Report Expectations (80%)

◦ (5%) A detailed description of the software you are building
◦ (5%) A list of user groups for your software
◦ (20%) A complete set of usage scenarios for each user group documented as use case diagrams following the UML syntax
◦ (20%) A system architecture diagram documented as ER diagram, data flow diagrams and dynamic models
◦ (10%) The following (depending on the technical specifications of your project):
◦ All the UI mockups needed and any feedback you've gotten from the client to modify your design
◦ All the detailed technical specification of what will be used to build the software (e.g., programming language, database, etc.)
◦ (20%) A detailed test plan explaining the type of testing you will be doing for each feature/component, as well as when those activities will take place. For each type of tests you identify, you must provide one specific example. Note that each diagram must be explained clearly. Do not assume that the reader understands the intentions of your designs

Project Description:

The purpose of the project is to provide a reliable solution to the users for tracking their desired busses without internet using an android application. 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 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 will be installed and configured on a bus/car and at bus stops by the engineering team, which create a mesh network. The application will receive the bus/car information through Bluetooth Low Energy (BLE) provided by the mesh network. 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.

Terminology

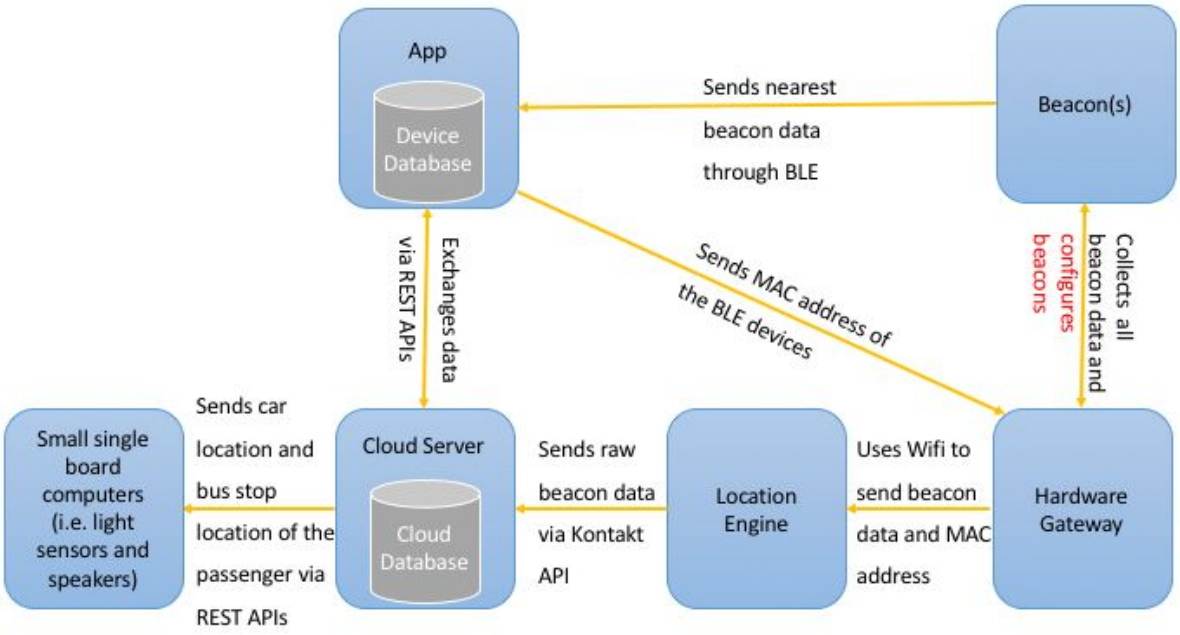
Item	Description
Beacon	A low-energy Bluetooth device used to determine a device's location within a mesh network. Developed by Kontakt.io
Continuous Integration (CI)	Development strategy used to automate integration, testing, and deployment.
Espresso	Java based testing API used to test UI of Android applications
Integration Test	A test that verifies interfaces (interactions) between two or more modules.
Mesh Network	A network of connected beacons allowing for Bluetooth tracking to occur within the network
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.

User Groups:

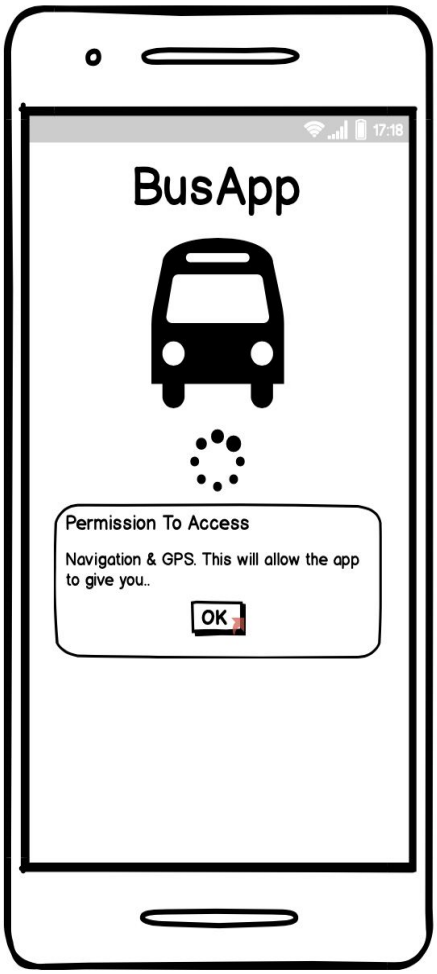
- University Students
- Retirees
- UBC Staff

- Commuters
-

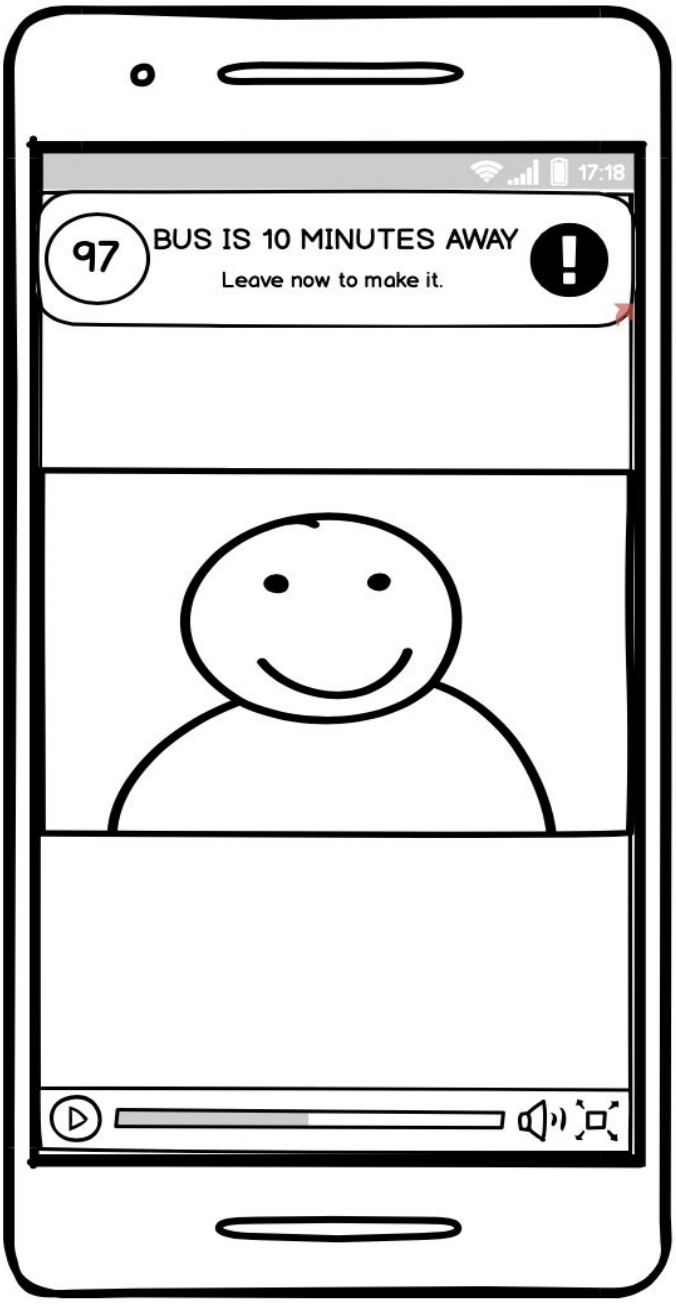
Software Architecture:



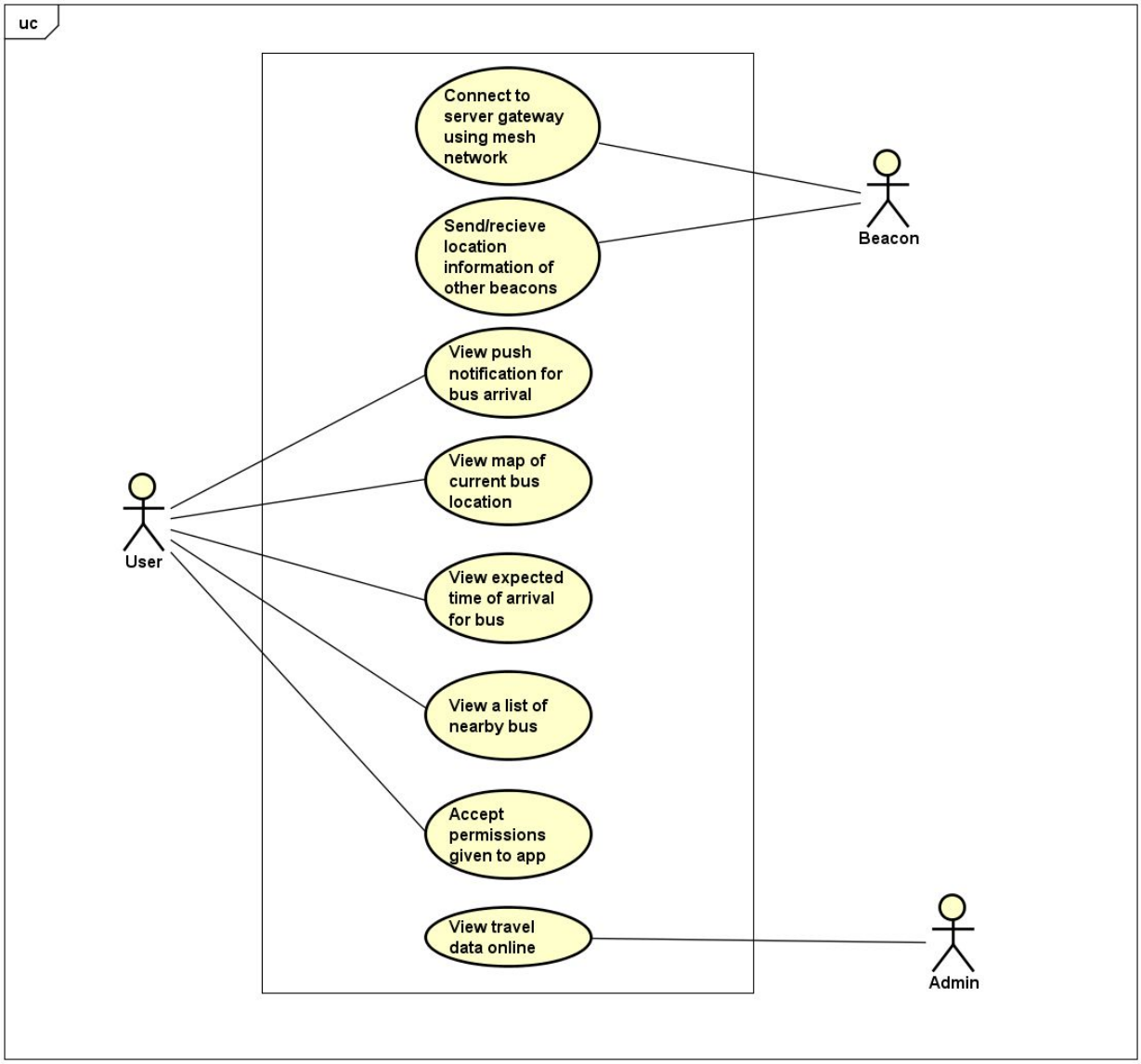
Wireframes:


17:18
BusApp
Permission To Access
Navigation & GPS. This will allow the app to give you..
OK


17:18
Bus tracker activated!
We will notify you once the bus is nearby.
Ok
97
10
Minutes
Track
Downtown
UBCO Exchange Bay B



Use-Case Model:



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

Use-Case Index:

Use-Case ID	Use-Case Name	Primary Actor(s)	Complexity (1-5)	Priority (1-5)
1	Accept Permissions given to App			
2	View Push Notification for Bus arrival			
3	View Map of Current Bus Location			
4	View Expected Arrival Time of Bus			

ID	1
Name	Accept Permissions given to App
Secondary Actor(s)	N/A
Flow of Events	1. App asks user to Enable Location Services 2. User Accepts 3. If Bluetooth is off the ap ask the user to enable it 4. User enables bluetooth
Pre-conditions	The user has location services off and does not have bluetooth enabled
Descriptions	

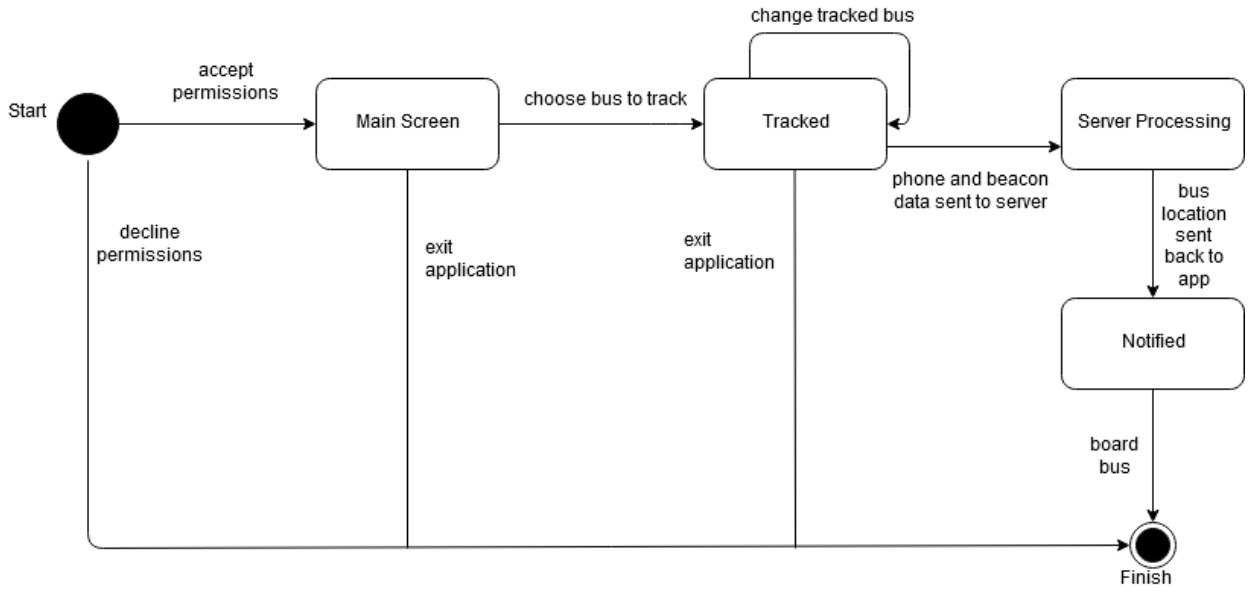
ID	2

Name	**View Push Notification for Bus arrival**
Secondary Actor(s)	**Beacon**
Flow of Events	**1. Beacon send location of bus to users phone 2. The app triggers the notification that tells the user where the bus is**
Pre-conditions	**The user is connected via bluetooth & has enabled location services**
Description	**A user will get a push notification sent to their phone even if the app isn't on screen**

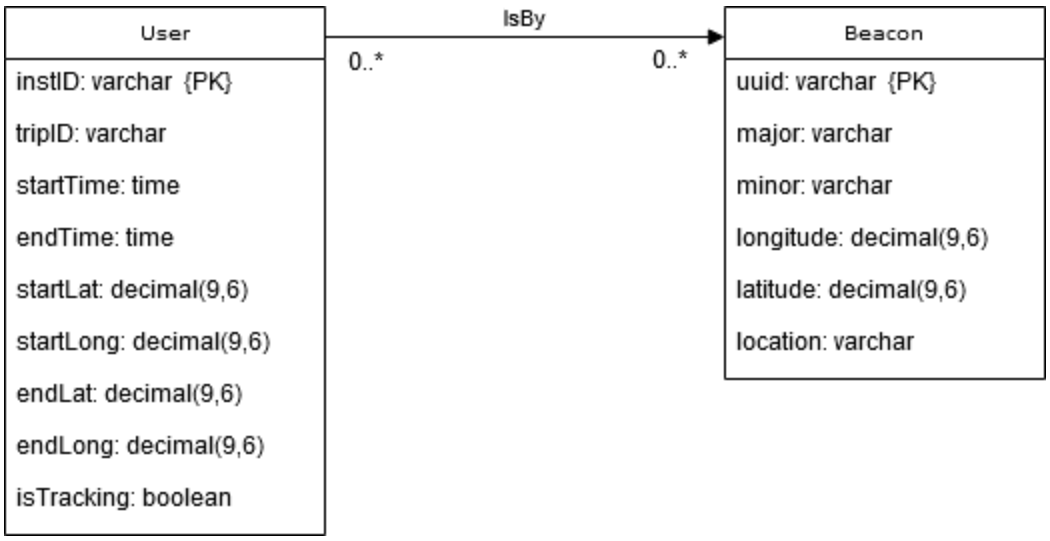
ID	**3**
Name	**View Map of Current Bus Location**
Secondary Actor(s)	**Beacon**
Flow of Events	**1. User clicks on bus 2. Bus then**
Pre-conditions	**The user is connected via bluetooth & has enabled location services**
Descriptions	

ID	**4**
Name	**View Expected Arrival Time of Bus**
Secondary Actor(s)	**Beacon**
Flow of Events	**The user is connected via bluetooth & has enabled location services**
Pre-conditions	
Descriptions	

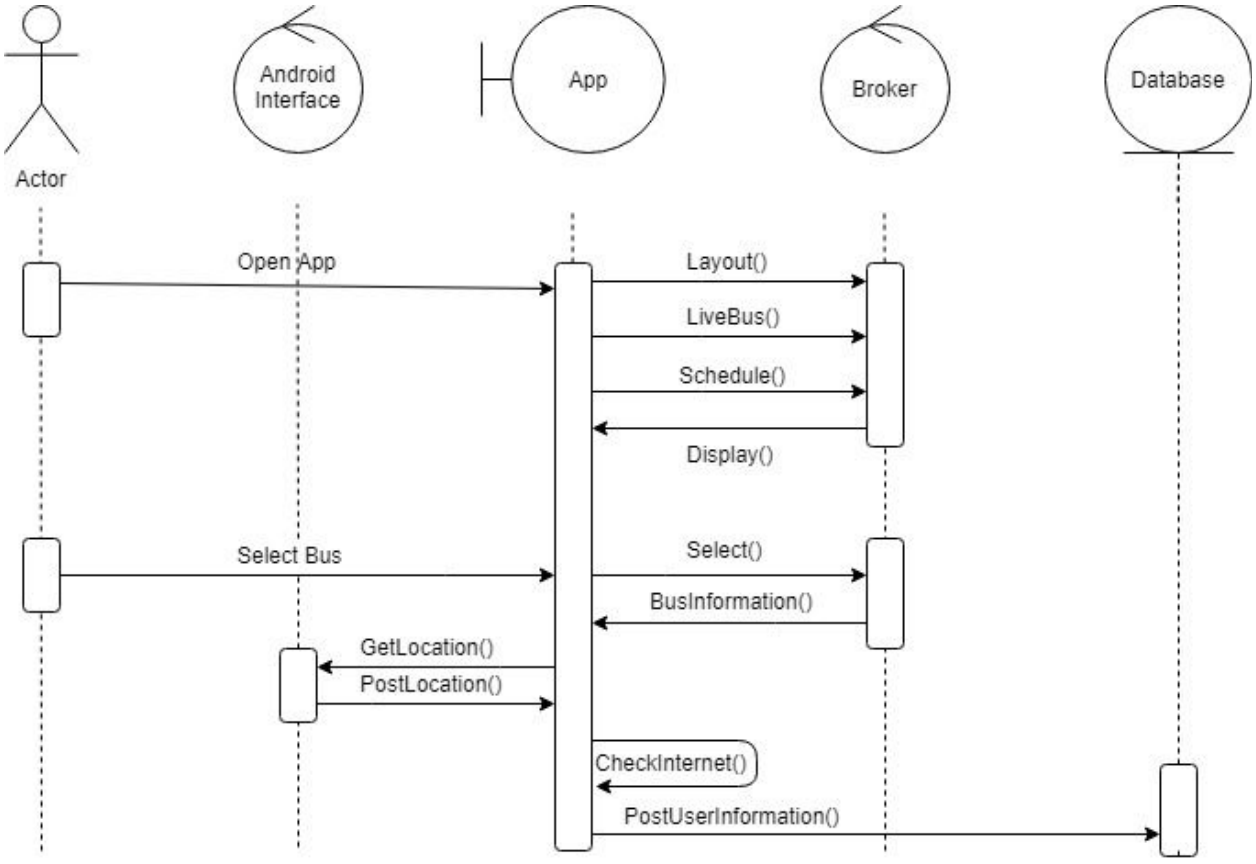
State Diagram:



ER Diagram:



User sequence diagram:



Technical Specifications:

Tools/Languages/Libraries	Type	Version
Android Studio	Development Environment	3.2.1
Oreo	Android OS Version	8
Jenkins/Travis	CI Solution	
Git	Version Control System	
Github	Version Control Hosting Service	
SQLite	Relational Database	
MongoDB	NoSQL database	
AWS/Firebase	Cloud Platform	
Kontakt.io APIs	REST APIs	
Java	Client side Programming Language	8
Node.js	Backend Programming Language	
Postman	API debugging and testing tool	
Swagger	API designing and documentation tool	
Here Maps	Maps API	

Client Side:

- The client side (android app) will be built with Java and XML using Android studio.
- Here Maps API 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/Firebase).
- 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.

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 Schedule

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

- Unit tests will be prepared prior to their respective module's development. Unit tests will be executed upon completion of the module. Defects must be fixed prior to any integration of the module.
- 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.

Regression tests contain all previously passed integration and system tests. 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.

Test Cases

User Input Validation

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. The input validation module will be tested at the unit level using a boundary test. As an example, one test will verify that the module can detect that the given bus beacon id is not a valid beacon id. Additionally, an integration test will be conducted to verify that only accepted data is added to the database and rejected data is correctly discarded.

API Testing

UI Testing

Espresso will be used to run unit tests for UI functionality. Unit tests will verify the UI behaves as expected. User testing will also be used to test the usability of the UI.

Performance Non-Functional Requirements

Out of Scope and Exclusions

Item	Description
Functionality of Extra Hardware	Functionality of hardware outside of beacons and android device will not be involved in integration testing. Such devices are out of scope.

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 beacon network, which is not possible on a cloud or server-based emulator. (Note: Such tests will still be conducted manually)

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.
- Travis CI - CI solution (Tests should be written in Java).
- Mocha - Test framework for node.js.