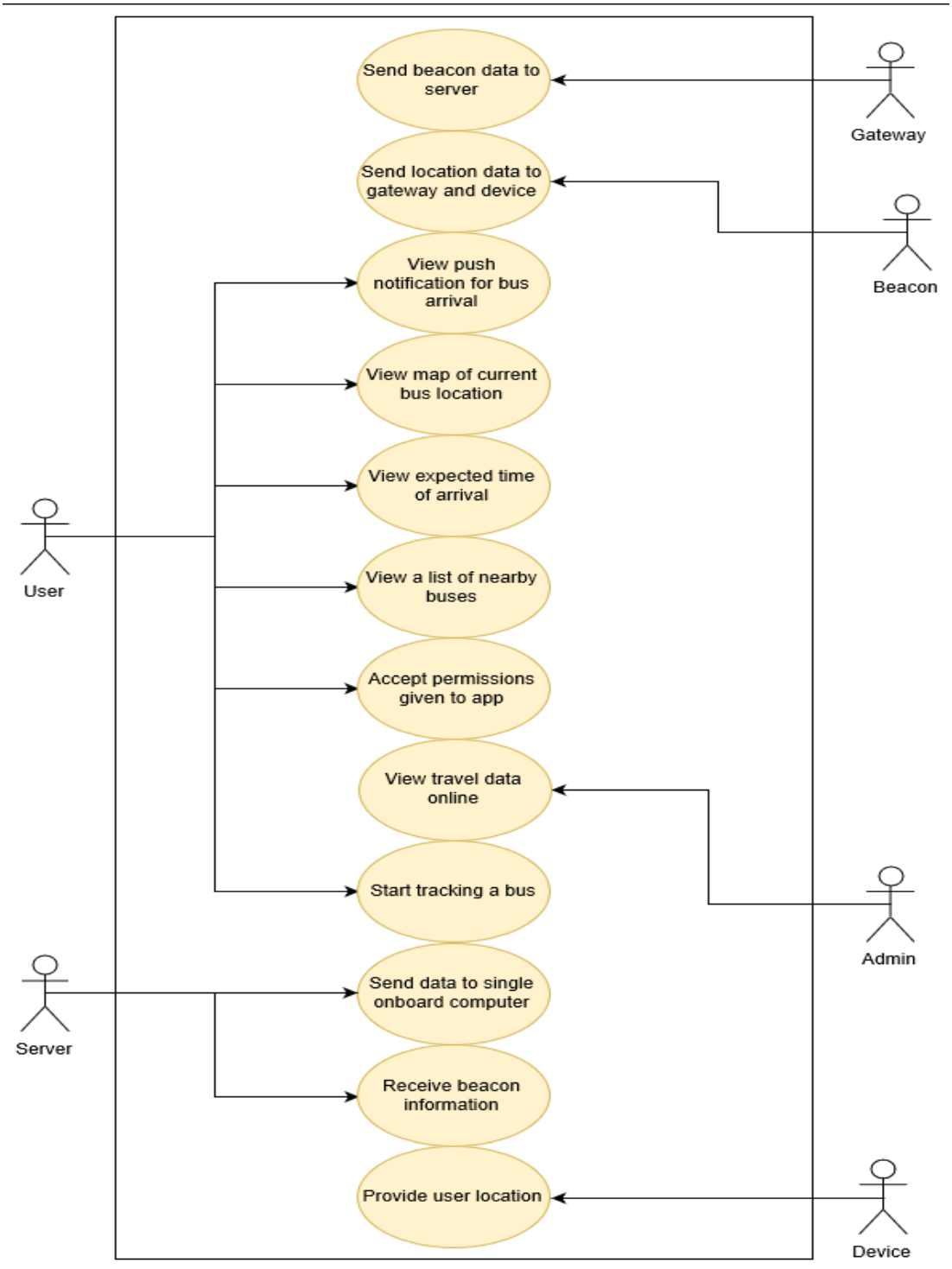
ID	9
Name	Start Tracking a Bus
Actor(s)	User
Flow of Events	1. Open App 2. View nearby bus on bottom half of screen 3. Select bus 4. Select “Track”
Pre-conditions	The user has the App opened, is connected to the internet and has enabled location services, or bluetooth enabled
Descriptions	The user will be able to track a bus live and see its location on the map.

ID	10
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	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.

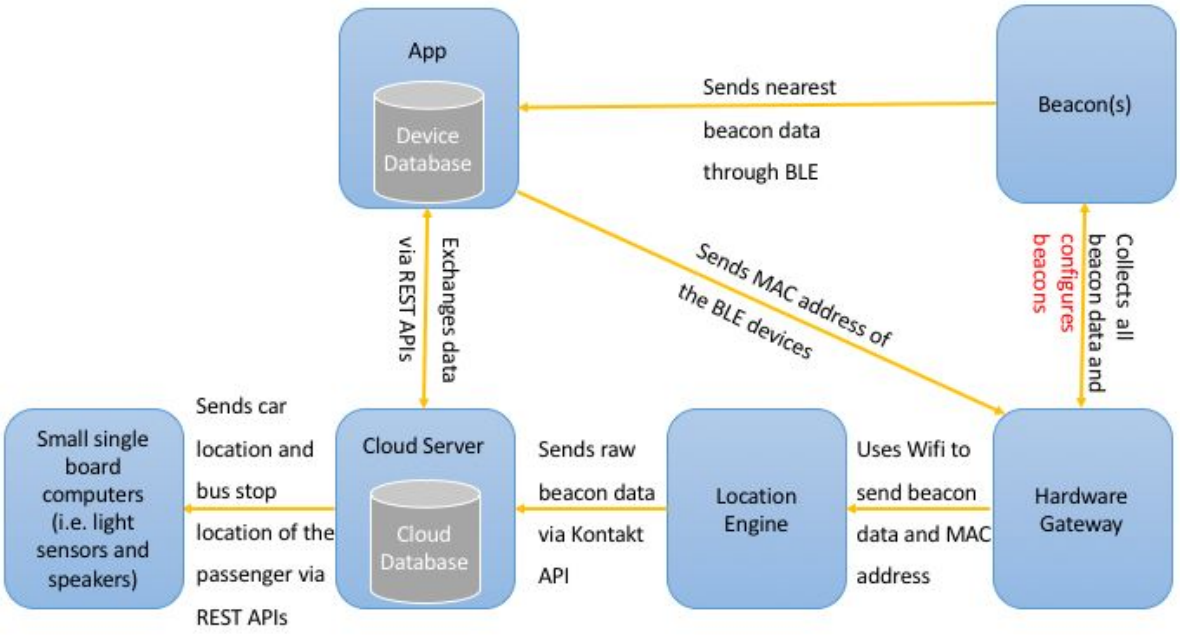
ID	12
Name	Provide User Location
Actor(s)	Device
Flow of Events	1. Provides user location via location services
Pre-conditions	Location services have been turned on
Descriptions	The devices will be providing the user location in order for them to use the app.

Use-Case diagram:

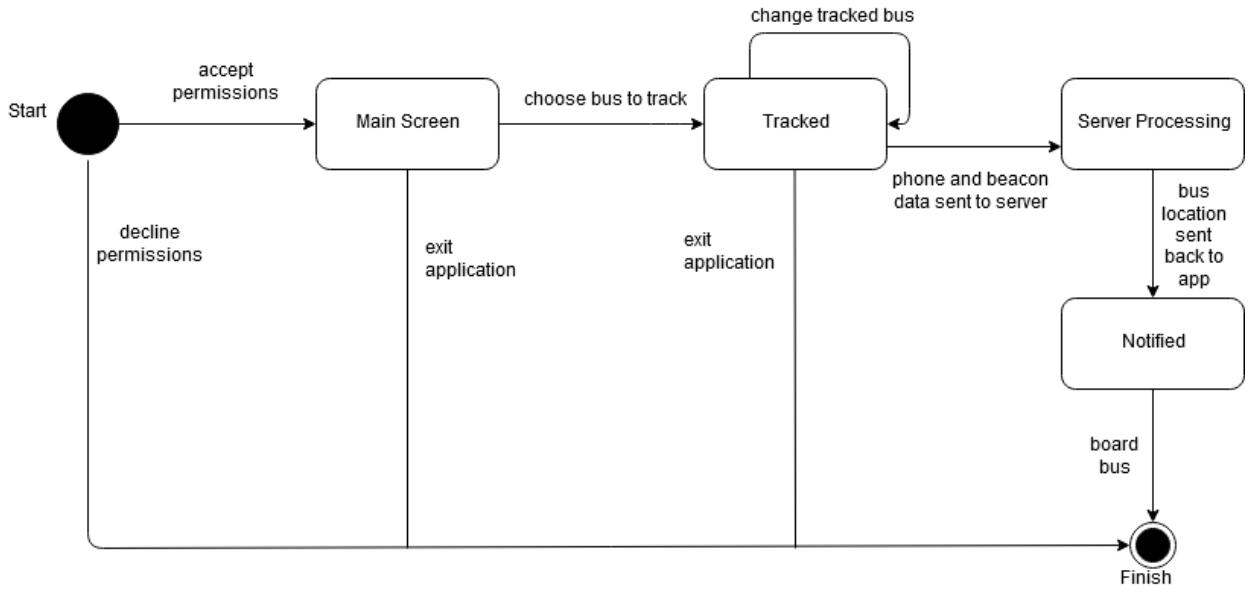


4. System Architecture Diagrams

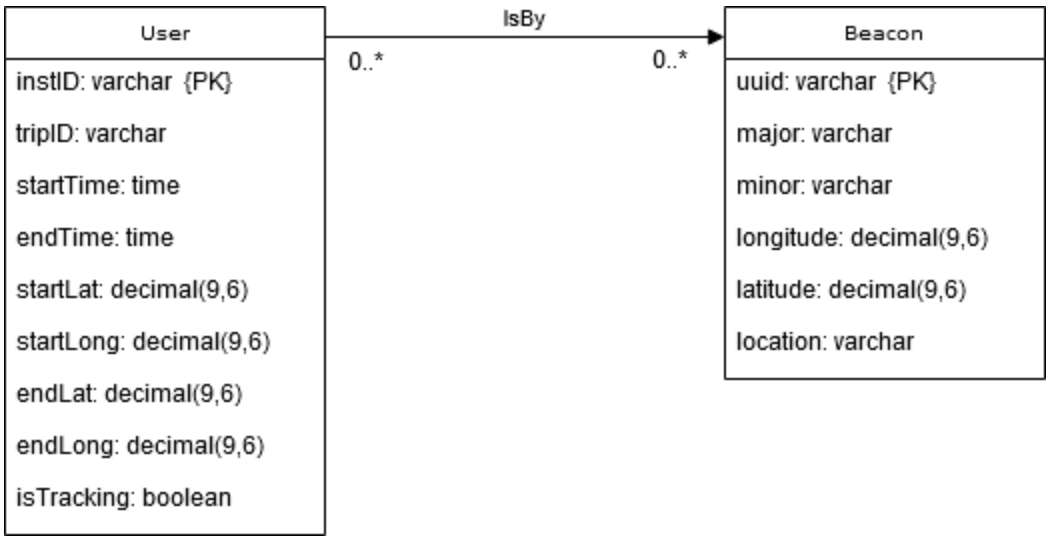
Software Architecture:



State Diagram:



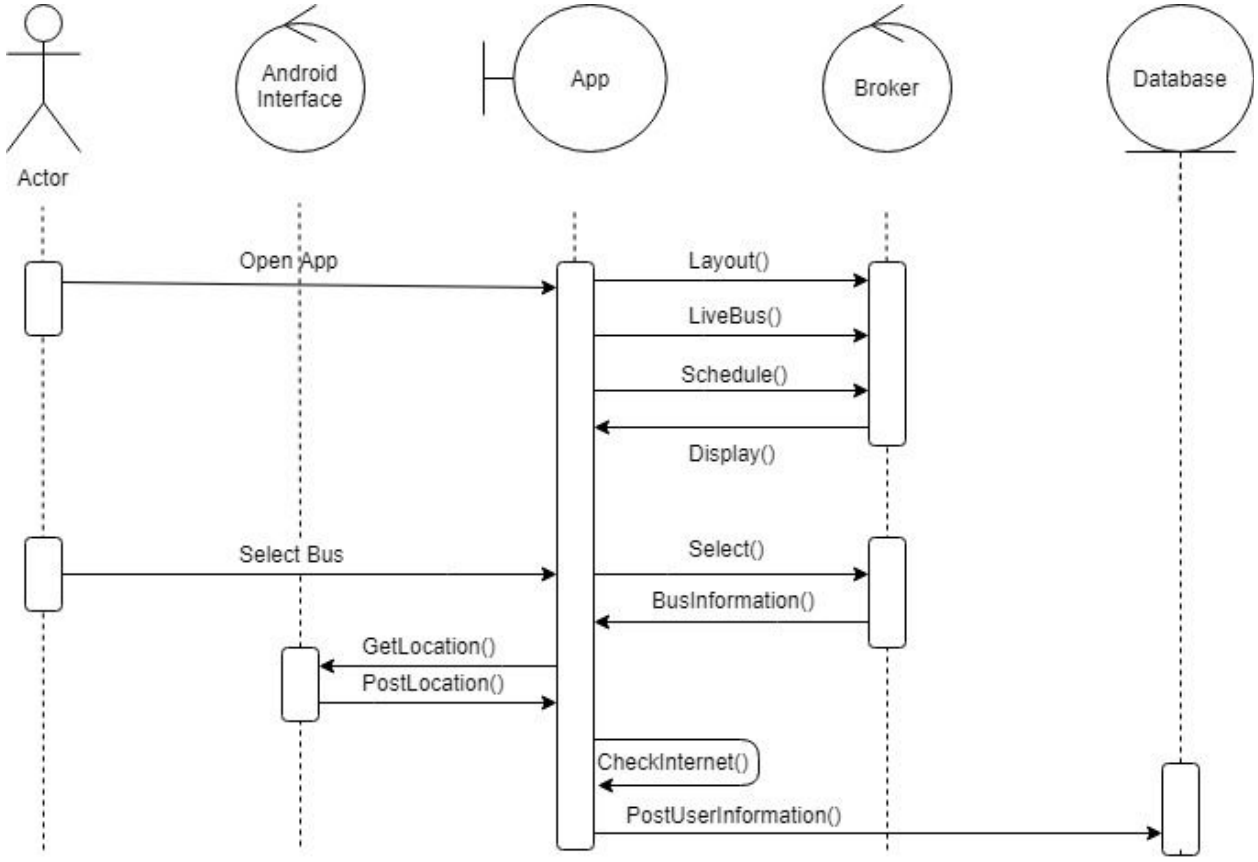
ER Diagram:



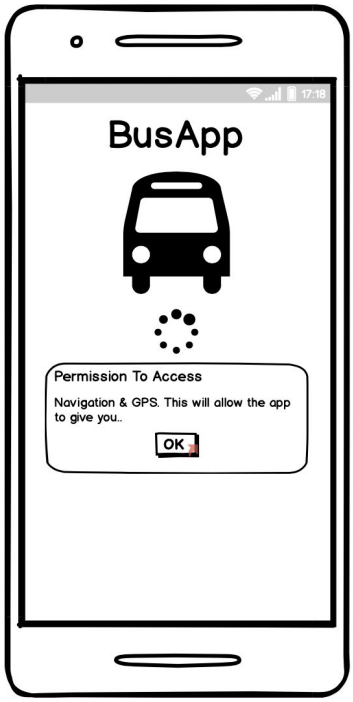
NoSQL Data Model:

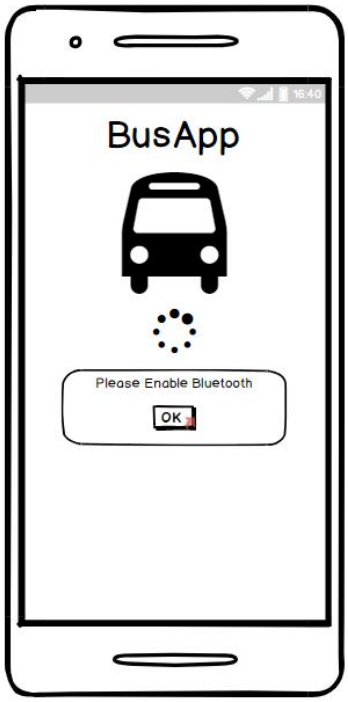

```
[{
   "instID": "4564-adaf-w4f4-3f44",
   "tripID": 3,
   "startTime": "2018-12-22T08:15:00Z",
   …
}, {
   "instID": "4564-adaf-w4f4-3f44",
   "tripID": 4,
   "startTime": "2018-12-24T08:17:00Z",
   …
}, {
   …
}]
```

User Sequence Diagram:



5. UI Mockup

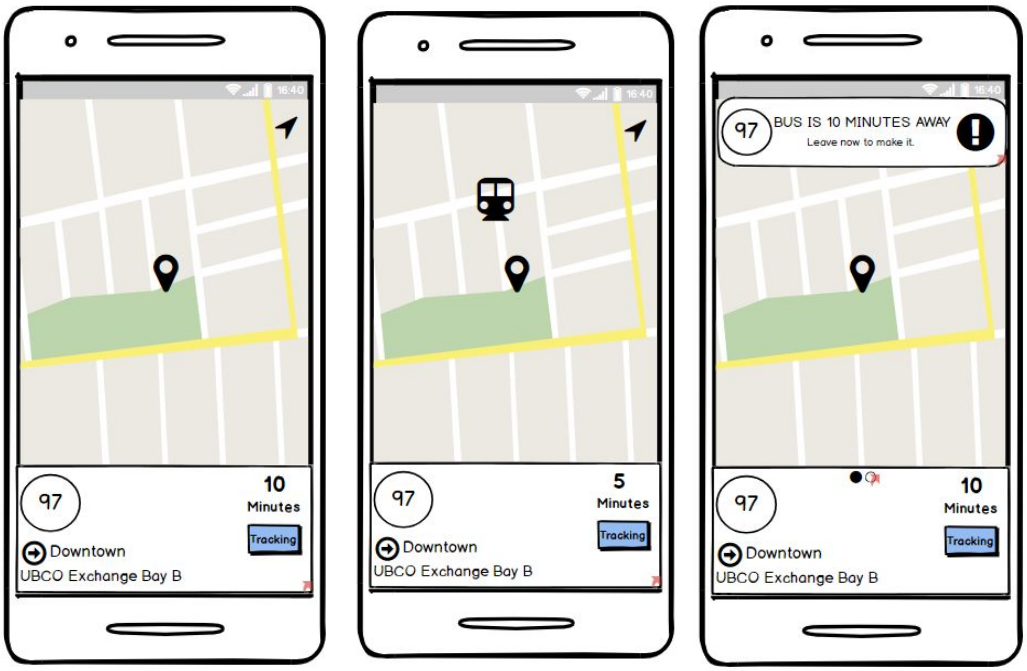

BusApp
Permission To Access
Navigation & GPS. This will allow the app to give you..
OK


BusApp
Please Enable Bluetooth
OK


97
10
Minutes
TRACK
Downtown
UBCO Exchange Bay B


Bus tracker activated!
We will notify you once the bus is nearby.
Ok
97
10
Minutes
Track
Downtown
UBCO Exchange Bay B


Track
10
Minutes
97
Downtown



6. 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	N/A

	Service	
SQLite	Relational Database	N/A
MongoDB	NoSQL database	4.0
AWS	Cloud Platform	N/A
Kontakt.io APIs	REST APIs	N/A
Java	Client side Programming Language	8
Node.js	Backend Programming Language	10.13.0
Postman	API debugging and testing tool	6.5.2
Swagger	API designing and documentation tool	N/A
Here Maps Android SDK	Maps API	3.9

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 will receive UUId, major, and minor from the nearest beacon and do some computations.
- SQLite will be used as the device database.
- JUnit and Espresso will be used for the client side testing.

Server Side:

- The server side will be hosted on a cloud platform (AWS).
- Express will be used as the server.
- Node.js will be used primary backend language and writing REST APIs.
- The response of the REST APIs will be in JSON.

- Swagger will be used for designing the APIs and Postman for debugging and testing.
- The cloud database will be MongoDB.
- Mocha will be used for server side testing.

7. 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 Location 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. Verify app receives information about additional beacons from the server.	Positive Integration Test
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 device is within range of a bus stop's beacon.	Positive Integration Test
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 device (while device is offline) and 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/gateway 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.
Interaction Between Beacon and Gateway	Any interaction not including cloud server or app 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.

8. Approval

Product Manager:_____

Signature: _____

Project Sponsor:___________________________________

Signature: ______________________________