National College of Ireland
BSc in Computing
2017/2018

Aaron Meaney
14326016
14326016@student.ncirl.ie

Bus Stop!

Technical Report

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name: Aaron Meaney
Student ID: 14326016
Supervisor: Dominic Carr

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: Aaron Meaney Date: 13/05/2018

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Complete the sections above and attach it to the front of one of the copies of your assignment,

Table of Contents

Executive Summary 5

1 Introduction 6

1.1 Background 6
1.2 Aims 6
1.3 Technologies 9
1.3.1 Unity 9
1.3.2 Mapbox SDK for Unity 9
1.3.3 Visual Studio IDE 9
1.3.4 PubNub 9
1.3.5 Sinatra 9
1.3.6 PostgreSQL 9
1.3.7 Google Maps API 10
1.3.8 Android Studio 10
1.3.9 Heroku 10

2 System 11

2.1 Requirements 11
2.1.1 Functional Requirement 1: User Views Realtime Map 11
2.1.2 Functional Requirement 2: User Hails Bus 13
2.1.3 Functional Requirement 3: Bus Sends Bus Data 15
2.1.4 Data requirements 17
2.1.5 User requirements 18
2.1.6 Environmental requirements 19
2.1.7 Usability requirements 19
2.2 Design and Architecture 20
2.3 Implementation 23
2.4 Graphical User Interface (GUI) Layout 27
2.5 Testing 31

3 Conclusions 32

4 Further development or research 33

4.1 Predictive Analysis for Time Slots 33

4.2 Full Traffic Simulation 33
4.3 Bus Alerts 33
4.4 Other Industries 33
5 References 34
6 Appendix 35
6.1 Survey Results 35
6.2 Project Proposal 38
6.3 Requirements Specification 46
6.4 Monthly Journals 84
6.4.1 September 84
6.4.2 October 85
6.4.3 November 86
6.4.4 February 86
6.4.5 March 87
6.4.6 April 87
6.4.7 May 87

Executive Summary

“Bus Stop!” is a proof-of-concept Android application that allows the user to view the live position, capacity and miscellaneous information of busses in a public transport network. The app also gives the user the ability to hail a bus remotely through the use of the app, for situations where the user cannot hail the bus at a stop. For example, the user is outside the line of sight of the bus or multiple busses are pulling into a stop and the user needs to hail one.

This proof-of-concept prototype is powered by a simulation running on the Unity game engine which provides the Android application with public transport data. Short term data (such as bus position, name, assigned route, etc) is transmitted between the Android app and Unity simulation by the PubNub networking service, while long term data (such as bus routes, route waypoints, bus stops) is stored on a PostgreSQL database running on a Sinatra server hosted by Heroku.

While the project was developed to a Minimum Viable Product, it was originally intended to have a much larger scope. The project manages to communicate its idea in its current form, however with more time it would have been able to do this even more so. Nonetheless, great care was put into the design, development and implementation of the project and despite the lack of time to finish it to the original goal, I am very proud of what I accomplished with it.

1 Introduction

1.1 Background

For my whole life I've depended on public transport. Because of this, my only way to reach the college is by bus. I've taken the same bus for the past four years and many times I have missed the bus just as I turned the corner to the bus stop. Having to wait for 30 minutes for the next bus, and by extension miss classes and appointments, is extremely annoying. This made me consider the development of a project that would improve the public transport experience.

During the process of considering what projects to develop, I liked the idea of developing a project with the Unity game engine. I'm personally very passionate about game development and I've used this engine in my spare time for hobbyist game development. I wanted to become more familiar with the engine and to improve my skills with using it. I also wanted to develop a project that had an IoT application, and so the idea of creating a simulation of IoT connected busses was conceived.

1.2 Aims

My goals for the development of this project are:

1. Develop a proof-of-concept Android app that:
    1.1. Allows users to view real-time bus data on a Google Map interface
    1.2. Allows users to hail a bus at a specified stop
    1.3. Reads in data from the Unity simulation
2. Develop a backend Unity simulation that:
    2.1. Simulates a public transport bus system composed of:
        2.1.1. Busses
        2.1.2. Bus Stops
        2.1.3. Bus Routes
        2.1.4. Bus Services
        2.1.5. Bus Timetables

    2.1.6. Bus Time Slots
  2.2. Sends real-time bus data to the Android app
  2.3. Predictably responds to hail commands from the Android app
  2.4. Allow the user to configure the simulation with different simulation components, E.g. routes, stops, timetables, busses, etc
  2.5. Allow the user to configure the bus timetables through a custom user interface
3. Develop a backend web service (Sinatra + PostgreSQL) that:
  3.1. Stores long-term important data for the application
4. Integrate a MQTT style network service (PubNub) that:
  4.1. Transmits messages between the app and the simulation

The basic flow of the system will be:

The user will configure the Unity simulation by adding bus stops, waypoints, bus routes, bus companies, and finally bus timetables to the main scene. The user can use a custom-built Timetable UI (built as a Unity Editor Window) to modify bus timetable data. The Timetable UI will mimic the display of a real bus timetable with multiple routes and will display the busses in the correct order by using a topological sort on the bus stops against the bus routes.

When the simulation is started, it will send the data that it has been configured with to the web server. The web server will receive this JSON message and will then encode the data as base64 to prevent any escape characters from interfering with the language interpreter, causing unintended behaviour. The Sinatra server will then save the data in a key-value-pair table in the PostgreSQL database.

Once the Unity simulation is fully initialized, it will then check the timetables that have been configured by the user. From this timetable data, the scheduler will dispatch busses to the appropriate services/routes once the current time reaches the time of these bus time slots.

Once a bus begins a service, it will broadcast this event (over PubNub) and it will service the first stop. For each stop that the bus stops at, it will let off any passengers that want to get off at that stop and will then allow passengers to embark if the bus isn't full. Passengers will only embark on a bus if it is going to their destination stop.

Once the bus gets close to a bus stop, if a passenger wants to get off at that stop or a passenger is waiting at that stop and wants to get onto the bus, the bus will be hailed. If a bus is hailed to a stop, it will service that stop once it reaches it.

Once a bus reaches the end of a bus route, it will let off everyone and will despawn, sending an end service event to PubNub, and returning to the bus companies' pool of busses.

While the bus is in service, it will transmit its current state to PubNub. The Android app will listen for this real-time data once it receives the simulation data from the Sinatra server.

The Android app will render a map of the bus stops in the simulation, along with markers representing the real-time position of the busses. If a user taps a bus icon, they will see that bus's visible route on the map represented by a line, alongside a list of bus stops that the bus has yet to visit. The bus's current capacity and other miscellaneous information will also be shown.

1.3 Technologies

1.3.1 Unity

The Unity Game Engine is the platform that the simulation is built on and is a main technology for this project. The components that make up the “Bus Stop!” simulation is built in C# and are put together in Unity.

1.3.2 Mapbox SDK for Unity

The Mapbox SDK adds Mapbox specific functionality to Unity. For example, the map of Dublin is generated using this SDK, as well as coordinate conversions.

1.3.3 Visual Studio IDE

Visual Studio IDE was used to write the C# code for the simulation, used in Unity. The VS debugger was also used, as Visual Studio can attach to the Unity instance to read the call stack and read watched variables to improve the debugging experience.

1.3.4 PubNub

PubNub was used to communicate fast, real-time messages between the Android application and the Unity simulation. For example, bus data.

1.3.5 Sinatra

Sinatra was used as the web service platform as it exposes an easy to use REST API with a low learning curve to develop on it. Sinatra was used to interface with PostgreSQL.

1.3.6 PostgreSQL

PostgreSQL was used to store important data on the Sinatra server in a key-value pair database. For example, bus stop and bus waypoint data.

1.3.7 Google Maps API

The Google Maps API was used to render the Google Maps display on the Android app.

1.3.8 Android Studio

Android Studio was used to develop the Android application.

1.3.9 Heroku

Heroku was used to deploy the Sinatra web server and to host the PostgreSQL database.

2 System

2.1 Requirements

Please see the appendix for previous functional requirements in the Requirements Specification.

2.1.1 Functional Requirement 1: User Views Realtime Map

Description & Priority:

Priority: High

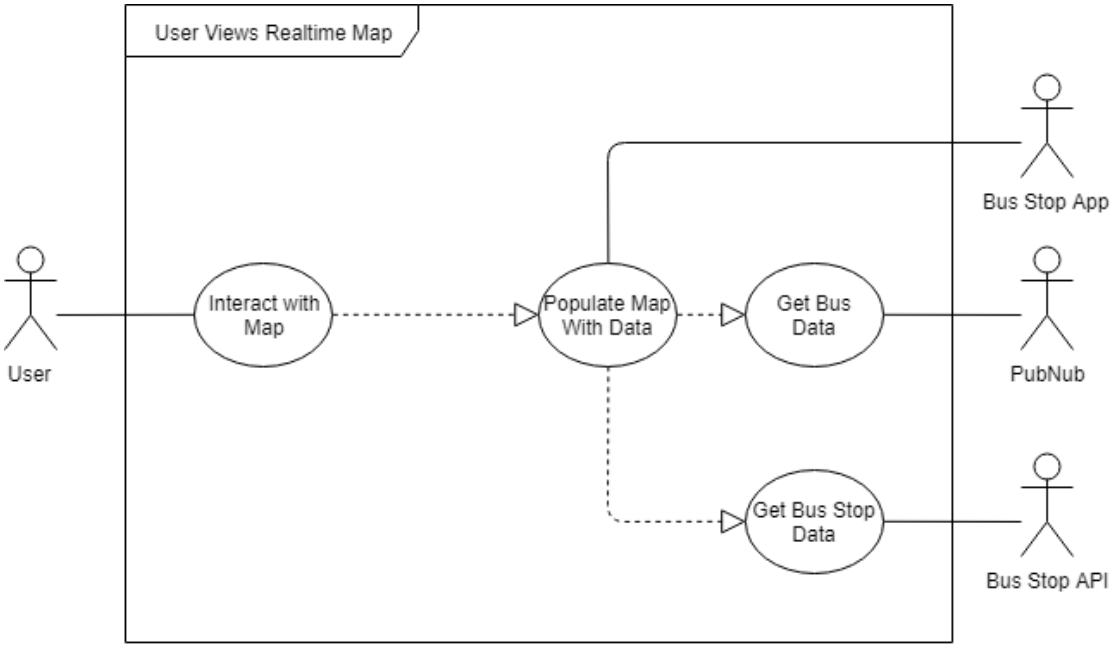
The user can view a Google Maps interface when they open the app. The app should display all the Bus Stops and Busses once the API is ready. The app should update the positions of Busses once the map receives that Bus's message from the API.

Use Case

Scope: The user should be able to navigate the populated map, and the bus markers should update in real time.

Description: This use case describes the process of getting the bus stop, bus route and live bus data from the Bus Stop API.

Use Case Diagram:



Flow Description

Precondition: The app has a connection to the internet. Location Services are Enabled.

Activation: The use case begins when the user opens the map.

Main Flow:

1. The Bus Stop App gets the Bus Data from PubNub.
2. The Bus Stop App gets the Bus Stop Data from the Bus Stop API.
3. The Bus Stop App populates the Map.
4. The User interacts with the Google Map.
5. The Main Flow returns to 1.

Termination: The use case terminates when the app closes.

Post Condition: The map contains up to date information.

2.1.2 Functional Requirement 2: User Hails Bus

Description & Priority:

Priority: High

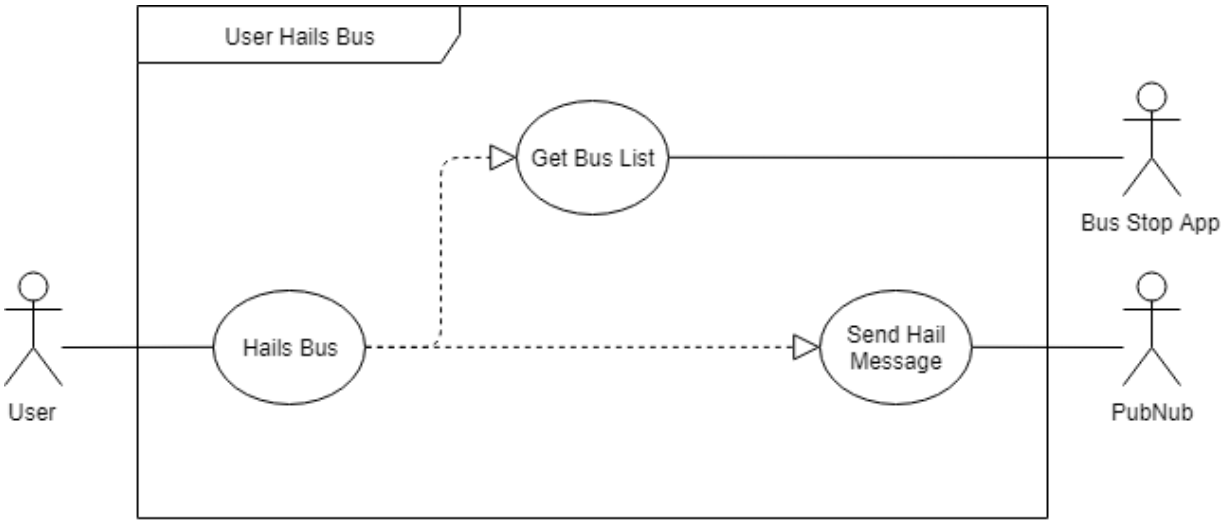
Once the map is initialized and the busses are being simulated, the user can hail the bus through the mobile app.

Use Case

Scope: The user should be able to hail the bus through the mobile app.

Description: This use case describes the process of sending a hail bus message to the Bus Stop Simulation through PubNub.

Use Case Diagram:



Flow Description

Precondition: The map is populated, and the user has selected a bus.

Activation: The use case is activated when the user presses the “Hail” button on one of the bus stop lists.

Main Flow:

1. The User presses the Hail button
2. The Bus Stop App provides the User with the Bus Id and Bus Stop Id of the hailed Bus

3. PubNub sends the hail message with the Bus Id and Bus Stop Id.

Termination: The use case terminates after the hail message is sent.

Post Condition: The bus that the user selected is hailed.

2.1.3 Functional Requirement 3: Bus Sends Bus Data

Description & Priority:

Priority: High

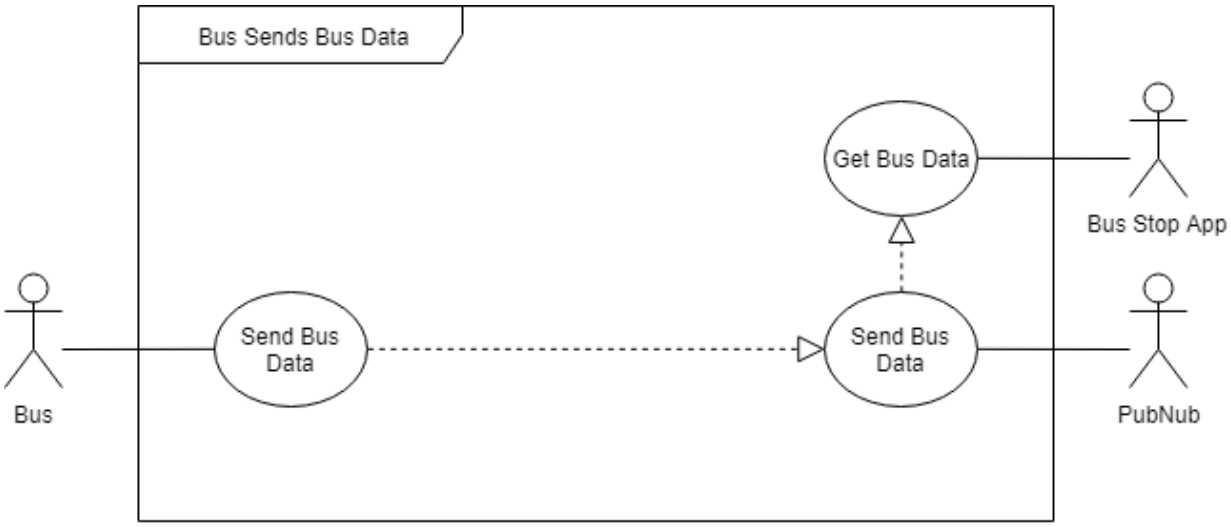
The Bus in the Unity Simulation can send Bus Data to PubNub.

Use Case

Scope: This use case considers an individual bus in the Unity simulation sending data to PubNub.

Description: In this use case the Bus pushes data up to PubNub on a regular interval for consumption by the Android app.

Use Case Diagram:



Flow Description

Precondition: The Unity Simulation is running and is connected to the internet. The Bus Stop App is connected to the internet and Location Services are enabled.

Activation: The use case begins when the Bus enters service.

Main Flow:

1. The Bus sends its data to the PubNub API
2. The PubNub API sends the Bus Data

3. The Bus Stop App gets the Bus Data from PubNub

Termination: The use case ends when the Bus exits its service.

Post Condition: The Bus Stop App received the Bus Stop Data.

2.1.4 Data requirements

The data model must exist on the Unity, and parts can be transferred to the Android app.

The data model that describes the bus system must be composed of:

Bus	Belongs to a Bus Company. Has a Bus Service. Has many Bus Passengers.
Bus Passenger	Belongs to a Bus or a Bus Stop.
Bus Stop	Has many Bus Passengers. Belongs to Bus Waypoint. Belongs to Bus Time Slot.
Bus Waypoint	Has a Bus Stop. Belongs to many Bus Routes.
Bus Route	Has many Bus Waypoints. Belongs to Bus Company.
Bus Time Slot	Has a Bus Stop. Has a Bus Service. Belongs to a Timetable.
Bus Service	Belongs to Bus. Belongs to Time Slots.
Bus Timetable	Has many Time Slots. Belongs to a Bus Company.
Bus Company	Has many Busses. Has many Bus Timetables. Has many Bus Routes.

2.1.5 User requirements

Must Have

The User can Navigate the Map on the App

The User can see live bus information by tapping marker icons on the Map

The User can see the selected bus's list of upcoming Bus Stops

The User can hail Busses using the App

The Busses can automatically route themselves in the Simulation

The Busses can follow timetable directions in the Simulation

The Unity User can configure the Simulation parameters

Should Have

The User can search for Bus Stops and Busses on the App

The User can see the Timetable data for each Bus Stop

The User can receive Predictive Analysis for Time Slots

Traffic is simulated by using a heatmap and by capping the bus speed

Could Have

Simulated Person system is implemented (See original Req Specification)

The User can get alerts when their Bus is near a stop

The Unity User can see Debug Information Panels in the Simulation

The Android App runs a security check to ensure it is not blacklisted

Busses deploy from and return to set Bus Depot locations in the Simulation

Traffic is fully simulated

Would Have

Detailed Simulated Person system (See original Req Specification)

2.1.6 Environmental requirements

The Unity simulation needs to run on a machine that has the graphical performance equivalent to or greater than a Nvidia GTX960m Graphics Card to run smoothly and accurately. The machine also needs a stable internet connection with ping less than 30ms, download speed greater than 10 Mbps, upload speed greater than 2Mbps.

The Sinatra web service needs to run on a Heroku server with a dedicated PostgreSQL database hosted and connected to the same account. A free Heroku account is sufficient to run the server application

The Android App needs to run on an Android OS of version equal to or greater than Android 7.0 Nougat (API Level 24). The user needs to allow the app to use ACCESS_FINE_LOCATION, INTERNET and ACCESS_NETWORK_STATE permission. The device needs a stable internet connection with less than 30ms ping, download speed greater than 10 Mbps, upload speed greater than 2 Mbps.

2.1.7 Usability requirements

The Android application must only contain one activity to make sure the user won't get lost navigating the application. The Bottom Sheet must be responsive to the user pressing it and the application UI state must be consistent to the context of that application; for example, if a user has tapped a bus stop, don't' show a different bus's list of upcoming bus stops. Instead, hide the bottom of the Bottom Sheet to prevent confusion to the user.

The user interface should be easy to understand by just looking at the app view; everything should be self-explanatory. Icons should be used instead of text where applicable. For example, use a hand icon instead of the words "hail" to represent a button to hail a bus.

2.2 Design and Architecture

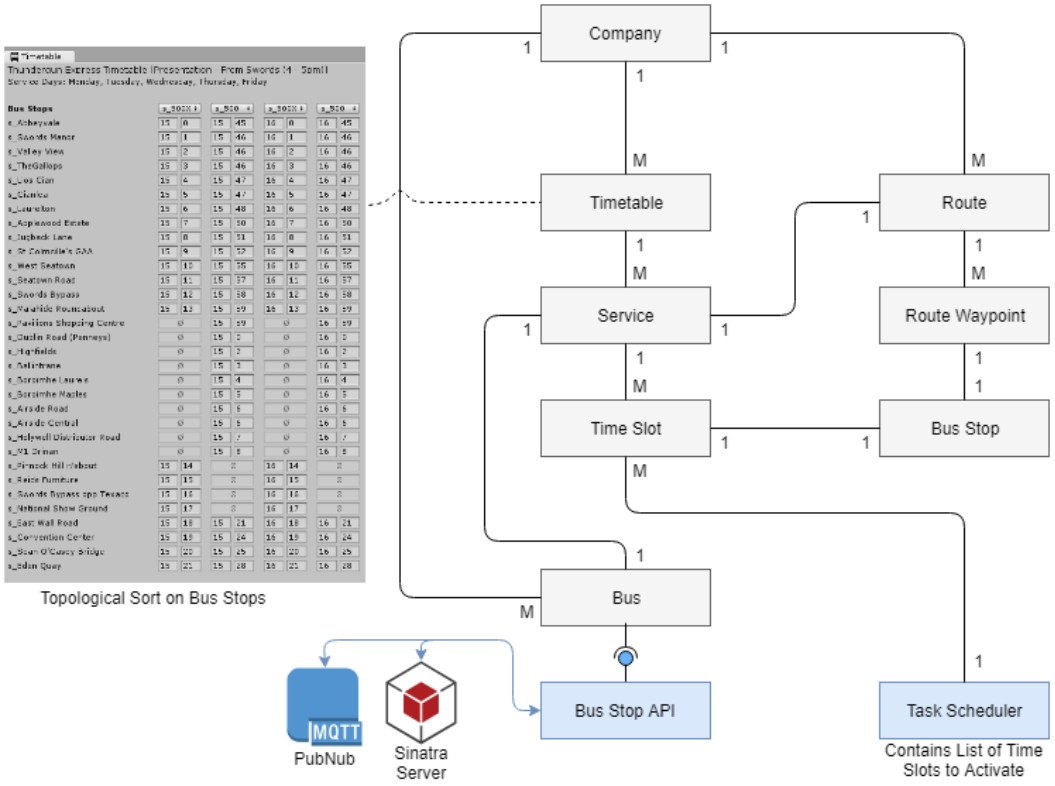


Figure 1 - Bus Stop Simulation System Architecture (Simplified Bus Stop API Interface)

In the above diagram *(Figure 1)* you can see the general architecture of the Unity simulation. This diagram shows 3 things: the entity relationships between the bus simulation objects (grey boxes), the relation between the bus system and the Task Scheduler / Bus Stop API Managers, and the Unity Editor Window display of the Timetable.

In the simulation, the Company is the root object for the rest of the entities that run the simulation. The Route Waypoints define absolute positions on the road that any other Route from any other Company can pass through, and Bus Stops can be linked to these Route Waypoints. This allows for cases where multiple Companies and Routes will share a bus stop. Routes are a list of Route Waypoints (coordinate data) that belong to a company, but they don't hold any scheduling data. Routes, Waypoints, Timetables and Busses can all be directly modified with the default Unity hierarchy and inspector *(Figure 2)*.

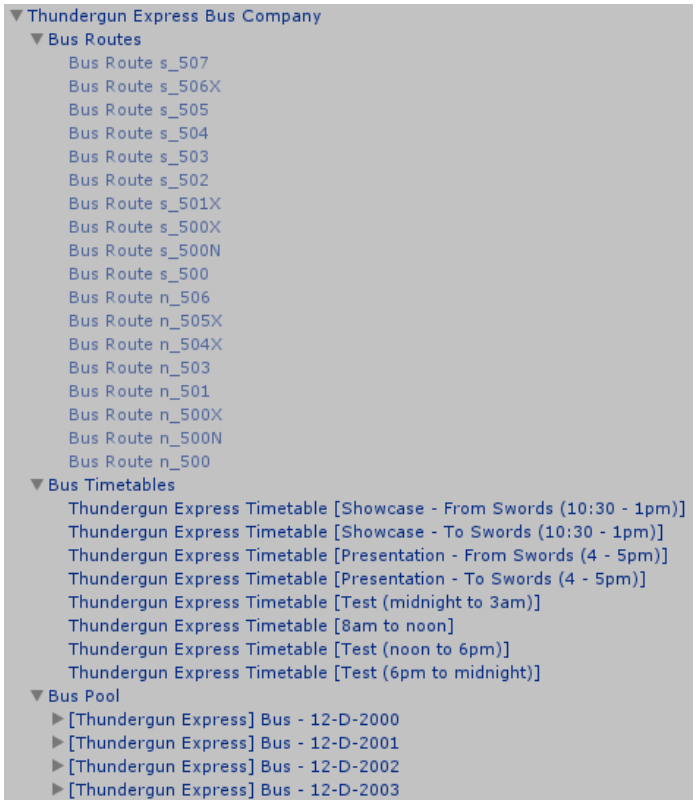
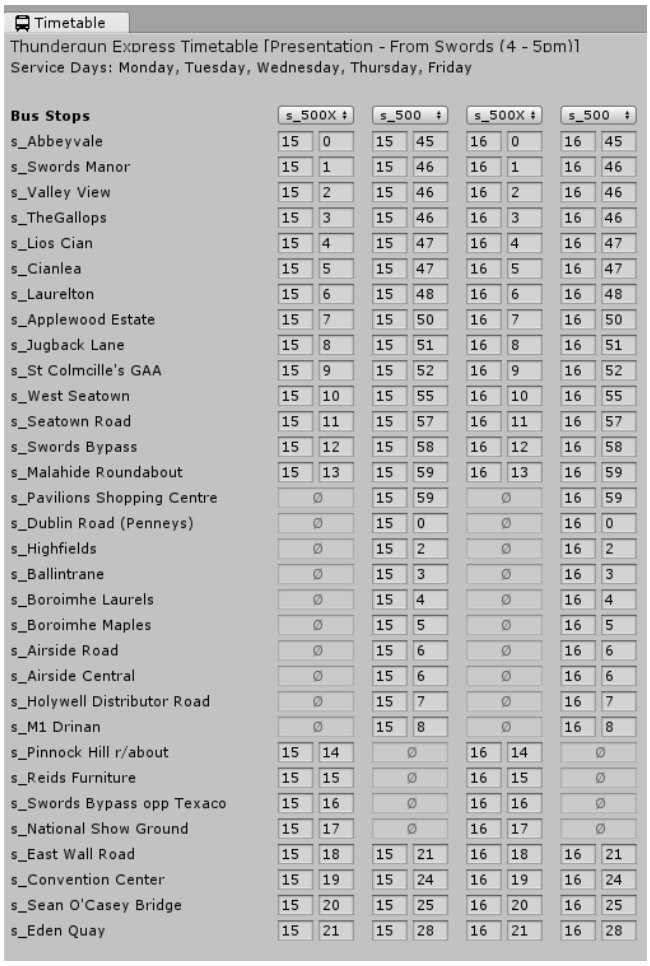


Figure 2 - Hierarchy of the Bus Company

Scheduling data is contained by the Timetable, Time Slot and Service objects. Services hold the scheduling information for a Route, as Routes are just a list of coordinates for a Bus to follow. A Timetable simply contains a list of Services, which in turn contains a list of Time Slots. The Time Slots contain the time data for each Bus Stop in the Service's Route. In the context of rows and columns, the Service would be considered a column and a Bus Stop would be considered a row. The intersection is the Time Slot for that Bus Stop, on that Service/Route.



Bus Stops	s_500X		s_500		s_500X		s_500	
s_Abbeyvale	15	0	15	45	16	0	16	45
s_Swords Manor	15	1	15	46	16	1	16	46
s_Valley View	15	2	15	46	16	2	16	46
s_TheGallops	15	3	15	46	16	3	16	46
s_Lios Cian	15	4	15	47	16	4	16	47
s_Cianlea	15	5	15	47	16	5	16	47
s_Laurelton	15	6	15	48	16	6	16	48
s_Applewood Estate	15	7	15	50	16	7	16	50
s_Jugback Lane	15	8	15	51	16	8	16	51
s_St Colmcille's GAA	15	9	15	52	16	9	16	52
s_West Seatown	15	10	15	55	16	10	16	55
s_Seatown Road	15	11	15	57	16	11	16	57
s_Swords Bypass	15	12	15	58	16	12	16	58
s_Malahide Roundabout	15	13	15	59	16	13	16	59
s_Pavilions Shopping Centre	Ø		15	59	Ø		16	59
s_Dublin Road (Penneys)	Ø		15	0	Ø		16	0
s_Highfields	Ø		15	2	Ø		16	2
s_Ballintrane	Ø		15	3	Ø		16	3
s_Boroimhe Laurels	Ø		15	4	Ø		16	4
s_Boroimhe Maples	Ø		15	5	Ø		16	5
s_Airside Road	Ø		15	6	Ø		16	6
s_Airside Central	Ø		15	6	Ø		16	6
s_Holywell Distributor Road	Ø		15	7	Ø		16	7
s_M1 Drinan	Ø		15	8	Ø		16	8
s_Pinnock Hill r/about	15	14	Ø		16	14	Ø	
s_Reids Furniture	15	15	Ø		16	15	Ø	
s_Swords Bypass opp Texaco	15	16	Ø		16	16	Ø	
s_National Show Ground	15	17	Ø		16	17	Ø	
s_East Wall Road	15	18	15	21	16	18	16	21
s_Convention Center	15	19	15	24	16	19	16	24
s_Sean O'Casey Bridge	15	20	15	25	16	20	16	25
s_Eden Quay	15	21	15	28	16	21	16	28

Figure 3 - Timetable Example

The custom Timetable Editor *(Figure 3)* allows the user to modify the Time Slot and Services of the Time Stamp, as they can't be directly edited with the Unity default inspector. Different routes can be displayed on the same timetable, just like a real bus timetable! Since all the Bus Routes in the Timetable can be logically represented as a digraph, a Topological Sort can be applied to them to sort them in the same manner as a normal timetable. The Topological Sort is applied dynamically as Routes are removed and added to the timetable.

Once the simulation starts, the Task Scheduler is invoked by all the Time Slots, adding a callback to activate the calling Time Slot. Once a Time Slot is activated, the Company dispatches a Bus to that service. The Time Slot is then scheduled to be called again the next day it's available. When the bus is dispatched, it begins calling the Bus Stop API and it sends its data to PubNub.

2.3 Implementation

There is no one main method in the Unity simulation. A combination of the **Task Scheduler** and **C# Delegates/Actions** allowed for the development of an asynchronous system that simulates the operation of a bus route.

The entry point for this behaviour is after Unity loads the map scene. Each timetable waits for the map visualiser to loads. Once this occurs, the timetables initialize their timeslots. See *(Code Snippet 1).*

```
private void Awake()
{
    taskRunner = FindObjectOfType<ScheduleTaskRunner>();
    dateTimeManager = FindObjectOfType<DateTimeManager>();

    // Initialize Time Slots when the Map is finished loading
    MapVisualizer.OnMapVisualizerStateChanged += (s) =>
    {
        if (s == ModuleState.Finished)
        {
            InitializeTimeSlots();
        }
    };
}
```

Code Snippet 1 - BusTimetable.cs (95-108)

“InitializeTimeSlots();” calls the Initialize method on each Time Slot. This sets the Time Slot’s isInitialized = true and calls “ScheduleTimeSlot();” on that Time Slot. This sets up the Time Slot to get activated once it reaches its scheduled time. This works by passing “ActivateTimeSlot” as a delegate to a ScheduledTask and then enqueuing that task to the Task Scheduler (a.k.a. ScheduledTaskRunner). See *(Code Snippet 2).* The Task Scheduler queue is pre-sorted from soonest callback time to latest callback time because every enqueue performs an ordered insert.

```
/// <summary>
/// Schedules this <see cref="BusTimeSlot"/>'s activation with the <see
cref="ScheduleTaskRunner"/>.
/// </summary>
private void ScheduleTimeSlot()
{
    DateTime currentDateTime = dateTimeManager.CurrentDateTime;
    List<DayOfWeek> runningDays = Service.ParentBusTimetable.DaysRunning();

    // Set scheduled date time
    DateTime scheduledDateTime = new DateTime(currentDateTime.Year,
currentDateTime.Month, currentDateTime.Day, scheduledHour, scheduledMinute, 0);

    // If the scheduled time is in the past or does not take place on a scheduled day,
advance day by 1 and check again
    while (DateTime.Compare(scheduledDateTime, currentDateTime) < 0 ||
!runningDays.Contains(scheduledDateTime.DayOfWeek))
    {
        scheduledDateTime = scheduledDateTime.AddDays(1);
    }

    // Create Scheduled Task
    ScheduledTask task = new ScheduledTask(ActivateTimeSlot, scheduledDateTime);
    taskRunner.AddTask(task);
}
```

Code Snippet 2 - BusTimeSlot.cs (106, 126)

Once the “ActivateTimeSlot” method is queued in the Task Scheduler, it is only a matter of time before it is called back. The Task Scheduler will check the list at each frame by calling “ExecuteReadyTasks” and when the front item is ready to be called (by comparing its timestamp to now), the Task Scheduler will dequeue and execute that Task Scheduler’s callback. The Task Scheduler will continue executing and dequeuing from the queue during this frame until it reaches a Task that is not ready to be scheduled. Once this happens, it stops and waits for the next frame. Because the list is sorted, this results in a very small performance hit as the loop will break very quickly. See *(Code Snippet 3).*

```
private void ExecuteReadyTasks()
{
    while (taskList.Count > 0 && DateTime.Compare(taskList[0].ScheduledDateTime,
dateTimeManager.CurrentDateTime) < 0)
    {
        taskList[0].ExecuteTask();
        taskList.Remove(taskList[0]);
    }
}
```

Code Snippet 3 - ScheduleTaskRunner.cs (30, 40)

Once the Time Slot is activated, it sets its service's "Scheduled Time Slot" to itself. This runs code in a C# property that starts the Time Slot's Service on this Time Slot if the Service has not already been started. This is done by calling DeployBus() on the Time Slot's Company. This in turn sets the bus's position, places it in the scene and starts the Service. The bus will then drive along its route until it gets hailed by other Bus Passengers waiting at a stop. They hail the bus once the bus gets close enough to their bus stop, which is implemented as an Action. See *(Code Snippet 4).*

```
public BusPassenger(BusStop originBusStop, BusStop destinationBusStop)
{
    …

    // Hail the bus once it approaches and it is going to the passenger's destination
    originBusStop.OnBusApproach += HailBus;
}

…

/// <summary>
/// The passenger will try to hail the bus.
/// </summary>
private void HailBus(Bus bus)
{
    Debug.Log("OnBusApproach called for " + bus.RegistrationNumber);
    if (bus.CurrentRoute.BusStops.Contains(destinationBusStop) &&
!bus.HailedStops.Contains(originBusStop))
    {
        Debug.Log(FullName + " hailed " + bus.RegistrationNumber + " to " +
originBusStop.BusStopIdInternal + " because it is going to " +
destinationBusStop.BusStopIdInternal);
        bus.Hail(originBusStop);
        originBusStop.OnBusApproach -= HailBus;
    }
}
```

Code Snippet 4 - BusPassenger.cs (60..76)

The bus will continue driving on its route, picking up and dropping off passengers until it reaches the final stop. At this point, the bus will drop off all the passengers and will then remove itself from the scene, returning to the Bus Companies' Bus Pool.

Finally, this is the code for the Topological Sort that was used to sort the busses in the timetable UI.

```
/// <summary>
/// Sorts the <see cref="BusStop"/>s in the list of <see cref="BusRoute"/>s by using
Topological Sort.
/// Adapted from Wikipedia: https://en.wikipedia.org/wiki/Topological_sorting#Depth-
first_search
/// </summary>
/// <param name="busRoutes">List of <see cref="BusRoute"/>s to sort</param>
/// <returns>An ordered topological list of <see cref="BusStop"/>s from each <see
cref="BusRoute"/></returns>
public static List<BusStop> TopologicalSort(List<BusRoute> busRoutes)
{
    // Sorted bus stops to return at end of method
    List<BusStop> sortedBusStops = new List<BusStop>();

    // Dict of Bus Stop edges for graph
    Dictionary<BusStop, List<BusStop>> busStopEdges = new Dictionary<BusStop,
List<BusStop>>();

    // List of unvisited Bus Stops for the sort
    List<BusStop> unvisitedBusStops = new List<BusStop>();

    // Create Graph of Bus Stops
    foreach (BusRoute route in busRoutes)
    {
        // Create Vertex for each pair of Bus Stops
        for (int busStopIndex = 1; busStopIndex < route.BusStops.Count; busStopIndex++)
        {
            BusStop stop = route.BusStops[busStopIndex - 1];
            BusStop nextStop = route.BusStops[busStopIndex];

            // Add Vertex if it doesn't already exist
            if (!busStopEdges.ContainsKey(stop))
                busStopEdges[stop] = new List<BusStop>();

            // Add next stop to Vertex if it doesn't already exist
            if (!busStopEdges[stop].Contains(nextStop))
                busStopEdges[stop].Add(nextStop);

            // Add stops list of unvisited Bus Stops for Topological Sort
            if (!unvisitedBusStops.Contains(stop))
            {
                unvisitedBusStops.Add(stop);
            }

            if (!unvisitedBusStops.Contains(nextStop))
            {
                unvisitedBusStops.Add(nextStop);
            }
        }
    }

    // Perform Topological Sort
    List<BusStop> beingVisitedBusStops = new List<BusStop>();

    while (unvisitedBusStops.Count != 0)
    {
        BusStop selectedStop = unvisitedBusStops[0];

        Visit(selectedStop, unvisitedBusStops, busStopEdges, sortedBusStops,
beingVisitedBusStops);
    }

    sortedBusStops.Reverse();
    return sortedBusStops;
}
```

Code Snippet 5 - Topological Sort Code

2.4 Graphical User Interface (GUI) Layout

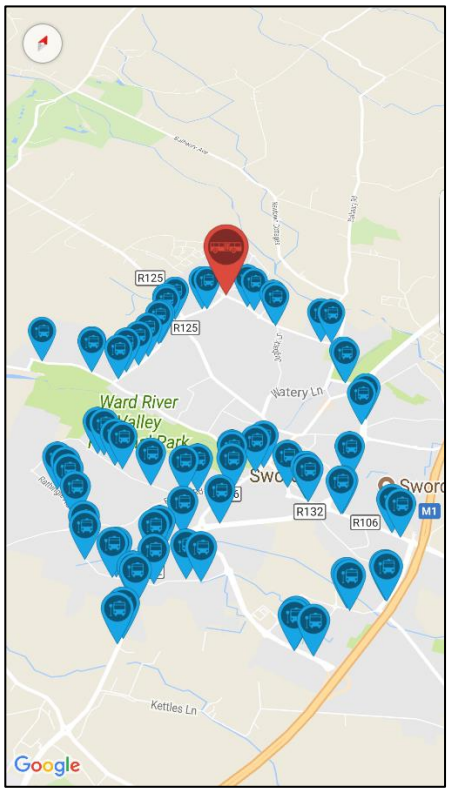


Figure 4 - Map on App Start

When the Android App *(Figure 4)* starts, the first view that the user will see will be the map. The blue marker icons represent bus stops. The red marker icon represents a bus. The bus marker will update in real time according to the update messages being sent from the Unity Simulation.

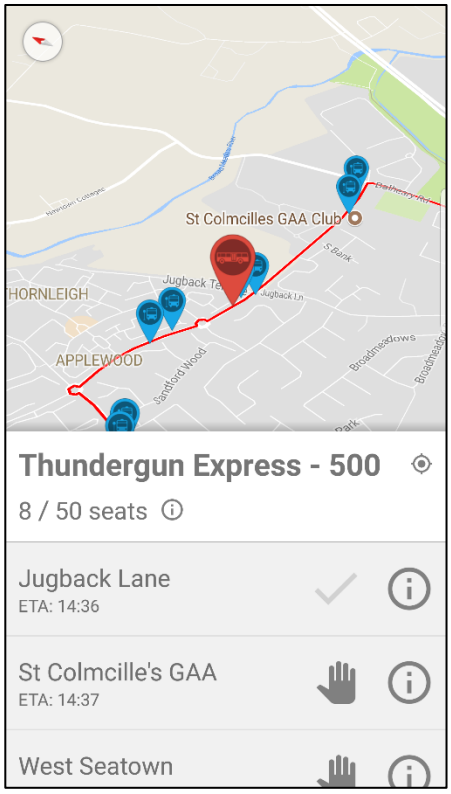


Figure 5 - Selected Bus with Stop List

When the user taps a bus marker, the camera will zoom into a marker position and the top of the Bottom Sheet will appear. The sheet will contain the name of the Bus and a live updating text view of the seat capacity in the bus. When the user taps the name on the Bottom Sheet (in this case, *Thundergun Express – 500*), the Bottom Sheet will expand, displaying a list such as the one in *(Figure 5)*.

From this list, the user can perform several different tasks:

- **Center the Camera on the Selected Bus** by tapping the “location” button on the top-right section of the sheet.
- **View a Bus Stop** by tapping the “info” button inside one of the list entries.
- **Hail a Bus at a Stop** by tapping the “hand” button inside one of the list entries. (A *Tick Icon* will appear if the bus is hailed for that stop)
- **View Extra Information** by tapping the “info” button beside the seat indicator.

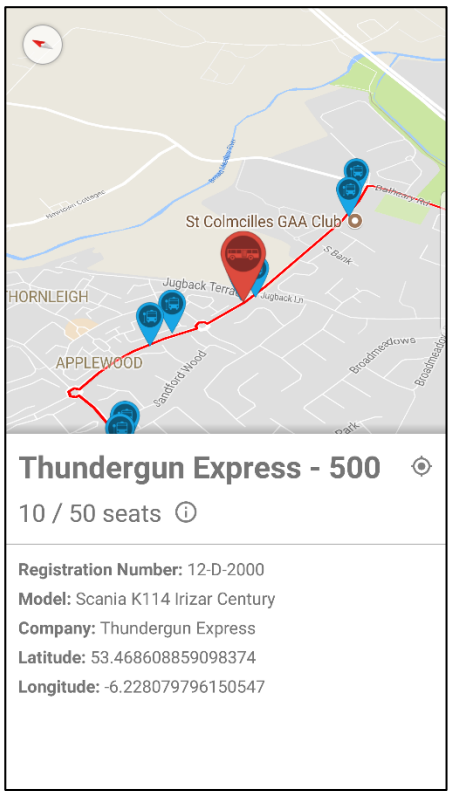


Figure 6 - Selected Bus with Extra Info

Once the user presses the “info” button beside the seats indicator, the list will be swapped out for the info panel seen in *(Figure 6)*. If the user presses the “info” button again, they will return to the list seen in *(Figure 5)*.

The extra info panel displays the following information:

- **Bus Registration Number**
- **Bus Model**
- **Bus Company**
- **Bus Position Latitude**
- **Bus Position Longitude**

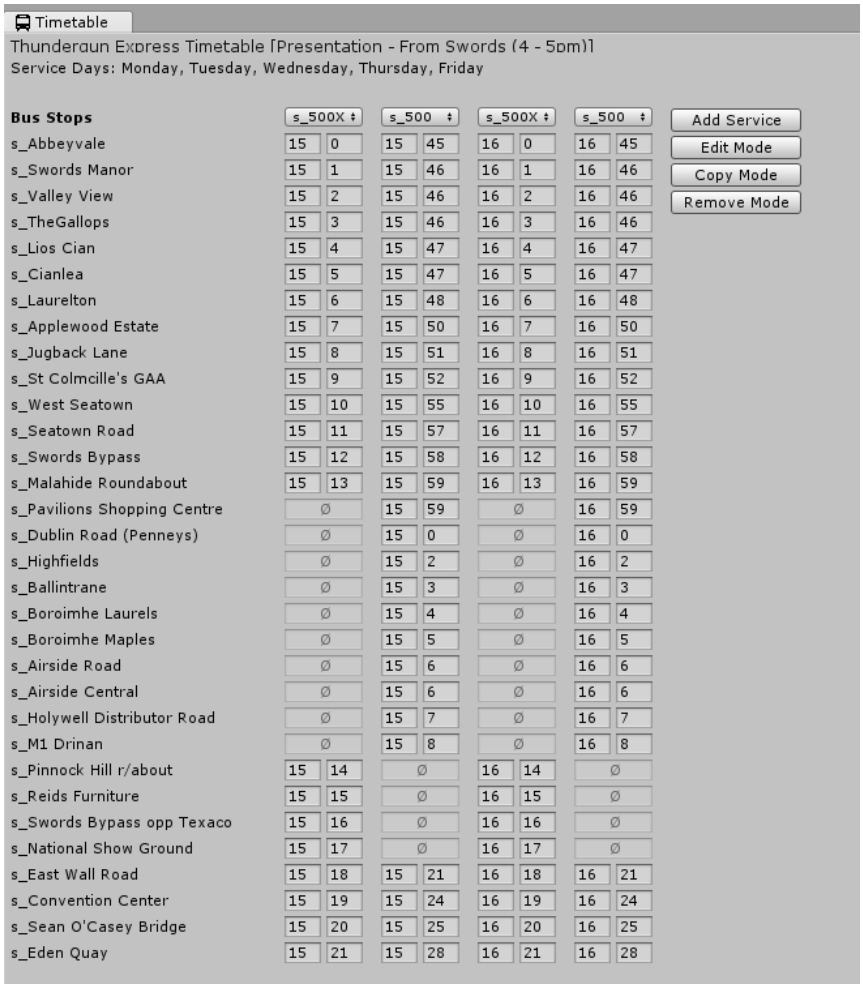


Figure 7 - Timetable with Controls

When the user selects a timetable in the Unity inspector, and opens a Timetable window, the timetable for that object will show up, naturally. This can be seen in *(Figure 7)*. The user can add new services by clicking the **Add Service** button. Services can be moved left and right by clicking **Edit Mode**. Clicking **Copy Mode** will copy a service to the end of the timetable. **Remove Mode** will remove selected services from the timetable. The user can **change Time Slot values by typing in the Hour and Minute** fields for each cell. The user can also **select the route for each service by clicking on the Dropdown Menu** at the top of each column. The timetable will auto-sort the bus stops by using a topological sort. If a bus stop row and a route column intersect where that bus stop does not belong to that route, the NULL symbol will be displayed instead, indicating that a Time Slot does not exist there.

2.5 Testing

I demonstrated an early version of this project on January 24th, 2018 at a game development meetup called “The Games Co-Op”. I got some good feedback regarding the future applications of the project, such as potential VR applications or expanding the project to new industries, such as rail and airlines. I also used JUnit testing on the Android application.

```
@Test
public void objectModel_isValid() {
    // Test to ensure that the object model is valid
    List<BusStop> busStopList = new ArrayList<>();
    busStopList.add(new BusStop("Yulin", "c_Yulin", 22.633333, 110.15));
    busStopList.add(new BusStop("Maoming West", "c_Maoming West", 21.65,
110.916667));
    BusRoute r = new BusRoute("100", "y_100", busStopList);
    List<TimeSlot> timeslots = new ArrayList<>();
    timeslots.add(new TimeSlot(busStopList.get(0), "0"));
    timeslots.add(new TimeSlot(busStopList.get(1), "5"));
    Bus b = new Bus("Guangxi Provincial Bus", 0, 0,
            "01-00-0000", "Guangdong Speedster",
            "China Transport Ltd", r, r.getBusStops().get(0), new
ArrayList<BusStop>(),
            timeslots, 20, 30, 0);
    assertEquals(b.getCurrentRoute(), r);
    assertEquals(b.getCurrentCapacity() <= b.getMaximumCapacity(), true);
    assertEquals(b.getCurrentStop(), r.getBusStops().get(0));
    assertEquals(b.getCurrentStop().getInternalId(), "c_Yulin");
}
```

Code Snippet 6 - Example Unit Test from Android (BusStopUnitTests)

3 Conclusions

I found that this was a very interesting project to work on. As I mentioned before, I enjoy working with the Unity Engine and I really wanted to work on a project that would allow me to improve my skills with Unity and to learn more about the engine. Simple for this reason I enjoyed working on the project.

I feel that the scope was far too big for what was possible to pull off with my allotted time over the last two semesters. I could have made the scope smaller but at the same time I was not aware of my total workload during Semester 8 when I determined the scope for this project. I feel losing the month of April to working on other projects seriously limited the potential of this project.

I was able to implement the MVP of the project and for that I am very proud. I put a lot of work into this project and despite what marks I get I can at least feel assured that I tried my best to make this project what it is today.

Even if I had an extra month, I wouldn't have hit my original target scope for the project. I was overambitious, but I learned a lot during the development of this project. I feel that my skills in Unity have improved significantly than they were 6 months ago and that's the core of what I wanted out of the project.

As for the scope of the project itself, I feel that even though it hit MVP a lot more could have been done with it. Since the conception of the idea there wasn't a clear end goal in sight for the project and this resulted in some features being implemented that weren't essential to the project succeeding at large.

Overall, I'm grateful for the opportunity to work on this project and for the experience gained during the development cycle.

4 Further development or research

4.1 Predictive Analysis for Time Slots

A type of predictive learning system would be a useful additional feature as it would allow users to see predictions of traffic patterns and would make planning their commute even more efficient and useful.

4.2 Full Traffic Simulation

With more time, a traffic simulation would be a good addition to the bus system simulation as it would provide a more realistic environment for the busses to traverse, generating more useful data for the predictive analysis.

4.3 Bus Alerts

A system to alert the user when to leave for their bus would be a very useful addition to the application.

4.4 Other Industries

The application could be extended to be used in other industries such as aviation and rail transport industries. There could be unexplored potential in using this application and it’s predictive analysis systems in these other industries.

5 References

Android Developers. (2018). Developer Guides | Android Developers. [online] Available at: https://developer.android.com/guide/ [Accessed 7 Mar. 2018].

Postgresql.org. (2018). PostgreSQL: Documentation. [online] Available at: https://www.postgresql.org/docs/ [Accessed 1 Feb. 2018].

Technologies, U. (2018). Unity - Manual: Unity User Manual (2018.1). [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/index.html [Accessed 9 Mar. 2018].

Unity Forum. (2018). Follow Orbit Camera. [online] Available at: https://forum.unity.com/threads/follow-orbit-camera.202490/ [Accessed 7 Jan. 2018].

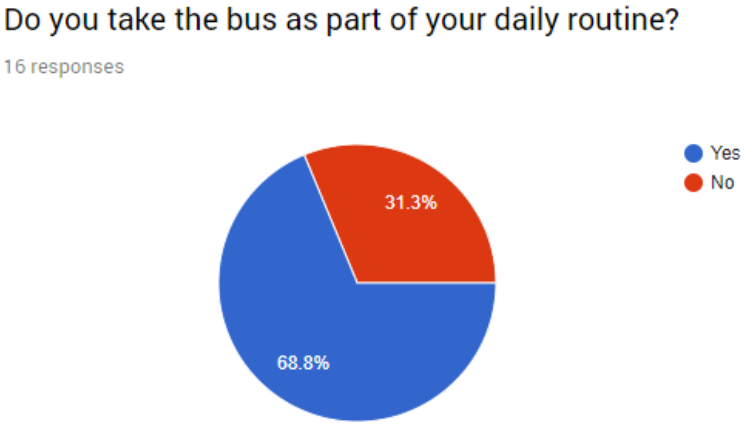
Unity Forum. (2018). Handles.Label fail when point behind camera. [online] Available at: https://forum.unity.com/threads/handles-label-fail-when-point-behind-camera.79217/ [Accessed 21 Mar. 2018].

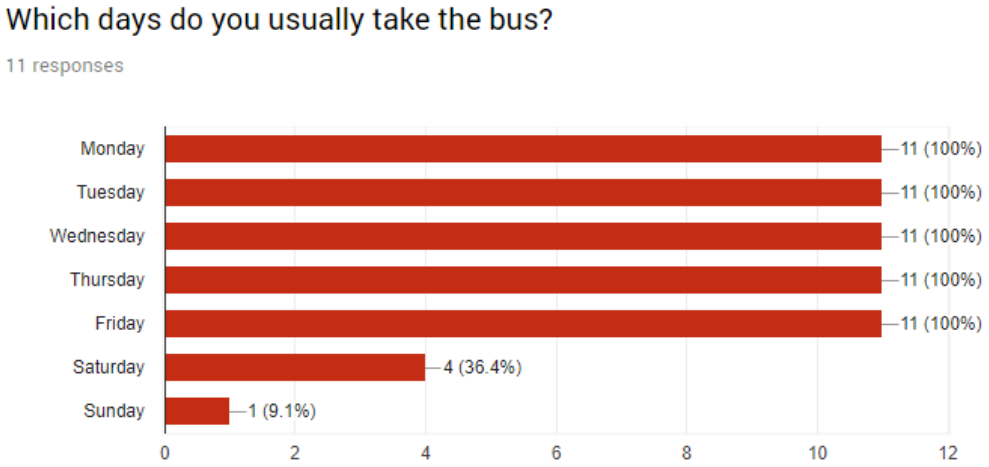
Unity Forum. (2018). Handles.Label fail when point behind camera. [online] Available at: https://forum.unity.com/threads/handles-label-fail-when-point-behind-camera.79217/ [Accessed 10 Feb. 2018].

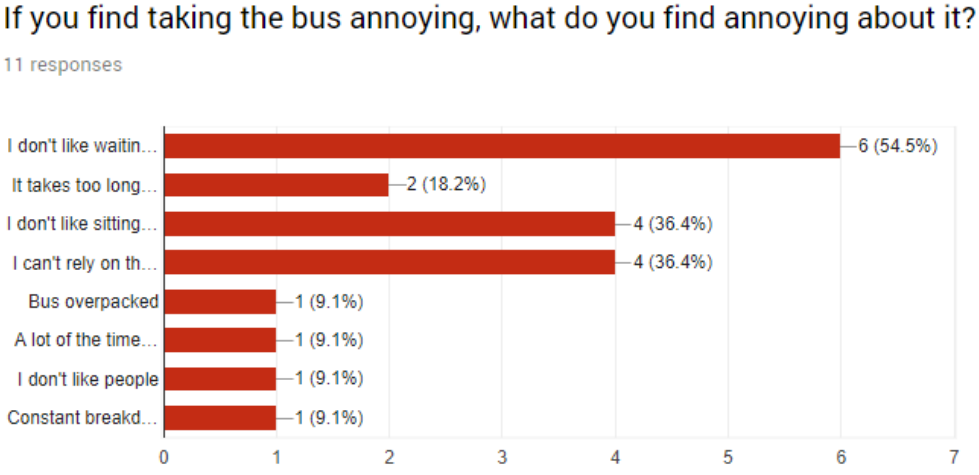
Unity, S. (2018). Serialize and Deserialize Json and Json Array in Unity. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/36239705/serialize-and-deserialize-json-and-json-array-in-unity [Accessed 13 Feb. 2018].

6 Appendix

6.1 Survey Results

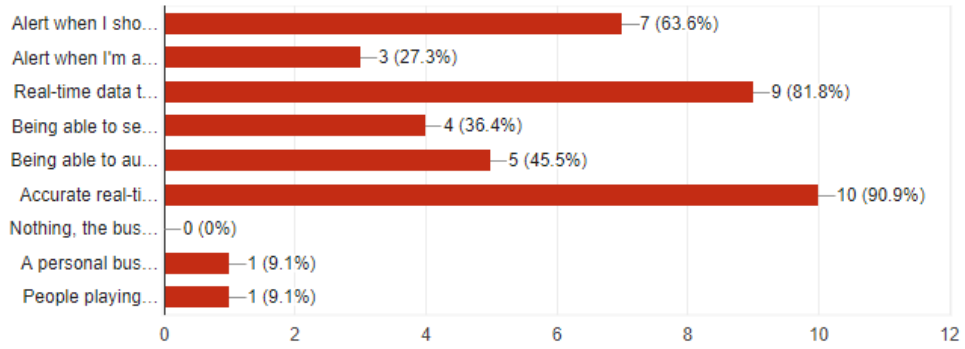






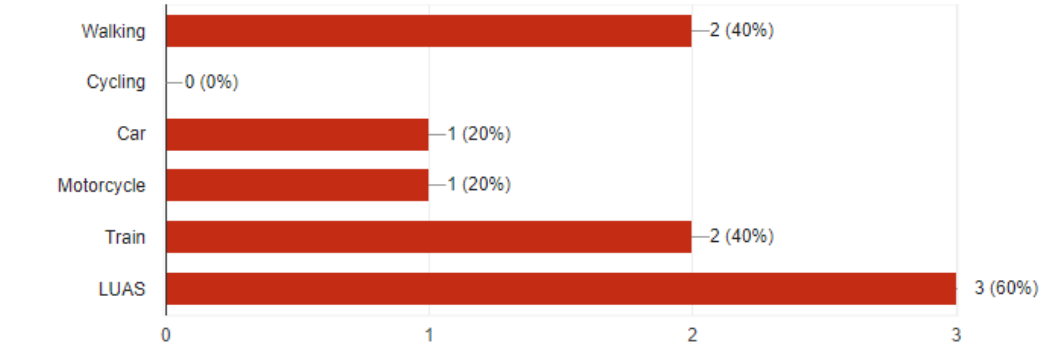
What would make taking the bus more enjoyable?

11 responses



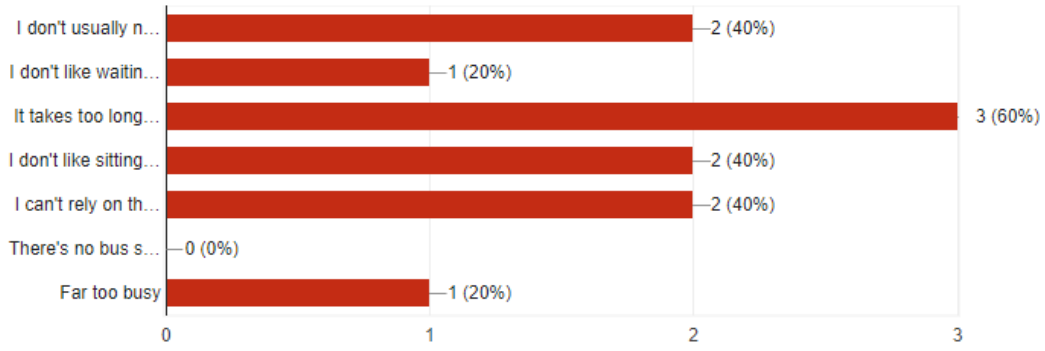
How do you usually get around?

5 responses



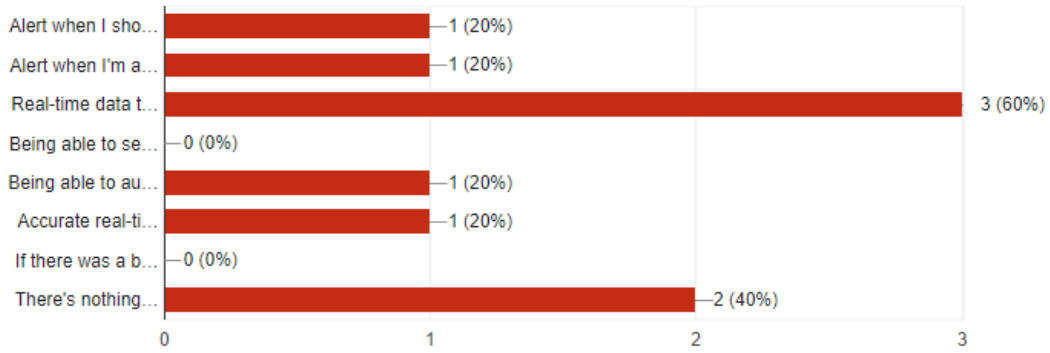
Why don't you usually take the bus?

5 responses



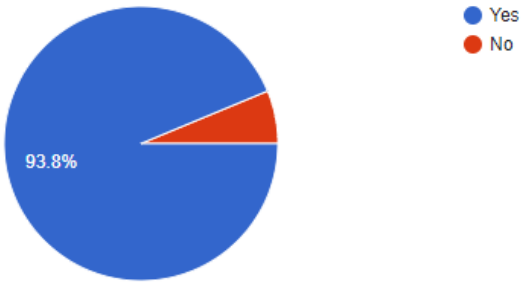
What would convince you to take the bus more often?

5 responses



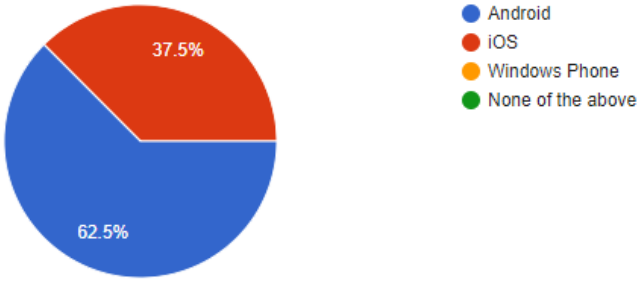
Would you use this app if it were available now?

16 responses



What Operating System is on your phone?

16 responses



6.2 Project Proposal

Project Proposal

Bus Stop!

Aaron Meaney, 14326016, aaronmeaney@hotmail.com

BSc (Hons) in Computing

Internet of Things

24th October 2017

1. Objectives

Primary Objective:

The purpose of this project is to build a proof-of-concept app that provides a more convenient public transport experience by ensuring passengers are provided detailed information on their bus journey. By doing this, the project should provide insight into how a ubiquitous public transport system might look in the future. If this project were to be released commercially, then it should aim to encourage more passengers to take the bus.

Build an app to provide passengers with real-time journey information:

The app should provide passengers with information on the bus company, the bus the passenger is taking, and their bus route. Estimated arrival times based on previous trends allow passengers to plan their day more effectively. Push notifications should inform passengers when they should leave for the bus stop to make it on time, and should also alert passengers when they're about to reach their stop; allowing them to sleep on the bus knowing they won't miss their stop. Notifications of delays and breakdowns will also be important information for the passenger. If the passenger has planned their journey and they're close enough to the bus stop, the app will automatically hail their bus by triggering the 'STOP' button remotely.

Build a web service to share bus companies' data with passengers:

The web service should provide an API for the passenger app to request data on the bus companies' services. For example; routes, busses in service, bus breakdown history, ETAs, etc. The service should also provide an API for busses to upload their sensor data. From a business point of view, the web service should be generic and contain a web page for administration purposes, making it easy to integrate the company's fleet into the service. The web service should also have a database to store sensor history, in which case trends can be analysed. Data such as breakdown history and arrival times can be used to determine breakdown rates and on-time percentages, which can be made known to passengers.

Build a bus network simulation to demonstrate the web service and app:

A simulation of a bus network will be built to send data to the web API as opposed to developing the app on a real bus. The simulation will handle passenger and bus routing behaviour with the goal to demonstrate how the app will be used in a real-world setting. The app will be able to send data back to the simulation to make it interactive, such as by hailing a specific bus.

2. Background

Idea Background

I take the bus to college every morning, and over the last 3 years several issues have started to annoy me. While public transport is supposed to be a cheap and convenient way to get around the city, I've found that it can be a very stressful and frustrating experience. In the mornings, I find that sometimes I'll skip breakfast since I'm worried that I'll miss my bus and be late for college. Sometimes, I'll get there early and I'll have just made it on time. Other times, I'll be waiting for 10 minutes for the bus to arrive. I never know when I should leave to catch the bus.

I know I'm not the only person who dislikes public transport, there are many issues with taking it. From waiting a long time for the bus, not knowing if there'll be enough seats on the bus, not knowing what time I'll arrive at my destination, not knowing which bus to take to get to my destination – it's a frustrating experience. From this I can understand why people drive instead. However, taking the bus is cheaper and is better for the environment than driving in a private car. This inspired me to build an app to make taking the bus more convenient by coming up with solutions to common public transportation problems, and to make a service that bus companies can easily integrate into.

Public Transport Problems and Solutions

Problem: People don't know when to leave for the bus to minimise waiting time.

To solve this problem, I planned on the passenger app sending a push notification to the passenger for when they should leave for the bus. This works by analysing data such as the traffic, bus position, arrival trends and how long it takes the passenger to reach the bus stop to determine when the passenger should leave their house. This minimises time wasted at the bus stop and gives the passenger peace of mind knowing that they won't be late for the bus once they leave.

Problem: The bus driver might not see the passenger trying to hail the bus.

If the passenger is in a crowd and multiple busses are driving past the stop at once, it can be difficult to hail the bus you want to get on. By using this app, the passenger can hail their bus by triggering the 'STOP' button on the bus. To prevent abuse, the mobile device must be GPS enabled and the user must be within a certain distance of the bus stop to hail the bus remotely. To prevent GPS spoofing, the app will check if the phone is rooted or is using the developer mode's mock GPS locations, and will disable the hailing feature in these cases.

Problem: The passenger is just around the corner and can't hail the bus on time.

If the passenger is running towards the bus and just can't make it around the corner on time to hail it, the app can hail the bus automatically if the passenger setup their journey on that bus beforehand. Once they're close enough to the bus stop and auto-hailing is enabled, the bus will be hailed and the passenger should have enough time to make it onto the bus.

Problem: Passengers want to sleep on the bus but don't want to miss their stop.

Many people have slept past their stop on the bus to solve this issue, this app will add a 'GPS alarm' function to wake people up once they get close to their stop. As the bus pulls away from the stop prior to their one, the phone will vibrate 3 times and a push notification will be sent, prompting the user to exit the bus. If they don't wake up and sleep past their stop, the vibrating will continue at a higher intensity.

Problem: The passenger wants to get on a bus with specific seats available.

If multiple busses are pulling over at once on the same route and the passenger wants to get on the bus with the least capacity, they can't know until they get on the bus. This app will solve that problem by not only showing the user the capacity of the bus, but by also showing them the actual seats available. On busses with a space for wheelchairs and buggies, a distance sensor can detect if it is free, notifying the passenger.

Conclusion

While these are just a few examples of problems and solutions that passengers encounter on the bus, there are many more that are outside the scope of this project. Each feature of this service should be designed to counter at least one of these problems.

Other features that could be added to this service are auto-routing, to detect which busses the passenger should take and the total price / duration of the journey will be. Bus passes, Tickets and Leap Cards could also be integrated into the app, so tapping onto the bus with the mobile device could be a possible feature.

The service also must be designed so that when a bus company subscribes to it, it takes a small amount of effort to integrate the company's bus fleet. The fact that the demo company is in a computer simulation should be abstracted away from the web service so that if it were to be integrated in real life, no extra coding work will have to be done to the web service or mobile app.

3. Technical Approach

Throughout the duration of this project I'll continue to update the documentation and project plan to account for changes to the project schedule and technical implementation. I'll use a system based on Agile Sprints to determine my weekly tasks and to keep track of the project's progress in relation to the feature list and deadlines.

Once I finish the requirements specification document, I can begin creating small prototypes to decide how I should build the project's tech stack. I've already put some thought into this and I've decided that I want to use the Unity game engine for the simulation since it's a very powerful and versatile tool that I happen to already have experience with. I've used node.js previously for developing an API and I would like to try using a different web framework since I dislike the lack of type-safety in Javascript which can result in buggy and unsafe code. Perhaps a framework based on C#, Kotlin or Java would be more suitable? I'll develop the Android app using Android Studio since it is the standard method of developing apps and I also have prior experience using the program and the relevant languages such as Java and XML. I currently don't know what other libraries or frameworks I'll need; however, I will investigate these once I begin developing the prototypes to ensure that the tech stack is ready to be used with the full application.

The first thing that I'll work on after finishing the requirements specification document is the prototype for the mid-term presentation. I want the prototype to focus on the most critical technical aspect of the project first. In this case I want to setup a basic simulation and to see the real-time results of the simulation on an app connected over a web service. For example, I want to simulate a GPS-enabled bus sending longitude/latitude data to an app embedded with Google Maps. This will prove that the foundation of the project is sound and that I can continue to build features on top of this application architecture.

Once the prototype is completed I can begin to implement the rest of the features across the whole tech stack. For example, if I want to implement monitoring for seat capacity, I can implement the code in the simulation first and then send it to the web service to ensure that I'm getting all the raw data that I need. I can then process the raw data on the server and prepare it to be sent to the app. Afterwards, I can implement the UI changes and functionality in the app and then begin pulling the processed data from the web service. By ensuring that each feature is implemented this way, I can split up each feature into different tasks which makes it easier to implement due to each part of the project having its own separation of concerns. This in turn makes it easier to integrate unit testing which will make it easier to debug the project in the long run.

4. Project Plan

Gantt Chart

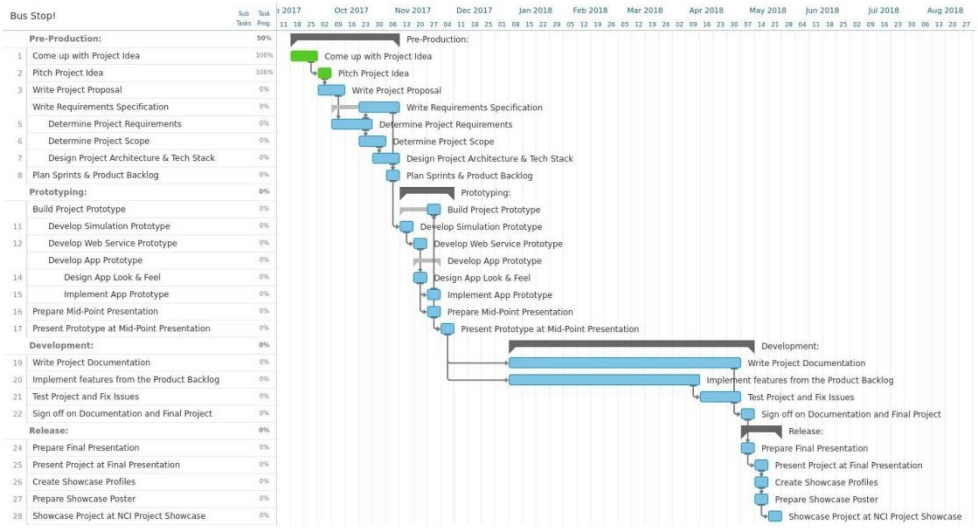


Figure 1. Snapshot of the Project Plan Gantt Chart, 10th of October 2017.

Description

As illustrated above, most of the work during 2017 is centred around requirements gathering, prototyping and project planning in preparation for the main development cycle during January and May 2018.

While there are very little details about the project during the main development phase, the project plan will be updated regularly to account for unforeseen changes during the project's development. For example, once the requirements specification is complete, it will then be possible to plan the sprints and product backlog to populate the development cycle with tasks to be completed and features to be implemented.

The time frame during the release cycle is also inaccurate as a date has not yet been chosen for the final presentations or project deadline. Therefore, the later stages of the project plan are naturally inaccurate.

5. Technical Details

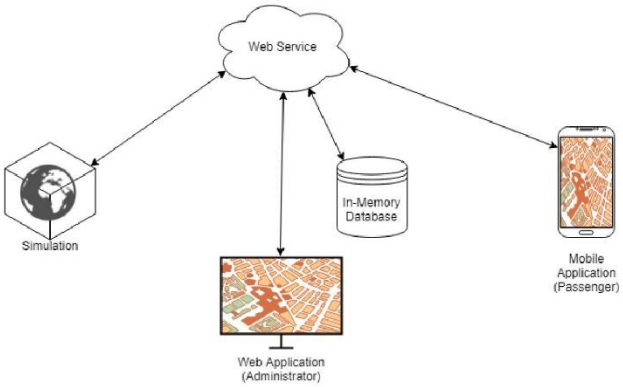


Figure 2. High-Level Diagram of the project architecture.

Simulation

As mentioned earlier, I plan on using the Unity game engine as the platform for developing my simulation. Unity scripts are written in C# and use the 'Entity-Component' programming paradigm which utilises the concept of composition over inheritance, allowing for faster development cycles. Once I have a basic simulation model built, I can connect to the web service using Unity's built in networking API. The simulation will run on a local machine such as a laptop.

Web Service

The web service could be implemented using a web framework such as Ruby on Rails or Django, however, as I mentioned before I would prefer to use a type-safe language to develop the web service. The current state of the bus network should be stored in primary memory for fast access. However, each update should be saved to the database so that if the server is restarted it can recover its previous operational state. The database will require very fast I/O speeds due to the potential high load and scalability requirement. I think an in-memory database such as Redis could be very useful, while maintaining the update log separately for persistence. I plan on hosting the web service on an online cloud-based-platform such as Amazon Web Services' Free Tier.

Mobile Application

As mentioned earlier, the mobile application will be programmed using Java and Android Studio. User settings such as what bus the user prefers to get will be stored on the local device, but live data will be retrieved from the web service.

6. Evaluation

Unit & System Testing

Throughout the whole development cycle of the application, I plan on using unit tests to ensure that any changes I make don't have unintended side effects within the system. I'd like to use Document-Driven Development and Test-Driven Development to ensure that the code I write for the application is designed to a professional standard and to prevent software brittleness; allowing me to develop new features without worrying about breaking implemented features.

I plan on using JUnit for unit testing on the Android application, and Espresso for GUI testing on the Android application. If my unit tests fail to catch an unintended side effect, the GUI testing should hopefully catch one of these changes. I also plan on load testing the web service since it will require high availability and scalability due to the potentially high user count. For Unity's unit-testing, I plan on using the Unity Test Runner, which is a variant of NUnit specifically designed for the Unity engine.

Continuous Integration

I'm also going to set up a Continuous Integration server to ensure that any tests that I forget to run are automatically executed when I commit my changes. Another benefit is that once a build has passed the CI process, it is available for download from the CI server, which will make distributing any project artifacts to testers much easier. I may use Travis CI for the Android application and the web service. Travis CI is an easy to setup service that integrates directly with my GitHub repositories. For the Unity simulation, I can use Unity's Cloud Build service which has similar functionality as Travis.

User Testing

I plan on testing my app with real users from the very beginning of the development cycle. Once I get feedback on my prototypes, I can update the UI design and can continue to get feedback by distributing new versions of the Android app by sending testers the link to the app's Travis CI download page. This will allow for rapid testing with users and should make the UI design process faster and more effective.

QA

Once the product backlog has been fully implemented or I reach the 3rd last week of the project plan, I'll commence a feature freeze and begin thoroughly fixing any remaining bugs. I will also continue testing the app throughout the final few weeks to catch any bugs I may have missed up until then.

6.3 Requirements Specification

BSCH

Requirements Specification (RS)

Bus Stop!

Aaron Meaney
22/11/2017

Requirements Specification (RS)

Table of Contents

Requirements Specification (RS) 1
1 Introduction 3
1.1 Purpose 3
1.2 Project Scope 3
1.2.1 Mobile App Scope 3
1.2.2 Web Service Scope 5
1.2.3 Simulation Scope 5
1.2.4 Constraints 7
1.3 Definitions, Acronyms, and Abbreviations 7
2 User Requirements Definition 8
2.1 Survey Results 8
2.2 Survey Analysis 11
3 Requirements Specification 12
3.1 Functional Requirements 12
3.1.1 Requirement 1: Navigate Map 12
3.1.2 Requirement 2: Select Map Item 14
3.1.3 Requirement 3: Create Alert 16
3.1.4 Requirement 4: Manage Alerts 18
3.1.5 Requirement 5: Hail Bus 20
3.1.6 Requirement 6: Bind Simulated Person 22
3.2 Non-Functional Requirements 25
3.2.1 Performance/Response time requirement 25
3.2.2 Security requirement 25
3.2.3 Reliability requirement 25
3.2.4 Extendibility requirement 25
3.2.5 Resource utilization requirement 25
4 Interface requirements 26
4.1 GUI 26
4.1.1 Map Screen 26

4.1.2 Options Menu 26
4.1.3 Bus Info Panel 27
4.1.4 Bus Stop Info Panel 27
4.1.5 Search Bar 28
4.1.6 Create Alert 28
4.1.7 Modify Alerts 29
4.1.8 Bind Simulated Person 29
4.2 Application Programming Interfaces (API) 30
5 System Architecture 31
5.1 Mobile App 31
5.2 Web Service 32
5.2.1 Get Map Data 32
5.2.2 Put Map Data 32
5.2.3 Get Bus Data 32
5.2.4 Put Bus Data 32
5.2.5 Get Bus Stop Data 32
5.2.6 Put Bus Stop Data 33
5.2.7 Get New Commands 33
5.2.8 Put Bind SP 33
5.2.9 Put Unbind SP 33
5.3 Simulation 33
6 System Evolution 34
7 Appendix 35
7.1 Survey Results 35

1 Introduction

1.1 Purpose

The purpose of this document is to determine the requirements for the development of the 'Bus Stop!' mobile application and its components: the 'Bus Stop!' web service to link the simulation to the mobile application, and the bus simulation to generate test data for the web service and the mobile application.

The target audience for this application are commuters who currently take the bus, to make their experience more convenient. Another target audience are commuters who primarily use private transportation. By solving some common problems with taking the bus, the mobile application should aim to convince private transport users that he bus is a cheaper, eco-friendlier alternative.

1.2 Project Scope

The scope of the project is to develop an Android mobile app, a backend web service and a simulation to provide a bus information feed to the user. This project will not be production ready on completion as a production-ready system would require specialised hardware. However, with the simulation component, the project is designed to be a proof-of-concept.

1.2.1 Mobile App Scope

The app will not be able to send commands to the web service on rooted or unlicensed phones, as GPS spoofing abuse will not be tolerated. A check will be executed on app start and periodically during the app's runtime to ensure that the phone has not been tampered with.

The mobile app should feature the ability for the user to navigate an interactive map of their location using Google Maps. From this interactive map, the user can select a bus or bus stop icon on the map. These stops and busses should be searchable, with results matching data such as a bus's route and next stop.

When selecting a bus stop, the user can see the name and timetable of that stop and can create an alert or select the corresponding bus for the time slot in the timetable.

When selecting a bus, the user can see the bus's live position, the bus's route, the bus's next stops and other miscellaneous information. From the next stops list, the user can select a time slot to create an alert or can select the bus stop. The user can also set the alert to repeat on certain days.

The app will also provide an 'all alerts' screen that can be accessed through the navigation bar, which will allow the user to edit, disable or delete the alerts that they've created so far. When an alert is created, the app will send a push notification to tell the user when to leave their home.

When the user has setup an alert for their bus and they are near the bus stop, the app can automatically hail the bus once it comes within range. Since this app is unaware if the user is on the bus or waiting for the bus, the user can also use this app to alert them for when the bus they are travelling on is about to reach its destination.

If the user has not setup an alert, but they are at the bus stop and the bus is about to arrive; the alert icon will change to a stop icon and the user can manually hail the bus.

When the user is waiting at a bus stop, the app will recognise this and will send a push notification to the user, allowing them to immediately select that bus stop and view its timetable.

The bus info panel will display miscellaneous information such as:

- Bus company name and website link
- Bus route name
- Bus route punctuality (If the bus is behind or ahead of schedule)
- Bus registration plate number
- Bus model
- Current seat capacity, grouped by seat types such as wheelchair access
- Bus GPS position co-ordinates
- Bus speed in km/h
- Number of total breakdowns
- Exceptional conditions, such as breakdowns

The time-slot info panel will also display miscellaneous information such as:

- ETA, as published by the bus company
- ETA, adjusted for the average lateness
- Average capacity
- Overall, Monthly, Weekly, Daily On-time percentage
- Warnings for exceptional conditions, such as high lateness or high breakdown rate
- Estimated carbon emissions reduction based on bus capacity as opposed to everyone driving in a private car. Used to convince people of the eco-friendliness of busses.

Since this project is a proof-of-concept, the user will not be able to interact with busses in real life to test out the app. To address this issue, the app will have a ‘simulation mode’ setting. This will bind a selected simulated person’s (SP) position in the simulation to the GPS position shown on the app. The SP will have to exist in the simulation and is selected by entering the SP’s identifier (SPID) in the app. This is analogous to a real person’s PPSN.

This is not a travel planning app and therefore an automatic route planner is out-of-scope.

1.2.2 Web Service Scope

The backend webservice should contain two different secure RESTful API endpoints providing distinct functionality. The first API endpoint is just the '/api/*' endpoint format and will be referred to as the 'Standard API'. The Standard API will be used for communication between the app and the simulation. The Standard API won't know that the sensor data is being collected from a simulation and should theoretically be able to collect real sensor data in a production system. The second endpoint will use the '/api/dev/*' endpoint format and will be referred to as the 'Developer API'. The Developer API will be used for special commands such as binding the SPs on the simulation. In a production system, the Developer API would be useless.

The Developer API will not have access to a persistent database, however the Standard API will. The Standard API will have data sent to and read from both the app and the simulation in real time. Because of the high I/O usage with the database, it is imperative that the database is ACID complaint to ensure accurate statistics are reported back to the user.

When data is sent to the Standard API, it goes through a 3-step process:

1. The raw data is written to the database (E.g. Bus's state)
2. The data is processed (E.g. Average On-Time %)
3. The processed data is written to the database

1.2.3 Simulation Scope

The simulation should provide a navigable 3D interactive space based on real-world OSM data. This will allow for the development of a simulation with terrain and road infrastructure that will match up with the Google Maps data.

Bus Stops and Bus Depots will be manually placed in their real-world locations. Every Bus, Bus Stop, Depot and SP in the simulation will determine their latitude and longitude based on their transform position within the simulation.

Each bus instance is persistent, containing data such as its registration plate and make/model. Since each bus is distinct, if there aren't enough busses to fill each route, then no busses can be deployed from the depot.

Each bus will have several seats and seat types. For example, a bus may have 20 standard seats and 1 wheelchair accessible seat. When a SP boards the bus, the current capacity is updated on the bus. If there are no more available seats of the SP's preferred type, then the SP will not be able to board the bus. The SP will wait at the bus stop for the next bus.

SPs will form a queue at a bus stop as it gets closer to the arrival time for their bus. The length of these queues is determined by the time of day and the day itself. For example, rush hour on a Monday will be busier than 3pm on a Sunday.

When a SP is instantiated in a queue, they are given a random SPID and a random designated stop belonging to the route they'll be travelling on. Once the SP is on the bus and is about to reach their designated stop, they will press the

'STOP' button and depart once the bus pulls over. Once they depart the bus, they will persist for a few seconds until their instance is destroyed.

Bus Routes will be represented in the simulation as an ordered list of nodes. These nodes will be defined as either navigation nodes or bus stops. Navigation nodes are used to tell the bus the route to take while the bus stops tell the bus where to stop along the route. The path to take between each node is determined using a pathfinding algorithm such as Dijkstra's or A*.

When a bus route is scheduled to begin, a bus will be deployed from the bus depot and will drive to the start of the route. It will wait at the bus stop and allow other SPs to board. Once the time slot passes and everyone has boarded, the bus will depart and drive along the route. If the next stop contains SPs forming a queue or if the 'STOP' button is activated, it will pull over to the next bus stop. Otherwise the bus will continue along the route until it stops at the final bus stop. Once the route is complete, the bus will start its next route if it's available. Otherwise, it will return to the depot.

If the simulation receives a message from the API to bind a SP, that SP will be set to 'simulation mode'. When a SP is in 'simulation mode', their instance will not be destroyed once they leave the bus. They will also send their GPS position data to the web service to be used in the app's 'simulation mode'. The SP will also lose the ability to hail the bus automatically as it's now dependent on input from the app. If an 'unbind' message is received, then simulation mode will be disabled and the SP instance will be destroyed if it has finished its journey.

To make the simulation more accurate, a basic traffic simulation will have to be implemented. Only vehicles within a small distance around each bus will be simulated to ensure that the computer isn't put under too much strain. Each car will be an autonomous agent that will try to drive to a random location in the simulation once it's created and will follow the rules of the road. Traffic lights will be included at intersections.

Busses, Depots, Stops and SPs will all display their 'Info Panel' when clicked in the simulation. The info panel will display configurable information for each of these objects. For example, the SPID, GPS co-ordinates and state of a SP and the company, route, speed, GPS position, route, etc. of a bus. The panel will also allow certain actions to be taken on that object. Such as forcing a bus breakdown or stopping queues forming on bus stops.

Each bus will contain a 'sensor hub' script that is responsible for recording the capacity of the bus, its speed, its position, route, next stop, registration plate number, model and if the bus is broken down or in service. Every few seconds, the sensor hub will send this information to the web service which will update the app view.

Certain features that will be out-of-scope include weather effects, a visible day/night cycle, random breakdowns and vehicle crashes.

1.2.4 Constraints

Before development of the project, a CI solution must be setup using Travis CI and Unity Cloud Build, and they should be linked to the project git repositories. The latest build of the app should be available on Travis CI for download. Unit tests should be executed on commit using the CI solutions.

The mobile app should be developed for the Android platform of at least API level 21, as it introduces lock-screen notifications to Android. While it only has 71.3% Android market penetration (as of November 2017), it is assumed that by the time the app would be production-ready, the market penetration would increase as people with older phones would begin to upgrade.

The web service should be built in a framework that supports RESTful APIs. After the development of the prototype, the framework should be determined.

The simulation should be built using the Unity game engine version 2017.3, as it will be the latest stable version once development of this project begins in early 2018.

Since this is a proof-of-concept, only two small bus companies will be simulated. Real-world bus data will not be downloaded and assets such as the depot, stops, etc. will be placed manually within the simulation.

The prototype should be completed by Monday, 4th December 2017. The project's features should be fully implemented by mid-April 2018. Testing should be complete and the final build distributed through CI before mid-May 2018.

1.3 Definitions, Acronyms, and Abbreviations

3D	Three-Dimensional
API	Application Programming Interface
CI	Continuous Integration
ETA	Estimated Time of Arrival
GPS	Global Positioning System
OSM	Open Street Map
SP	Simulated Person
SPID	Simulated Person Identifier

2 User Requirements Definition

2.1 Survey Results

To understand the user requirements, a survey was sent out on 17th November 2017. Screenshots are available in the appendix. 16 people responded. The results are as follows:

Section A: Initial Question

Question 1: Do you take the bus as part of your daily routine?

11 recipients said 'Yes'
(68.8%)

5 recipients said 'No'
(31.3%)

Section B: Regular Bus Passenger (Recipients who said 'Yes')

Question 2: Which days do you usually take the bus?

11 recipients take the bus on Monday
(100%)

11 recipients take the bus on Tuesday
(100%)

11 recipients take the bus on Wednesday
(100%)

11 recipients take the bus on Thursday
(100%)

11 recipients take the bus on Friday
(100%)

4 recipients take the bus on Saturday
(36.4%)

1 recipient takes the bus on Sunday
(9.1%)

Question 3: If you find taking the bus annoying, what do you find annoying about it?

6 recipients said, 'I don't like waiting at the bus stop.'
(54.5%)

2 recipients said, 'It takes too long for my bus to reach its destination.'
(18.2%)

4 recipients said, 'I don't like sitting beside strangers on the bus.'
(36.4%)

4 recipients said, 'I can't rely on the bus, it's usually late or never shows up.'
(36.4%)

4 recipients also had other responses regarding constant breakdowns and busses being too full and not being able to pick up any more passengers.

Question 4: What would make taking the bus more enjoyable?

7 recipients said, 'Alert when I should leave for the bus stop so I don't miss it.'
(63.6%)

3 recipients said, 'Alert when I'm about to reach my stop, so I can sleep on the bus.'
(27.3%)

9 recipients said, 'Real time data to see how full the bus is before it arrives.'
(81.8%)

4 recipients said, 'Being able to see if specific seats are available – such as wheelchair access.'
(36.4%)

5 recipients said, 'Being able to automatically hail the bus when I'm near the stop.'
(45.5%)

10 recipients said, 'Accurate real-time statistics on lateness and bus reliability.'
(90.9%)

Section C: Occasional Bus Passenger (Recipients who said 'No')

Question 2: How do you usually get around?

2 recipients said, 'Walking.'
(40%)

1 recipient said, 'Car.'
(20%)

1 recipients said, 'Motorcycle.'
(20%)

2 recipients said, 'Train.'
(40%)

3 recipients said, 'LUAS.'
(60%)

Question 3: Why don't you usually take the bus?

2 recipients said, 'I don't usually need to take the bus.'
(40%)

1 recipient said, 'I don't like waiting at the bus stop.'
(20%)

3 recipients said, 'It takes too long for my bus to reach its destination.'
(60%)

2 recipients said, 'I don't like sitting beside strangers on the bus.'
(40%)

2 recipients said, 'I can't rely on the bus, it's usually late or never shows up.'
(40%)

Question 4: What would convince you to take the bus more often?

1 recipient said, 'Alert when I should leave for the bus stop so I don't miss it.'
(20%)

1 recipients said, 'Alert when I'm about to reach my stop, so I can sleep on the bus.'
(20%)

3 recipients said, 'Real-time data to see how full the bus is before it arrives.'
(60%)

1 recipient said, 'Being able to automatically hail the bus when I'm near the stop.'
(20%)

1 recipients said, 'Accurate real-time statistics on lateness and bus reliability.'
(20%)

2 recipients said, 'There's nothing that would convince me to take the bus.'
(40%)

Section D: General Questions

Question 5: Would you use this app if it were available now:

15 recipients said, 'Yes.'
(93.8%)

1 recipient said, 'No.'
(6.3%)

Question 6: What Operating System is on your phone?

10 recipients said, 'Android.'
(62.5%)

6 recipients said, 'iOS.'
(37.5%)

2.2 Survey Analysis

The list of features from most popular to least popular, as voted by people who regularly take the bus:

1- Accurate real-time statistics on lateness and bus reliability
2- Real-time data to see how full the bus is before it arrives
3- Alert when I should leave for the bus stop so I don't miss it
4- Being able to automatically hail the bus when I'm near the stop
5- Being able to see if specific seats are available – such as wheelchairs
6- Alert when I'm about to reach my stop, so I can sleep on the bus

For people who don't usually take the bus, they were most interested in this feature:

- Real-time data to see how full the bus is before it arrives

People who don't usually take the bus were equally interested in the following features:

- Alert when I should leave for the bus so I don't miss it
- Alert when I'm about to reach my stop, so I can sleep on the bus
- Being able to automatically hail the bus when I'm near the stop
- Accurate real-time statistics on lateness and bus reliability

People who don't usually take the bus were disinterested in the following feature:

- Being able to see if specific seats are available – such as wheelchairs

From this survey, this is a sorted list of the most important features to consider for both regular bus users and occasional bus users:

1- Real-time data to see how full the bus is before it arrives
2- Accurate real-time statistics on lateness and bus reliability
3- Alert when I should leave for the bus stop so I don't miss it
4- Being able to automatically hail the bus when I'm near the stop
5- Alert when I'm about to reach my stop, so I can sleep on the bus
6- Being able to see if specific seats area available – such as wheelchairs

This sorted list will affect the priority of feature development during the main development cycle of the project.

3 Requirements Specification

3.1 Functional Requirements

Functional requirements refer to the basic functionality that the project should meet, disregarding non-implementation specific factors such as performance or security.

3.1.1 Requirement 1: Navigate Map

3.1.1.1 Description & Priority

Priority: High

The user should be able to navigate a map of the world on their screen by using standard Google Maps interface. The map should display a search bar, real-time bus icons and bus stop icons. When the app loads, the app should initialise the map at the user's location to make it more convenient to search for nearby bus stops. Users can search for busses and bus stops using the search bar. The app should update the selectable icons on the map every few seconds.

3.1.1.2 Use Case

Scope

The scope of this use case is to allow the user to navigate the world map and to search for bus stops and busses in real-time.

Description

This use case describes the process of downloading real time bus data from the Standard API and displaying it on the map. It also describes the search functionality.

Use Case Diagram

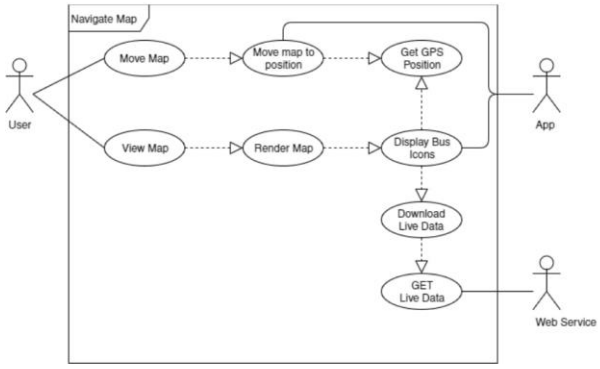


Figure 1 Navigate Map Use Case

Flow Description

Precondition

The app must have an internet connection and location services must be enabled.

Activation

The use case begins when the user opens the map.

Main flow

1. The app detects the user's current geographical position.
2. The app map zooms into the user's current geographical position.
3. The app queries the Standard API and downloads the latest bus state data. If the Standard API cannot be reached, see A1.
4. The app parses the download data and translates that to icons on the map.
5. The app renders the icons on the map and begins listening for the user's taps on the icons.
6. The app loops back to Main Flow step 2 to continue gathering live data.

Alternate flow

A1: Standard API could not be reached.

1. The app attempts to connect to the Standard API 3 more times.
2. If the app manages to connect, return to Main Flow.
3. Else see E1.

Exceptional flow

E1: Standard API connection failed.

1. The app displays an error to the user notifying them of the network error. The app also displays a 'retry' button.
2. The app waits until the user presses the 'retry' button.
3. When the user presses the 'retry' button, see A1.

Termination

This use case terminates when the app is closed or the user navigates to the alerts or settings screen.

Post condition

The map has been populated with information.

3.1.2 Requirement 2: Select Map Item

3.1.2.1 Description & Priority

Priority: High

While navigating the map, the user should be able to tap on the icons that have been pulled from the server. Once these icons are pressed, the map should center on that icon and should display a floating information panel with relevant information on that object.

For objects that contain a list (such as busses containing a list of their stops), action buttons should be available on the list entries. For example, timetable entries should provide a 'Create Alert' button and a 'Select Bus' button.

3.1.2.2 Use Case

Scope

The scope of this use case is to allow the user to select and deselect map items.

Description

This use case describes the process that occurs when a map item is selected and when a map item is deselected.

Use Case Diagram

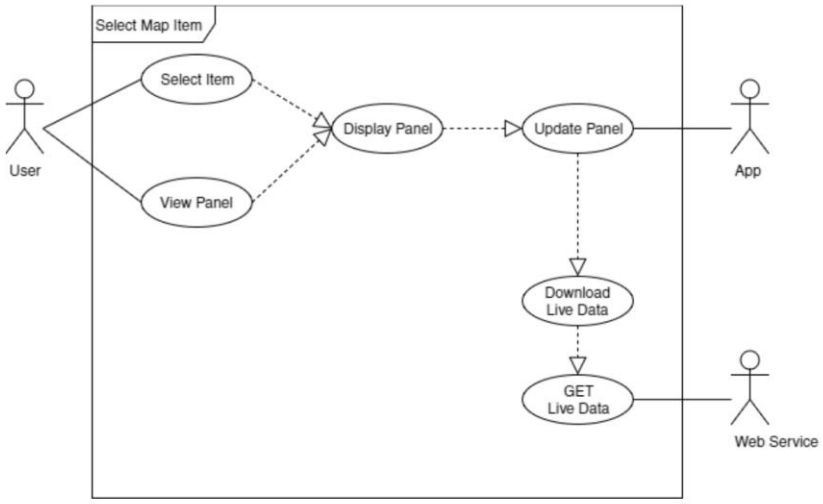


Figure 2 Select Map Item Use Case

Flow Description

Precondition

The app must have an internet connection and location services must be enabled. The map has been populated with icons.

Activation

The use case begins when the user taps a bus or depot icon on the map.

Main flow

1. The app displays a floating panel with placeholder information.
2. The app queries the Standard API for information on that specific object. If the Standard API cannot be reached, see A1.
3. The app downloads the information data.
4. The app replaces the displayed data with the new data.
5. The app loops back to Main Flow step 2 to continue gathering live data.

Alternate flow

A1: Standard API could not be reached.

1. The app attempts to connect to the Standard API 3 more times.
2. If the app manages to connect, return to Main Flow.
3. Else see E1.

Exceptional flow

E1: Standard API connection failed.

1. The app displays an error to the user notifying them of the network error. The app also displays a 'retry' button.
2. The app waits until the user presses the 'retry' button.
3. When the user presses the 'retry' button, see A1.

Termination

This use case terminates when the user closes the information panel.

Post condition

The app returns to the navigation map.

3.1.3 Requirement 3: Create Alert

3.1.3.1 Description & Priority

Priority: Medium

When the user presses the 'Create Alert' button in the information panel, another panel with options will be displayed to the user allowing them to configure and create the alert. Alerts are push notifications sent to the user for when they should leave their home to reach the bus on time.

3.1.3.2 Use Case

Scope

The scope of this use case is to allow the user to create alerts for when they should leave for the bus.

Description

This use case describes the process that occurs during the configuration and creating new alerts.

Use Case Diagram

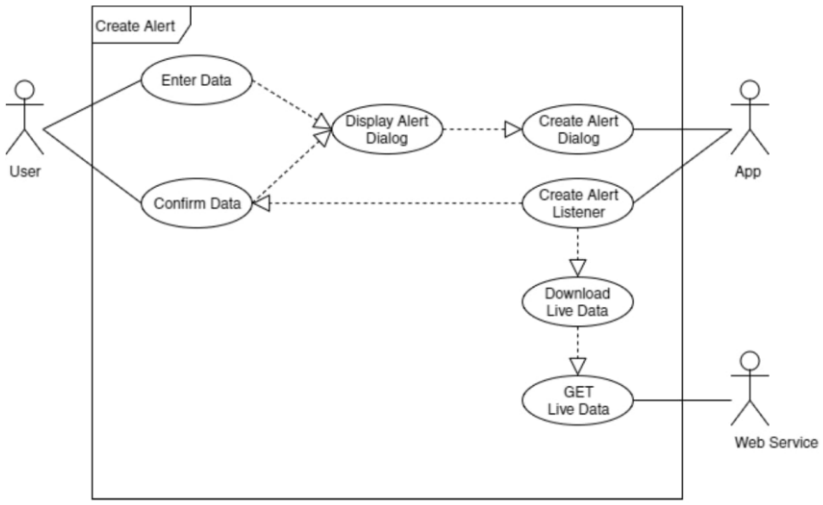


Figure 3 Create Alert Use Case

Flow Description

Precondition

The app must have an internet connection and location services must be enabled. The user is viewing a bus stop information panel.

Activation

The use case begins when the user taps the 'Create Alert' button under a time slot entry.

Main flow

1. The app displays a dialog requesting the user to insert configuration data for the alert.
2. The user inserts configuration data for the alert.
3. The user taps 'Confirm'.
4. The app processes the configuration data and creates the alert. If there is an issue with the configuration, see A1.
5. The app creates an alert listener to send a push notification once the bus's ETA reaches a certain value.

Alternate flow

A1: Misconfigured Alert

1. Determine which elements of the alert are misconfigured.
2. Highlight the misconfigured fields and display a tooltip to help the user understand how to fix this error.
3. Return to Main Flow step 2.

Termination

The use case ends when the user closes the dialog or successfully creates the alert.

Post condition

The app is now prepared to send the user a push notification once the alert is triggered.

3.1.4 Requirement 4: Manage Alerts

3.1.4.1 Description & Priority

Priority: Medium

Multiple alerts can exist at the same time, and the user may want to edit or delete these alerts. An alert menu will exist in the app, allowing the user to see all the created alerts and will allow them to edit and delete these alerts.

3.1.4.2 Use Case

Scope

The scope of this use case is to allow the user to view all the alerts, and edit/delete specific alerts.

Description

This use case describes the process that occurs during the display, modification and deletion of alerts.

Use Case Diagram

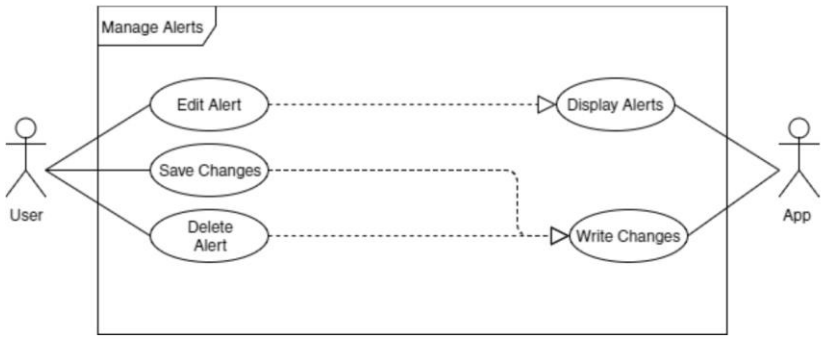


Figure 4 Manage Alerts

Flow Description

Precondition

The app must have an internet connection and location services must be enabled. The user is on the navigation map screen.

Activation

The use case begins when the user opens the 'Alerts' screen.

Main flow

1. The app displays any alerts that have already been created.
2. The user taps an alert to open its configuration.
3. The app displays the configuration options.
4. The user changes the configuration.
5. The user taps 'save' to save the alert configuration. If there is an issue with the configuration, see A1. If the user taps 'delete', see A2.
6. The app saves the changes made to the alerts.
7. The app closes the configuration panel.
8. Go to step 2 as the alerts screen will be refreshed.

Alternate flow

A1: Misconfigured Alert

1. Determine which elements of the alert are misconfigured.
2. Highlight the misconfigured fields and display a tooltip to help the user understand how to fix this error.
3. Return to Main Flow step 4.

A2: Delete Alert

1. The app deletes the alert data.
2. The app won't send any push notifications related to the related bus.
3. Return to Main Flow step 8

Termination

The use case terminates when the user closes the app or navigates away from the alerts screen.

Post condition

The user's changes to the alerts will be saved.

3.1.5 Requirement 5: Hail Bus

3.1.5.1 Description & Priority

Priority: Medium

When the user sets up an alert or presses the 'Hail Bus' button, the app will send a signal to the web server, which will send a signal to the bus, triggering the 'Stop' buzzer to activate. While this is only going to occur in a simulation, the real-life hardware implementation is not within the scope of this project.

3.1.5.2 Use Case

Scope

The scope of this use case is to send a stop signal to the bus, regardless of how it was triggered.

Description

This use case describes the process of sending the stop signal to the bus.

Use Case Diagram

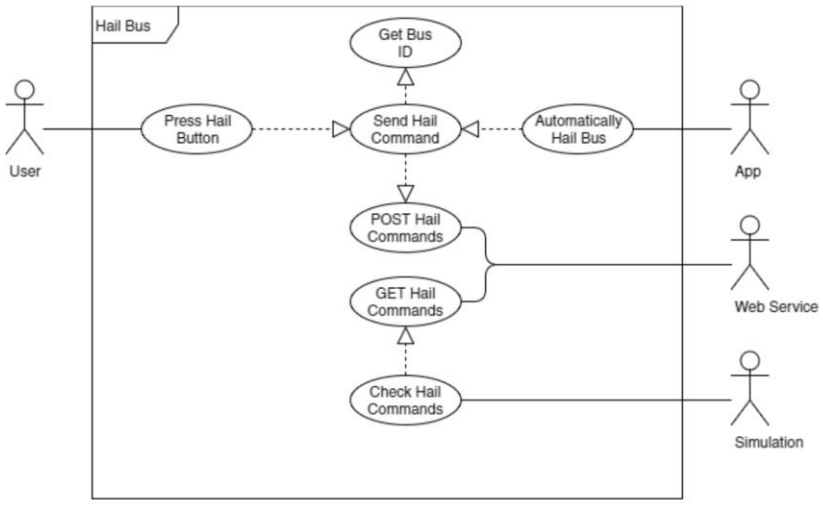


Figure 5 Hail Bus Use Case

Flow Description

Precondition

The app must have an internet connection and location services must be enabled.

Activation

The use case starts when the user or app invokes the 'Hail Bus' function.

Main flow

1. The app gathers the bus's unique ID.
2. The app sends a hail bus command to the web service along with the bus ID. If the Standard API cannot be reached, see A1.
3. The app will receive a response from the web server when the bus is successfully hailed.

Alternate flow

A1: Standard API could not be reached.

1. The app attempts to connect to the Standard API 3 more times.
2. If the app manages to connect, return to Main Flow.
3. Else see E1.

Exceptional flow

E1: Standard API connection failed.

1. The app displays an error to the user notifying them of the network error. The app also displays a 'retry' button.
2. The app waits until the user presses the 'retry' button.
3. When the user presses the 'retry' button, see A1.

Termination

The use case is terminated at the end of the main flow.

Post condition

The bus's 'STOP' buzzer/light will have been triggered remotely.

3.1.6 Requirement 6: Bind Simulated Person

3.1.6.1 Description & Priority

Priority: Low

The app contains a developer screen that allows the user to 'bind' the GPS coordinates of the app to the translated co-ordinates of a SP in the simulation. This allows the user to test features of the app within the simulation.

3.1.6.2 Use Case

Scope

The scope of this use case is to allow the user to bind the app's GPS coordinates with that of a SP, and to unbind the app from the SP.

Description

This use case describes the process that occurs during the binding and unbinding process.

Use Case Diagram

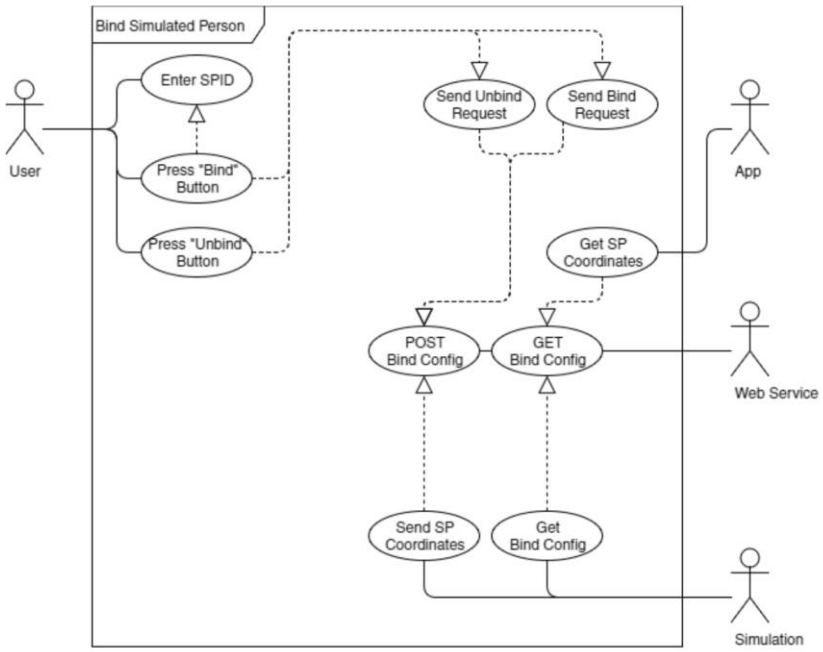


Figure 6 Bind Simulated Person Use Case

Flow Description

Precondition

The app must have an internet connection and location services must be enabled.

Activation

The use case begins when the user navigates to the developer screen.

Main flow

1. The user gets the SPID by looking at the simulation.
2. The user fills out the SPID field on the form on the app.
3. The user taps the 'BIND' button.
4. The app sends a bind request to the Developer API. If the Developer API cannot be reached, see A1.
5. The simulation pulls the bind request from the web service. If the Developer API cannot be reached, see E2.
6. The simulation begins sending that SPs co-ordinates to the web service. (E2)
7. The app begins pulling the SP co-ordinates from the web service. (A1)
8. The app replaces the GPS co-ordinates with the SP co-ordinates.
9. When the user is finished using the binding feature, they tap the 'UNBIND' button.
10. The app sends an unbind request to the Developer API. (A1)
11. The simulation pulls the unbind request from the web service. (E2)
12. The simulation stops sending that SPs co-ordinates to the web service.
13. The app stops pulling the SP co-ordinates from the web service.

Alternate flow

A1: App could not reach the Developer API.

1. The app attempts to connect to the Developer API 3 more times.
2. If the app manages to connect, return to Main Flow.
3. Else see E1.

Exceptional flow

E1: App failed to connect to the Developer API.

1. The app displays an error to the user notifying them of the network error. The app also displays a 'retry' button.
2. The app waits until the user presses the 'retry' button.
3. When the user presses the 'retry' button, see A1.

E2: Simulation failed to connect to the Developer API.

1. The simulation continues to retry to connect every 5 seconds.

2. Once connection succeeds, the simulation returns to the Main Flow.

Termination

The use case is terminated when the app is closed or the user unbinds the SP.

Post condition

The app goes back to using standard GPS co-ordinates.

3.2 Non-Functional Requirements

Non-Functional Requirements refer to general behaviour that the project must adhere to, disregarding the implementation of the functional requirements. For example, the response time and security requirements.

3.2.1 Performance/Response time requirement

It is important for this application to have a quick response time, as it is a real-time application working over a three-tier architecture. The Unity simulation can afford to have a low framerate of at least 15 frames per second. However, the actual simulation of the vehicles must still run in real time. To accomplish this, the simulation must be framerate independent and be deterministic.

3.2.2 Security requirement

The web service API endpoints must only be accessible using an authentication method to prevent tampering or accessing restricted data. Since this is a proof-of-concept, the API does not require SSL or HTTPS encryption.

To prevent GPS spoofing, the user's mobile phone will be checked for tampering on startup and every minute from then. The Android 'SafetyNet' library is a package designed by Google to check if the Android device for rooted or unlicensed. The 'SafetyNet' library will be used to prevent GPS spoofing.

3.2.3 Reliability requirement

The project should be built to the standard so that simply using it how it was intended to should not result in any errors. Rigorous testing throughout the development cycle should ensure that the final project has no critical bugs in it that make the app unreliable.

3.2.4 Extendibility requirement

The app and web service must be designed so that a production ready build would be easy to implement. The simulation tier does not need to be extendible. The Standard API should be ignorant to the fact it is getting data from a simulation, since it would be getting data from a hardware device on a bus. The app will need minor tweaking for a production-ready build, such as removing the developer mode.

3.2.5 Resource utilization requirement

The Unity simulation will have to run at an acceptable framerate medium-end gaming laptop with an Nvidia GTX660m Graphics Card. Because of this, simulation factors outside of the main scope should not be processed or should be processed only when necessary. For example, instead of simulating an entire city with traffic, the traffic simulation will only occur around the areas where the busses already exist.

4 Interface requirements

4.1 GUI

4.1.1 Map Screen

When the user opens the app for the first time, they will encounter the main map screen. This screen contains a search bar, options menu and the icons the user interacts with to use the app's main functionality.

In this example, the person icon represents the user's position, the alert icon represents the bus stop position and the star represents the bus's position.

Figure 7 Map Screen GUI

4.1.2 Options Menu

To navigate the app, the user will have to tap the hamburger menu on the top-right of the app screen.

This will open a dropdown menu containing the links to the Alerts screen and the Bind SP screen.

When the user taps one of these links, the app will transition to that screen.

This hamburger menu will persist throughout the entire app and its contents will change depending on what the current active screen is.

Figure 8 Options Menu GUI

4.1.3 Bus Info Panel

The bus info panel is displayed when the user taps on a bus icon on the map.

Essential information on that bus is displayed on the top of the panel, along with an info button that will replace the panel content with miscellaneous information.

The lower panel is the bus's current route timetable. These timeslots display the bus stop name, ETA (including lateness), a 'Hail' / 'Alert' button and a 'Select Bus Stop' button.

Figure 9 Bus Info Panel GUI

4.1.4 Bus Stop Info Panel

The bus stop info panel is displayed when the user taps on a bus stop icon on the map.

The bus stops' name and timetable is visible on this panel. An info button is placed beside the bus stop name. When pressed, extra information will be displayed about the bus stop in place of the timetable.

Like the bus info panel, the timetable displays the bus name, ETA (including lateness), a hail, alert and 'Select Bus' button.

Figure 10 Bus Stop Info Panel GUI

4.1.5 Search Bar

The search bar will begin to populate results from the map as the user types, using a fuzzy search algorithm.

The search results will contain the name of the result, the distance to that result from the user and an icon representing the result type.

When the user taps on a result, the search box will be cleared and the map will select that item, following it and displaying its information pane.

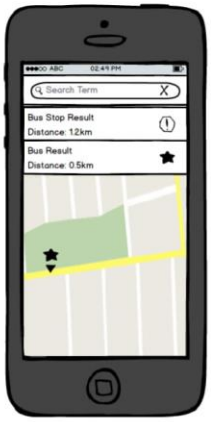


Figure 11 Search Bar GUI

4.1.6 Create Alert

The create alert dialog box is displayed when the user taps the 'Create Alert' button on either the bus or bus stop info panels.

The alert displays the name of the time slot and bus stop to set the alert for, and allows the user to select days to repeat the alert for.

The alert can be closed by tapping cancel or the top-right X button. The alert is saved when the user taps the 'Confirm' button.

Figure 12 Create Alert GUI

4.1.7 Modify Alerts

The alert settings screen can be accessed from the options menu.

This screen lists all the currently active alerts. Each alert displays the name of the bus stop it is currently assigned to, and the time slot. It also displays the days that the alert should repeat.

Tapping on an alert will open its configuration menu, like the 'Create Alert' UI. From here, the user can confirm the changes or delete the alert.

Figure 13 Modify Alerts GUI

4.1.8 Bind Simulated Person

The dialog to configure the SP binding is accessed through the options menu.

When the 'Bind SP' dialog is displayed, the user should fill in the SPID field and then tap the 'Confirm' button to finish the binding.

Once the binding is complete, the GPS co-ordinates on screen will match the dialog co-ordinates of the SP.

The dialog can be dismissed by tapping the 'Cancel' or the X button.

Figure 14 Bind Simulated Person GUI

4.2 Application Programming Interfaces (API)

The app and simulation will interact heavily with the web service's API. As explained before, there are two different API types hosted on the web service. The Standard API is for interfacing with the simulation and providing the basic features of the app that would be used in production. The Developer API is for connecting the binding feature to the simulation and for any other non-critical functionality such as logging.

The app will also interact heavily with the Google Maps API. When the app receives data that must be transposed onto the map, the Google Maps API will handle the rendering and the placement of the map icons. From this, the rest of the app can function by interfacing with the Google Maps API.

The simulation should push position data to the Standard API. However, the simulation elements must exist within a simulated real-world environment that can overlay the Google Maps API data. The simulation will pull terrain and road data from the OpenStreetMap project using a Unity package. By doing this, the terrain and roads will be accurate to the real world; therefore, the co-ordinates that are send back to the API and to the app are accurate on Google Maps.

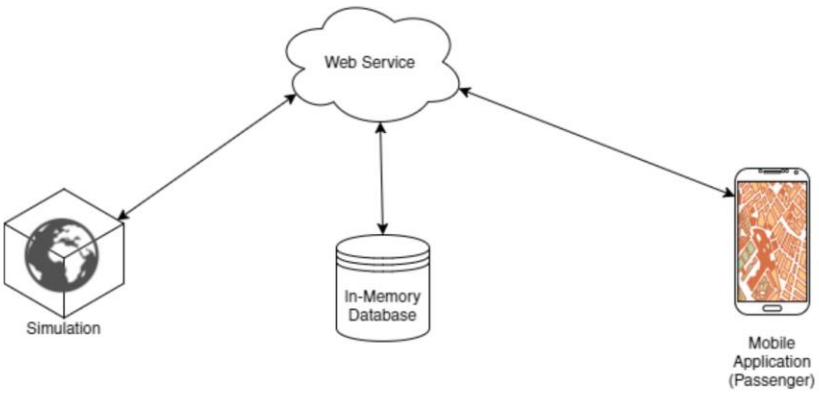


Figure 15 Three-Tier Architecture

5 System Architecture

5.1 Mobile App

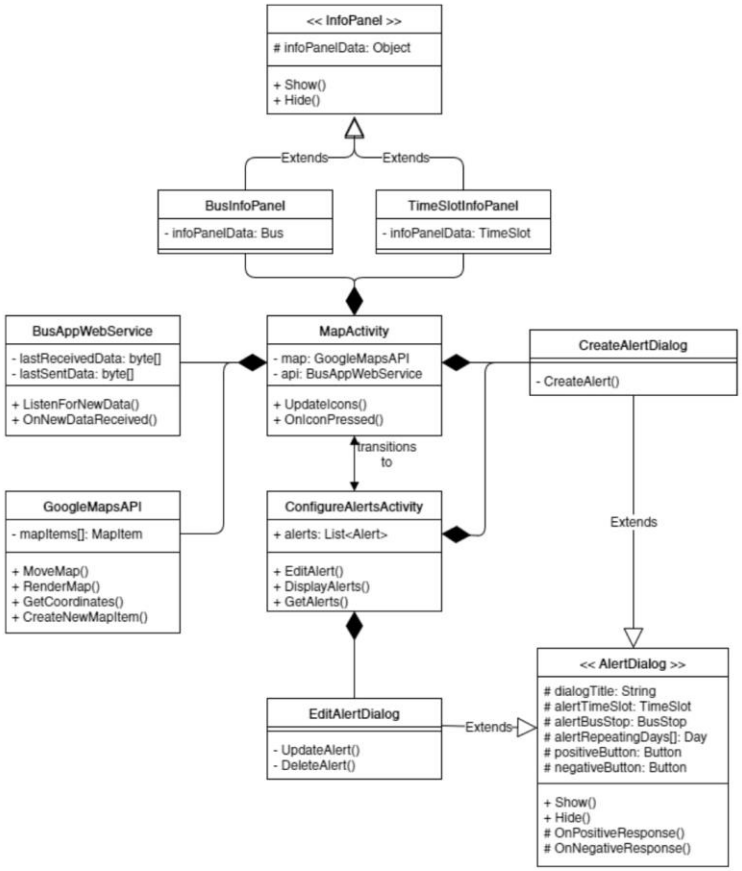


Figure 16 Mobile App Class Diagram

This is a class diagram of the Android mobile application. I chose this architecture because it is the most straight-forward way to build the app. The Alert Dialogs and Info Panels both inherit from their abstract classes, which means that if extra types of panels or dialogs need to be implemented, they can just inherit from their abstract parents. The MapActivity is the main activity of the app, and essentially acts as a wrapper for the Google Maps API and the BusAppWebService.

5.2 Web Service

5.2.1 Get Map Data

Endpoint: *GET /api/map*

Description: Downloads formatted data of all busses and bus stop co-ordinates to be placed into the map view.

Parameters: None

Returns: Formatted Map Data

5.2.2 Put Map Data

Endpoint: *PUT /api/map*

Description: Uploads formatted data of all busses and bus stop co-ordinates to be placed into the map view.

Parameters: Formatted Map Data

Returns: None

5.2.3 Get Bus Data

Endpoint: *GET /api/bus/{busReg}*

Description: Downloads all data relevant to the passed in bus object.

Parameters: Bus Registration Number

Returns: Formatted Bus Data

5.2.4 Put Bus Data

Endpoint: *PUT /api/bus/{busReg}*

Description: Uploads formatted data belonging to the bus.

Parameters: Bus Registration Number

Returns: None

5.2.5 Get Bus Stop Data

Endpoint: *GET /api/stop/{stopId}*

Description: Downloads all data relevant to the passed in bus stop object.

Parameters: Bus Stop ID

Returns: Formatted Bus Stop Data

5.2.6 Put Bus Stop Data

Endpoint: *PUT /api/stop/{stopId}*

Description: Uploads formatted data belonging to the bus stop.

Parameters: Bus Stop ID

Returns: None

5.2.7 Get New Commands

Endpoint: *PUT /api/dev/changes*

Description: Downloads a formatted list of commands to be executed from the developer mode.

Parameters: None

Returns: A formatted list of developer commands

5.2.8 Put Bind SP

Endpoint: *PUT /api/dev/bind/{spid}*

Description: Uploads a bind command to the web server.

Parameters: The ID of the Simulated Person to bind the app to.

Returns: None

5.2.9 Put Unbind SP

Endpoint: *PUT /api/dev/unbind*

Description: Uploads an unbind command to the web server.

Parameters: None

Returns: None

5.3 Simulation

Development in the Unity engine is done with the entity-component programming paradigm. Instead of being strictly Object-Oriented, game object behaviour in Unity is defined by composition over inheritance. Each game object in Unity follows this architecture design pattern:

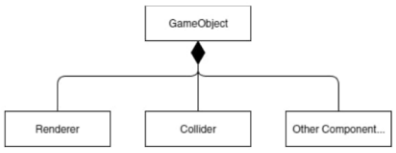


Figure 17 Unity Entity-Component Architecture

6 System Evolution

At the end of the development cycle for this proof-of-concept, the next logical step would be to make the project production-ready. This would require building a hardware device to be installed on busses to send data to the Standard API. It would also require developing an online web application for bus companies to manage their fleet. However, some changes will have to be made to the web service to make it scale better for thousands of users and busses at once. The simulation would be abandoned, as the Standard API should be almost production ready.

From the passenger's point of view, the app should be updated to include more features, making it more enjoyable to use. For example, a journey planner could be worked into the app after the 1.0 release. This would further enhance the app's ability to make public transportation easier to use and more ubiquitous in our lives.

Social Media scraping could also be a future feature. Looking at social media pages for events could allow the app to determine when there's going to be an extraordinary amount of traffic in the future. This could then be factored into the estimated ETA and a push notification or email could be sent days prior to the traffic delay to inform the user of the delay ahead of time.

Of course, with these extra features, the system requirements increase as well. The web service may eventually have to be refactored into a distributed system, depending on the workload. A large portion of the potential userbase is also being left out by not developing an app for iOS. Developing an iOS app should be made a priority and should be released alongside or soon after the Android application. Once the web application is built for the bus companies, a consumer facing web application would also be a good way to improve the user experience.

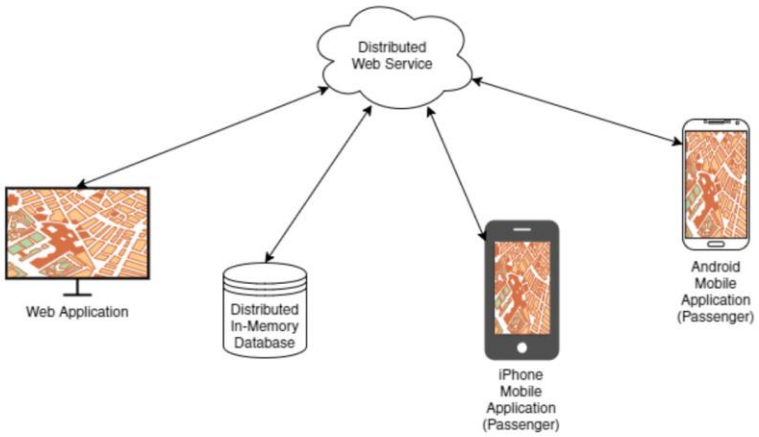


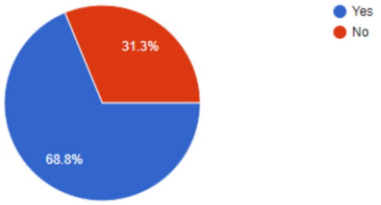
Figure 18 Future System Architecture

7 Appendix

7.1 Survey Results

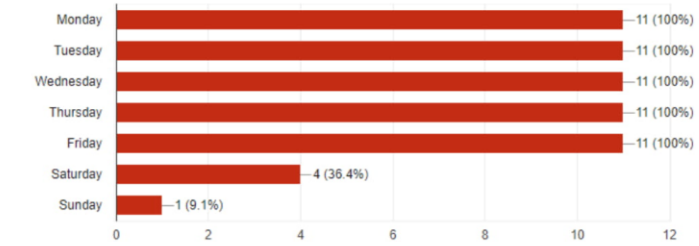
Do you take the bus as part of your daily routine?

16 responses



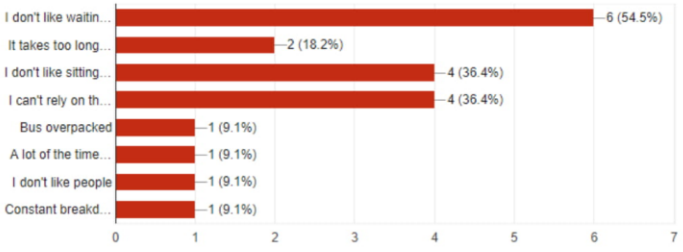
Which days do you usually take the bus?

11 responses



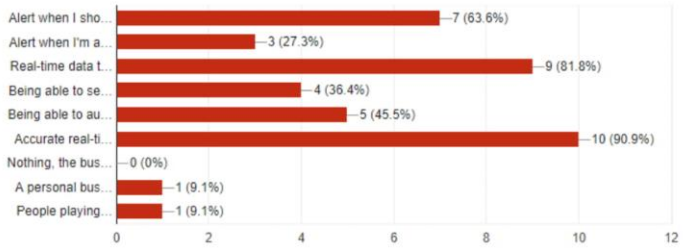
If you find taking the bus annoying, what do you find annoying about it?

11 responses



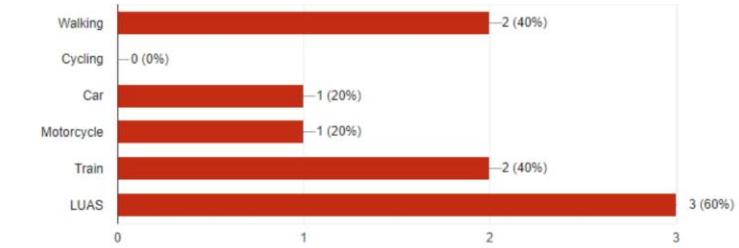
What would make taking the bus more enjoyable?

11 responses



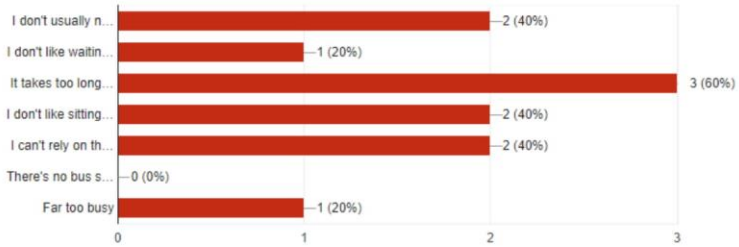
How do you usually get around?

5 responses



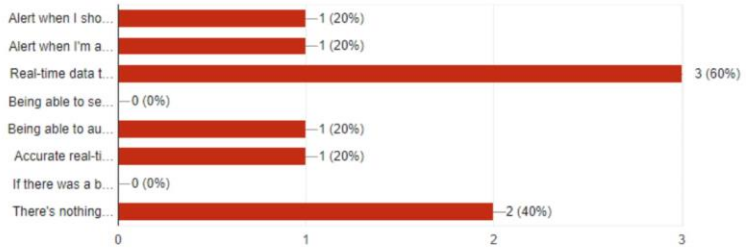
Why don't you usually take the bus?

5 responses



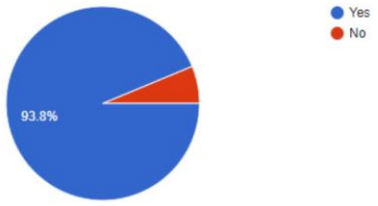
What would convince you to take the bus more often?

5 responses



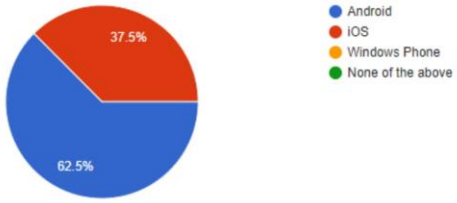
Would you use this app if it were available now?

16 responses



What Operating System is on your phone?

16 responses



6.4 Monthly Journals

6.4.1 September

I spent most of this month coming up with an idea for my software project. While I had a lot of time to come up with an idea over the course of my internship, I hadn't really given it much thought, as fourth year felt months away. Coming up with a viable idea in a short amount of time was quite stressful. There were a few moments where I regretted picking IoT, since I felt that choosing Software Development would have given me a broader range of projects to come up with.

Since I have a keen interest in game development, I sometimes wanted to make a game related project. In fact, specialising in game development was my plan since before college. However, I heard a few people say that making a game for your final year project is generally not a good idea since recruiters in normal software companies use it as a filter for potential candidates. I wanted to keep my options open, and specialising in software development sounded a bit too familiar, so I opted to choose IoT. Game development didn't run this year anyway.

I spoke with some of my colleagues during my internship and they gave me some great advice for choosing a final year project. Taking their advice into account, I chose some criteria for my final year project:

• The project must be interesting enough to keep me working on it for the whole year.

• The project scope must be scalable depending on the project schedule.

• The project must investigate solutions to a problem, or improve a service.

• The project must be marketable.

• The project must not include robotics, since they're too hard to deal with.

• The project must not be a video game.

After a few weeks of thinking, it was the day before the pitch and I had two ideas on my shortlist. One was a robot that links up with video services like YouTube,

Twitch and Skype. Essentially a video camera on a remote-control car. The other idea was a sensor platform in busses to give bus companies the ability to plan their routes more efficiently and to give more information to their customers.

I spoke to my friend Dan about this idea and he helped me narrow it down to just focusing on the passenger functionality of the app. My final idea included using sensors on the bus to find out data like how many seats were available, the bus location, etc. Using this data, users could automatically hail a bus based on their distance to the bus stop, they could calculate their ETA based on multi-trip journeys, and a lot more functionality could be extruded from this idea of putting sensors on a bus. My project idea filled out all the criteria, except for one. I still didn't find it interesting enough, and my desire to do game development work was still strong. There was another problem still, I don't have a bus to develop the app on. My solution was to build the 'real world' implementation of the project in a simulation using the Unity3D game engine. It would be just like making a game!

The next day, my project idea was accepted.

6.4.2 October

This month I spent my time working on the project proposal. At the start of the month, I worked on the Objective and Background sections of the proposal. On the 12th of October, I sent in my first draft of the proposal to Dominic and I had a proper meeting with him on the 18th of October.

During the meeting on the 18th, Dominic gave me feedback regarding the Objective and Background sections of the project proposal. I also got some suggestions regarding the Requirements Specification document. I made some changes over the next few days and I returned it for more feedback. On the 22nd of October, I got the greenlight on my proposal and I uploaded it that day.

I haven't started working on the Requirements Specification yet, I'll begin that once I take care of other module assignments.

6.4.3 November

I began work on the Requirement Specification document on Thursday, 16th of November as I was very busy with other modules this month. I spent the next few days filling out the Requirements Specification document until I had a first draft ready on the 18th of November. Once I had completed the first draft, I sent it Dominic to get some feedback.

From the 19th of November to the 21st, I began working on my project prototype. I set up a Unity simulation, a Sinatra web server and an Android test application to begin the development of the prototype. By the 21st I had a basic prototype working between the three systems, however a few minor changes had to be made for it to be ready for the mid-point presentation.

Dominic replied to me on the 20th of November with good feedback. I was very busy this week as well so I couldn't make the changes to the document until Wednesday the 22nd of November. I then uploaded the final version of the Requirement Specification.

I'll finish the prototype on the weekend of the 1st of December, in time for the mid-point presentation.

6.4.4 February

I didn't have a lot of time to work on the project this month as I was focusing on other Continuous Assessments within the degree. I did manage to fix some minor bugs and I also cleaned up some code after the prototype from December. I managed to add in some small easy to implement features such as a random name generator for bus passengers and a quick refactor of the API and Control Input systems.

The main bulk of the work was attempting to set up the Travis CI service for my project. This involved a lot of configuring and pushing to git to synchronise my GitHub and Travis CI accounts. I couldn't get the service working correctly by the end of the month, so I abandoned the CI requirement as I needed to start developing features as soon as possible.

6.4.5 March

This month was spent focusing entirely on implementing the bulk of the project's functionality. In particular I designed and implemented the object model for the simulation; for example, Busses, Bus Stops, Bus Routes, Timetables, Companies etc. I also implemented a Scheduling System that allows the simulation to trigger an event set at a certain time. For example, when a bus must start servicing a stop according to the timetable, the Scheduling System will notify the Bus to service this stop.

A large part of this month was also spent implementing the Timetable editor. This allows the user to create and modify timetables using a custom build editor window. The bus timetable can display multiple bus services/routes at once and keeps the correct order of the bus stops by applying a topological sort to all of the bus stops in the selected bus routes.

Passenger Boarding and Disembarking was also implemented, along with the ability for the bus to drive along a route and stop at each stop once it's hailed. Without getting into too much extra detail in this entry, the foundation of the simulation was implemented in this month; however, a lot more work has to be done to implement the web server and Android application.

6.4.6 April

Unfortunately, I was too busy working on other Continuous Assessments and I didn't have time to work on this project.

6.4.7 May

This month I began work on finishing my project by designing my poster's initial design. Once the design was done, I immediately started working on implementing the Android application section of the project. This required the development of a web server using Sinatra to store any long-term information (such as Route and Bus Stop data), and an MQTT service for real-time information (such as Bus Position and Capacity). I decided against using AWS IoT for this project as it is not supported by Unity.

I instead decided to use PubNub as it has a similar Publish/Subscribe functionality as AWS IoT, a more straightforward setup process, is supported by Unity and I also had experience using it to build a messenger app in 2nd year.

Setting up PubNub was straightforward, however the servers went down on the 5th of May, which disrupted my plans to work on the app's functionality. I then decided to use this time to start work on the Sinatra web server. It took me a few days to get this working as expected as I had to setup PostgreSQL on Heroku, and to figure out a good way to store JSON in the database without it causing any errors regarding character escapes. After some trial and error, I was able to use base64 encoding to store the JSON.

While I was working on the web server, PubNub services were repaired and I was able to start work on sending over the JSON data from the Unity simulation. I also was able to start developing the Android app's functionality. Not everything went as expected, a lot of time was spent recoding the object model that was present in the simulation. Due to time constraints, I had to make some sacrifices in the design that would prevent some original features that were in the requirements specification to be implemented in this project.

It's now the final day of development. I couldn't implement as many of the features that I was hoping to implement at the project. However, a solid groundwork was implemented and I'm proud of what I've accomplished within this academic year. The project was ambitiously scoped and while I couldn't implement everything, I'm happy that I was able to implement an MVP version that can communicate the concept of the project, while also improving my skills with the Unity engine, Sinatra framework, and Android Studio.