Pathway Project Report

Cory Sabol, Daniel Cregan, Johnny Bragg, Eric West, Chris Carney

September 2017

Contents

1 Project Definition

Pathway is a generic software, green-field development project of a native android application that we will be building with the idea of socializing the activity of exercise, in hopes of encouraging people to become more active. The application is targeted at users of all ages and levels of fitness. This task would be accomplished by the application by use of social features that would allow users that created accounts through the application to challenge those within a certain geographical distance of the user/s to beat their time on tracked, saved, and shared routes the user/s have walked, ran, etc. The application would solidify this by showing each user their improvement over the lifetime of usage of the application user on their account page within the application. It would also maintain leader-boards on popular routes in a user's current area encouraging the user to challenge and beat those above them to take their place on the leader-board.

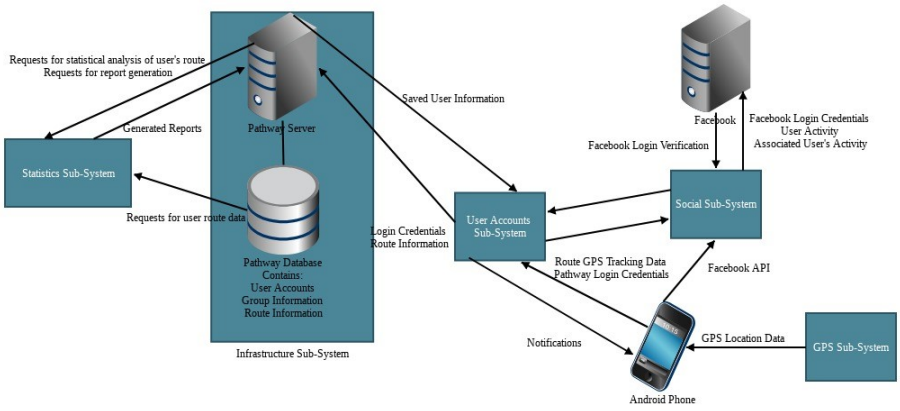


Figure 1: System Model

2 Application Structure And Flow

2.1 Pathway Structure

Pathway will be comprised of five primary components, as shown in the System Model in Figure 1:

1. Database Backend
2. User Accounts
3. Social Interaction
4. Geolocation
5. Statistics

The above systems are detailed in full in their respective subsections in Section 3: Sub-Systems of this report, but will we will briefly outline the functionality of each here and discuss their high-level interaction with one another.

The Database backend subsystem will serve as the data foundation for the application by providing data storage and look-up crucial for the functionality of all other subsystems. Each subsystem will interact with the database through an API developed for the application to facilitate passing of data back and forth.

The User Account subsystem will provide each user with a persistent experience across the application while maintaining account security for each user, ensuring that user data is correct, retrievable, and secure.

The Geolocation module will provide the primary functionality of the application, which is the collection of geographic route data associated with a user account and used in the Statistics subystem. This subsystem relies heavily on device sensors to generate accurate data as well as several web services to supplement the collected data.

The Social system will make use of the Facebook API to provide social networking features to users of the app, and is the primary avenue of motivation provided by the app, through social interaction.

Finally, the Statistics subsystem will be how the application allows users to derive and track useful information from repeated workouts. Using collected route data, users will be able to analyze performance metrics.

2.2 Pathway Flow

The Database system will necessarily communicate with systems at some point in the execution cycle of the application. Further, the Social and User systems will be dependent on each other, while also retrieving information from Statistics as needed. The Statistics module will rely on data collected by the Geolocation module and saved account data from the Users module to calculate metrics.

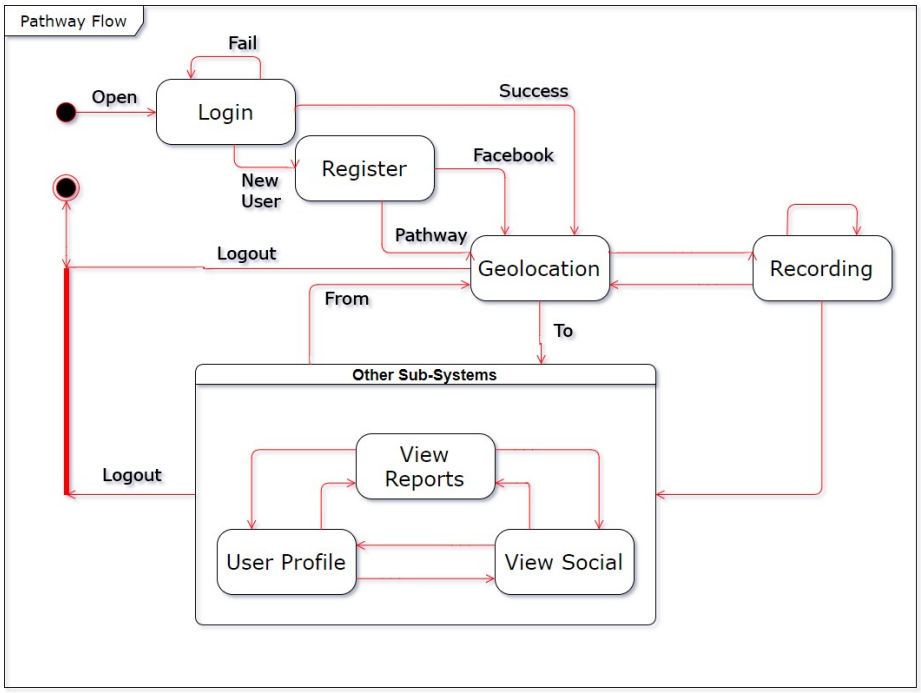


Figure 2: System Flow

2.3 UI Description

Pathway will rely on Figure 2. may also be used to describe a general flow of the user interface, which is as follows:

1. Upon loading the application, the user is met with a login screen and prompted to login.
2. If the user has an account, they may login directly.

   If unsuccessful, the user is returned to the login prompt.

   If successful, the user is taken to the main UI fragment, the geolocation fragment.
3. New users will be prompted to register, using either Pathway or Facebook.
4. The geolocation module depicts a map with the users location, routes in the immediate vicinity, and a start button.
5. If the user presses start, the application begins recording user location and draws it to the map.

   User can record from the start of an existing route if tapped.

6. From the map view, the user may use a drawer panel to navigate to their profile, a summary of their social page, or metrics for the user and/or current route.
7. From any of these views, the user may navigate to all other views using the navigation drawer.
8. From the navigation view, user may select logout to be taken back to the login screen.

3 User Requirements

3.1 Functional

1. Request an account be created.
2. Request that an account be logged into.
3. request that an account be logged out.
4. delete/remove an account, with appropriate privilege levels.
5. update an account's information.
6. create a route in the database.
7. delete a route in the database.
8. update a route in the database.
9. query for statistical calculation on a user's data.
10. The user will be able to link their Facebook account with the app to create an account
11. The user will be able to fill out a form based application to create an account
12. The system will allow the user to challenge other users within their area through the application interface
13. Users will be able to compete for best time on routes in their area
14. The system will be designed to accommodate future updates and additional functionality as needed.
15. The system will be available for constant use, except during a small as of yet undetermined timeframe to allow for system maintenance and updates.
16. The system will be stable, minimizing application crashes, freezes, or hang-ups.

17. The system must be energy efficient with respect to the mobile device.

18. The system must be efficient with respect with to cellular data consumption.

19. If Facebook is down the account integration feature in the app will be expected to have downtime as well.

20. The system will use collected geospatial data and user demographic information to present analysis of the users performance.

21. The system will take user data and generate leaderboards for heavily used routes in certain areas, in order to allow users to compete for bragging rights on those routes.

22. The system will generate a report after a run detailing the users performance with comparisons of previous runs.

23. The system must generate reports that should be easily readable and detailed while avoiding being confusing to users.

24. System will use device sensors to record GPS data in a Latitude/Longitude format.

25. System will use device sensors to record user elevation.

26. System will plot Geolocation data against a Google provided background map.

27. System will record relative time stamps to use in calculation of speed and direction.

28. System will display user activity duration, speed, distance, current elevation, and coordinates during data collection.

29. System will display user routes clearly and in real-time as data is being collected.

30. System will allow the user to save individual routes to a comma-delimited file for personal use.

31. System will display traversed routes in the User's immediate vicinity.

32. System will allow a user to tap on a route to retrieve and view information associated with that route.

33. System will store a users routes in a particular format for use in Pathway.

34. System will maintain a local device database to store a user's route data.

35. System will be available as long as a persistent data connection is allowed.

36. System will be made available in a limited capacity with reduced accuracy and no map service while no data connection exists.
37. System will accurately and precisely locate the users position within an acceptable margin of error, allowing for a larger margin where service is weak or none exists.
38. The System will allow the user to create an account that will be kept up to date by the system.
39. The system for user accounts will keep track of a user's life time statistics and display them in an always up to date state, statistics include, life time steps, distance, best course, best time, etc.
40. The user accounts system will save all user routes in the appearance of a fanned stack of cards.
41. The system will allow the user to pull up these saved routes and see their best time or if that time has been beaten by anyone that they have shared it with.
42. The system will also create graphs that display the users progression over time of running a route, that user can see.
43. The system will be designed to accommodate future updates and additional functionality as needed.
44. The system will be available for constant use, except during a small as of yet undetermined timeframe to allow for system maintenance and updates.
45. The system will be stable, minimizing application crashes, freezes, or hang-ups.
46. The system must be energy efficient with respect to the mobile device.
47. The system must be efficient with respect with to cellular data consumption.
48. The system must generate reports that should be easily readable and detailed while avoiding being confusing to users.

3.2 Quality

1. The system will be designed to accommodate future updates and additional functionality as needed.
2. The system will be available for constant use, except during a small as of yet undetermined time-frame to allow for system maintenance and updates.
3. The system will be stable, minimizing application crashes, freezes, or hang-ups.

3.3 Platform

The client application will be written to support a minimal android version of 5. The back end will be built using Python3 and Flask, and will run inside of a Docker container. The database itself will be written in Python, MySQL, and SQLAlchemy and will also be housed inside of a Docker container. The container itself will be hosted on an Amazon Web Services (AWS) instance. The tracking and mapping of routes will be implemented through the use of the Google Maps API, and the statistical analysis of user data will be implemented with Python3 and Java, where necessary. The systems to provide statistical analysis will also be housed along with the back-end application. Specific technologies should be listed here, including version numbers of languages.

3.4 Process

The project will be completed over the course of the Fall 2017 semester, with regular progress updates provided every few weeks. We are currently deciding upon a software engineering methodology to follow, such as agile methodologies. As we near the actual development phase we will settle upon a methodology to adhere to.

4 Sub-Systems

4.1 Back-end Infrastructure - Cory Sabol

4.1.1 Introduction

The Infrastructure subsystem of the Pathway application is responsible for providing a stable API by which the client application can access, create, and update data stored in the database, which is also a part of the infrastructure subsystem. The API exposed also provided a means for the client application to query for various statistics that are calculated server side. However these functions aren't a part of the subsystem, they are their own subsystem which is merely housed on the server.

4.1.2 User Requirements

We will refer to any client application which may consume the API as a client from this point forward. The word "user" will refer to a person which may utilize the Android application and services provided by Pathway. Below is a list of the user/functional requirements that the infrastructure subsystem will satisfy. These are all actions performed by a client, not a user.

1. Request an account be created.
2. Request that an account be logged into.
3. request that an account be logged out.

4. delete/remove an account, with appropriate privilege levels.
5. update an account's information.
6. create a route in the database.
7. delete a route in the database.
8. update a route in the database.
9. query for statistical calculation on a user's data.

4.1.3 Platform Requirements

To implement the subsystem described in this document, the platform required must support standard HTTPs connections, be stable, and provide a way to establish RESTful architecture. That is why the following tools have been chosen.

- Python version ≥ 3, to write server side logic.
- Flask, to provide a simple, scalable web server that utilizes python.
- MySQL, to provide the database to house all user and application data.
- SqlAlchemy, to provide an Object Relational Mapper (ORM) for accessing the database in a more efficient and secure manner.
- Docker, to isolate the application in a consistent environment which can easily be moved from development to deployment.
- Amazon Web Services (AWS), to provide the physical server architecture needed to host a centralized server application.

4.1.4 Domain Analysis

4.1.4.1 Introduction

The following sections are devoted to examining the domain which the infrastructural subsystems of the Pathway application reside.

4.1.4.2 Extensions

The infrastructure for the Pathway project isn't necessarily extensible in the same way that the other subsystems are. However, it could be extended by making it more scalable. This would allow the system to be more redundant and support a larger volume of users. In addition to scalability, the system by it's nature, will have to be extended along with most extensions that could take place within the other subsystems.

4.1.4.3 General Domain Knowledge

This susbsytem falls into the category of backend web development, which is a domain that the developer already has a sizable amount of professional experience in.

- Because HTTP is transactional, a REST API incurs some network overhead.
- Python is a language which facilitates rapid development. Making it great for a web service.
- The flask framework isn't particularly well suited to handling large volumes of requests. Meaning we would need to port to a web server which scales better should we have a user base increase.

4.1.4.4 Clients and Users

The only clients that this subsystem can have are any applications which consume it's API. In the case of Pathway this will be all of the other subsystems which make up the native Android application.

4.1.4.5 The Environment

The computing environment for this subsystem will be an Ubuntu 14.04 instance which has been dockerized. This will be deployed to an AWS instance. The purpose of running the server side code in a docker container is that docker allows us to tailor the OS environment to cater specifically to the application. Which provides a more isolated development and production environment for the code, minimizing variables which could contribute to bugs and other such undesirable behaviors.

4.1.4.6 Tasks and Procedures

This subsystem does not replace an real world tasks or procedures. It is more like the glue which allows the application to network with other instances of the application and to house and retrieve data.

- Receive incoming HTTP requests.
- Process data sent to API endpoint via HTTP request.
- Respond to HTTP requests which were successfully received.
- Update the database.
- Provide user authentication.

4.1.4.7 Competing Software

Because this subsystem, by nature of it's purpose, is customized to facilitate the possibility of the actions that the client Pathway application provides, there is not necessarily any competing software.

4.1.4.8 Domain Similarities

Other applications may require a backend which is similar to the backend for Pathway.

- Other fitness applications such as Strava.
- Other social applications such as Facebook, or Twitter may have similar backend logic.

4.1.5 System Analysis

4.1.5.1 Introduction

This document describes the planning and analysis of the infrastructural subsystem of the Pathway project. Which encompasses the database, server logic and houses several other subsystems by providing an interface for various client side systems to access.

4.1.5.2 Process Model

Since this subsystem is not necessarily an entirely programmer created system, but rather a composition of application environments and configurations as well as primarily declarative logic to handle server routing, the high-level behaviour and structure of the system is best described using a sequence diagram rather than UML class descriptions. **Figure 2** shows a sequence diagram of the backend systems. It also describes environment encapsulation and which system objects belong to which tools.

4.1.5.3 Detailed Model

4.1.5.4 System Analysis

Detailing the model of the infrastructure really comes down to describing how the various components communicate with one another, and with other systems. The server itself communicates with the database via and ORM system, which is built into Django. The server can be communicated with the following example API;

1. API Example (not a full definition of the API)
    (a) /auth - handle user authentication and deauthentication
        i. /login - POST the user login data

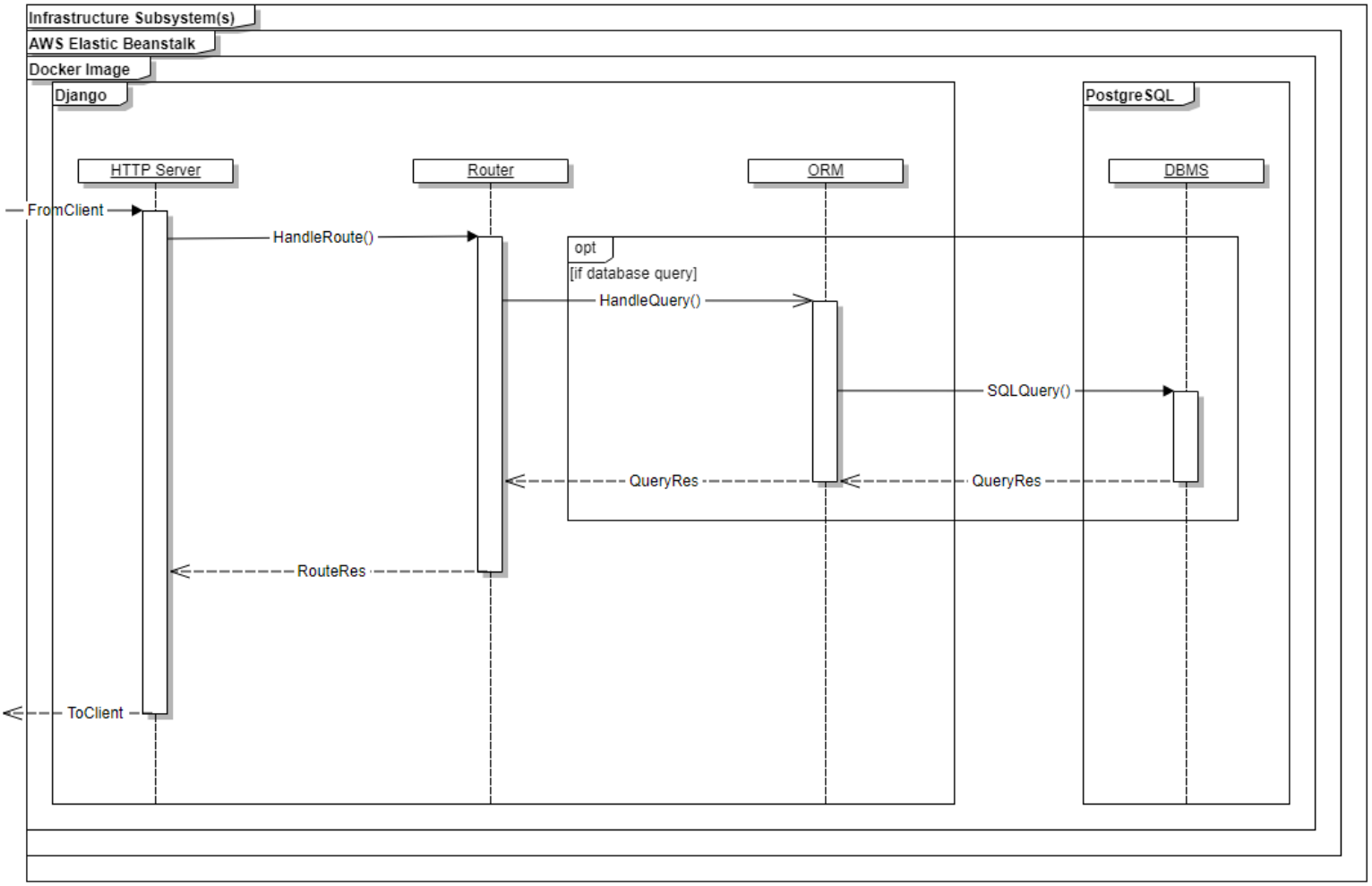


Figure 3: Pathway Backend Sequence Diagram

ii. /logout - POST request to logout current user, destroys session.
iii. /create - POST create an account from posted data
iv. /deactivate - POST deactivate user account (requires authentication)

(b) /users - GET all users
i. /[userName] - GET user summary belonging to userName
ii. /[userName]/[routes,stats,profile] - GET routes or stats or profile of userName

(c) /routes - GET all routes (paginated)
i. /[routeId] - GET specific route

Clients may directly craft and send http requests to the API endpoints, or java based applications may leverage the java library which is a wrapper to the server's API. This is the recommended approach, as it abstracts the work of making http requests and handling the responses into a native library which exposes an interface of functions and it's own types.

4.1.5.5 System Analysis

Since this subsystem relies heavily on both a webserver framework and a DBMS which means that we do not face the same challenges in determining efficient solutions that we do when we define and develop algorithms. The implementation of the webserver and the DBMS are hidden away from us, therefore we must consider which tools at our disposal are the best for our purposes. Rather than analyze specifically for algorithm efficiency (although we will do this should the need arise), we will focus on issues such as scalability, redundancy in the database, environment set up, resource size (how much data are we sending to the client and vice-versa), concurrent connections, and many other challenges to explore.

4.1.5.6 Use Cases

With respect to this subsystem a client refers to any application which interfaces with this subsystem.

1. A client wishes to authenticate a user.
   (a) Client makes call to 'auth/' API endpoint, passing username and password over HTTPs.
   (b) Subsystem authenticates username and password using the User Account subsystem.
   (c) Subsystem sets the client session to authenticated or non-authenticated.
2. A client deauthenticate a user.
   (a) Client requests to be deauthenticated.

(b) System destroys client session.

3. A client wishes to store a new route in the database.
   (a) Client makes a POST request to '/route/add/', with the route data in the body.
   (b) Subsystem stores route in database via ORM.
4. A client removes a route from the database.
   (a) Client makes GET request to '/route/del/[id]' with the id of the route to delete.
   (b) Subsystem drops route from the database.
5. A client updates a user profile.
   (a) Client make POST request to '/user/profile/update' with user profile data blob in request body.
   (b) Subsystem makes appropriate update to user entry in database.
6. A client requests a report on user data.
   (a) Client makes GET request to '/report/[type]/[date]' with report type and data in the url.
   (b) Subsystem passes type and date to the statistic subsystem to generate the report.
   (c) Subsystem returns report data as JSON string.

4.1.5.7 Scalability

Scalability is one of the largest challenges that is faced when developing a web based application as well as development time. For this system, it needs to provide access to a database as well as expose a RESTful API and an admin page. There are two choices that make sense here with respect to providing these features and reducing development time; Django and Flask. These two tools imply that we have decided to utilize Python3 on the server side, for it's ease of use. Django and Flask are both python based web frameworks, which provide essential HTTP functionality. However, Flask is much smaller than Django and provides less out of the box. Meaning that more developer effort is required to implement the features that we desire. Django on the other hand, is very much a "batteries included" framework. It provides several key things out right; an ORM, REST plugin, admin tools/pages, and deep database integration, and robust security options. This makes it the choice of web framework for the Pathway project (other security implications?). Django is also built with scalability in mind.

4.1.5.8 Hosting

When developing an application which requires a back end server, it's also important to realize that hardware is required in order to actually host the application. The choices for this come down to Amazon Elastic Beanstalk or Heroku. Both of these services provide similar things. Hosting of web applications, and their databases, as well as management/admin tools. However, Elastic Beanstalk is somewhat cheaper as a student, and provides better load balancing tools, which is important should the service ever need to scale according it's userbase. There is a big difference between 1000 concurrent users, and 10,000. Amazon web services has a great track record of being an industry standard when it comes to deploying hosting applications which may need to scale and handle large numbers of connections. The drawback to Beanstalk versus Heroku is that Heroku is vastly easier to use and interface with from a developer standpoint.

4.1.5.9 Database

Choice of database is a very important choice as well. For this decision however we have opted to go with the choice which aligns best with ease of setup and lowering developer time. PostgreSQL has very tight integration with Django, and there are extensive community resources with regard to it. At this stage of the project, the database schema is relatively simple, and it will certainly be expanded upon as we continue to develop. **Figure 2** shows the database schema.

4.1.5.10 Classes

There isn't much to discuss here, the only classes which will be developed for the infrastructure will be classes for the ORM (object relational mapper). These classes will be essentially be 1:1 with the tables described in **Figure 2**. These classes will make use of the Django models library, and will be used in the actual construction and initialization of the database.

4.1.5.11 Deployment

Lastly deployment from development to production is a complex topic worth study of it's own. For this challenge we could either create an isolated development environment with a tool like Docker, which will allow us to have a localized development environment which can mimic the production environment as well as being fully scriptable, or we could have each developer install python3, Django, PostgreSQL and any other tools to their machine and fight with inconsistencies across each machine. The choice here is pretty obvious. We have opted to utilize Docker to provide an isolated development and testing environment which contains exactly the tools necessary for the server application and database to run. This would only require that each developer use the Docker build script to build the image on their machine, and then develop

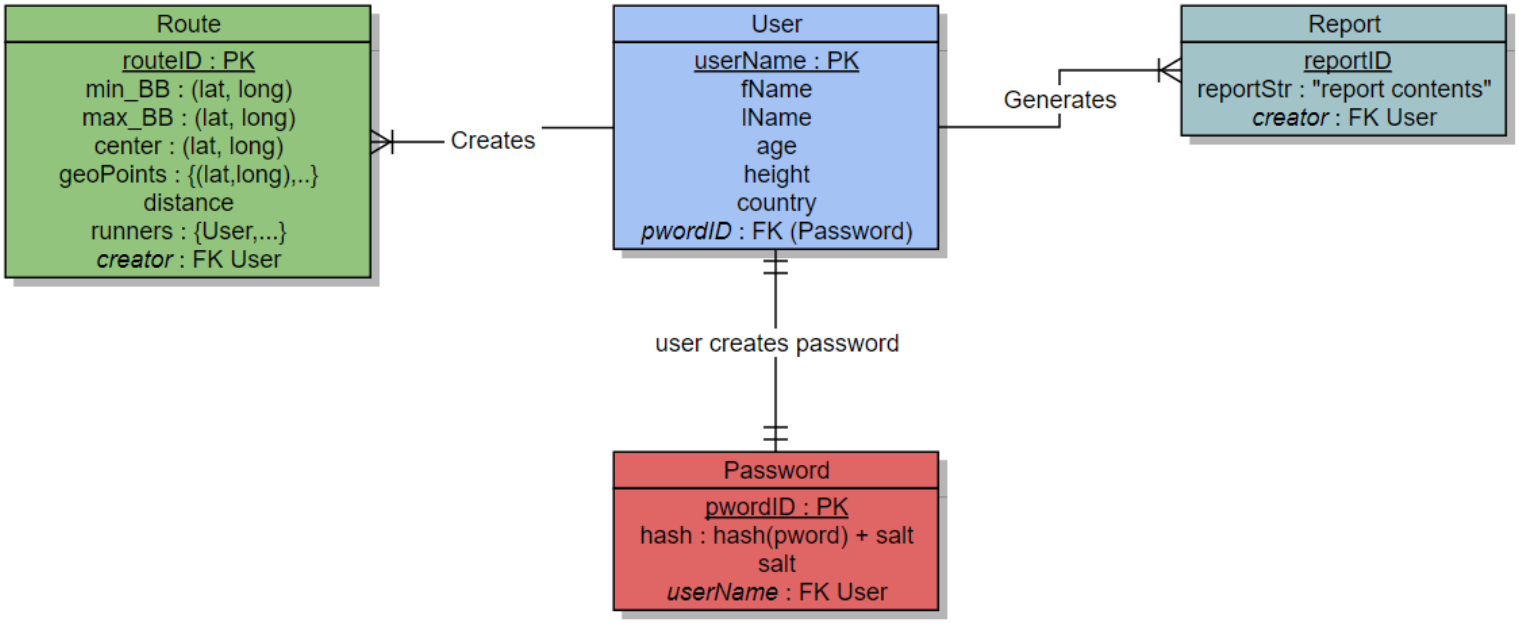


Figure 4: Pathway Database ER Diagram

and test against it. Amazon Elastic Beanstalk also has support for deploying dockerized applications.

4.1.5.12 Tool Selection

- Docker
- Django
- Python3
- PostgreSQL
- Amazon Elastic Beanstalk

4.1.6 Selected Source Code

4.1.6.1 Serializers

```
from django.contrib.auth.models import User, Group
from rest_framework import serializers

class UserSerializer(serializers.HyperlinkedModelSerializer):
  class Meta:
    model = User
    fields = ('url', 'username', 'email', 'groups')

class GroupSerializer(serializers.HyperlinkedModelSerializer):
  class Meta:
    model = Group
    fields = ('url', 'name')

class RoutrSerializer(serializers.HyperlinkedModelSerializer):
  class Meta:
    model = Route
    fields = ('routeID', 'dateCreated', 'creator', 'geoData')
```

4.1.7 Views

```
from django.shortcuts import render
from django.contrib.auth.models import User, Group
from rest_framework import viewsets
from pathwayserv.api.serializers import UserSerializer, GroupSerializer

# Create your views here.

class UserViewSet(viewsets.ModelViewSet):
  '''
```

```
    API Endpoint that allows users to be viewed or edited.
    '''
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserSerializer

class GroupViewSet(viewsets.ModelViewSet):
    '''
    API Endpoint that allows groups to viewd or edited.
    '''
    queryset = Group.objects.all()
    serializer_class = GroupSerializer

class RouteViewSet(viewsets.ModelViewSet):
    '''
    API Endpoing to handle creation and viewing of routes
    This is currently undefined.
    '''
```

4.1.7.1 Docker File

```
FROM python:3

MAINTAINER Cory Sabol
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
ADD requirements.txt .
RUN pip install -r requirements.txt
ADD . /code/
```

4.1.7.2 Docker-compose File

```
version '2'

services:
  db:
    image: postgres
  web:
    build .
    command: python3 manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      -db
```

4.1.7.3 Test Cases

We can see that the selected code snippets satisfy several of the test cases, which we define as use cases that have been backed by working code. Currently the code for the infrastructure subsystem satisfies several test cases, some more concrete than others.

- The system accepts HTTP requests and is capable of responding via HTTP.
- The system is able to handle the creation of a user.
- Viewing of users in serialized json format is also handled.
- Groups may also be created.
- Groups may also be viewed.
- The system has communication with the database.
- The system satisfies the need for administration with admin users and an admin portal.
- The system supplies a contained development environment which mimics production. i.e the system is deploy-able and meets the criteria for a sane development pipeline.

While this is not a full overview of the system, it is fully ready to supply a robust API and backing for the Pathway mobile application. The API only needs to be defined from it's current point. Team member which need to develop applications against the API or develop systems which run on the server need only run a local instance by utilizing Docker which is configure to run a version of the server application that is local to the developers machine and a database which is a clean slate for every developer. Once a version of the application is complete, the source and the docker configs can be deployed to an AWS instance, where the docker image will be built and the application server started.

4.1.7.4 Data Conversion Plan

Since this subsystem is the one which houses the data and provides access of the data to other systems, a conversion to a new implementation system or even server hardware would not affect the other subsystems. Only an API revision would trigger a possible need for a change in the way that the other systems handle the data belonging to the Pathway application.

This section is best explained with an example. If the need arose to build the server side API application in a new language, it would be trivial due to the decoupled nature of all the parts that comprise the entire subsystem. This is thanks to Docker, which allows the subsystem to be built in interchangeable layers. Each layer effectively has it's own isolated OS in the form of a docker image in which the relevant software runs. Each layer (container) communicates

via a network constructed solely for the containers which make up the application as a whole. Consider an entire rewrite of the API layer from Python to Haskell (something which is plausible in the future). This event would in no way affect the existing data in the database container, as the database container is completely isolated from the API layer. The only changes that would cascade through all layers would be changes to the database schema itself. This is because every other part of this subsystem depends upon it.

4.1.7.5 Design Choice

At the start of this project it was conceptualized that the user account and login system would handle user authentication and creation. However as the two systems were further developed it became evident that rather than writing a separate module which the sever would interface with for user authentication, it would be better to utilize the built in features that Django provides for user authentication, as they are better tested and already built, and simply expose an API. So, the functionality for user authentication was moved out of the User system, developed by Daniel Cregan, and into the Infrastructure system. This allowed the implementation of such functionality to be done more rapidly and allowed the development of the User system to focus on other aspects.

We also had to make a change in what server infrastructure provider we utilized. The original design was to use AWS. AWS is extremely powerful and sees the most usage for hosting complex applications in the "cloud" [**?**]. However with that power comes complexity. AWS is an IaaS service, meaning infrastructure as a service, platform. Because of this there are many options for what to use, how to use it, and how to configure it. This introduced unnecessary complexity into the subsystem's development compared to our hosting and deployment needs. So, the choice was made to utilize DigitalOcean instead. DigitalOcean, on top of being cheaper, was also much simpler to use, as they provide developers with a virtualized linux machine somewhere in their data centers, and a developer is able to simple SSH into the machine and begin configuring the application. This was preferable because the developer of this subsystem was already familiar with this kind of work flow.

4.2 Social - Chris Carney

4.2.1 Introduction

The Social subsystem will utilize some simplistic features that enable the user to interact with other users using the app within their local area. We will be implementing certain aspects from a Facebook API that will allow the user to integrate their Facebook profile when they first create an account that will let them expand their use of the app to include others. The user will also have a form based alternative to make a profile which would give the user an opportunity to create a profile straight from the app but would limit its inclusivity of other users and just be used to keep records of their routes that

they recorded without any social aspect in place. The app will give the users an option to choose a route in their specified location which will then list the top times that the route was traveled, then the user will be able to choose a top time that is displayed in the list and will be able to implement the Challenge function, which is similar to a poke on Facebook. After the user picks the time they want to challenge on a certain route, and they complete it, the person that the user chose to challenge would be notified if their time was beaten depending on if the user completed that route in a faster time. We will also include an achievement system feature that will give incentives the user based on certain accomplishments within the app.

4.2.2 User Requirements

- The user will be able to link their Facebook account with the app to create an account
- The user will be able to fill out a form based application to create an account
- The system will allow the user to challenge other users within their area through the application interface
- Users will be able to compete for best time on routes in their area
- The system will be designed to accommodate future updates and additional functionality as needed.
- The system will be available for constant use, except during a small as of yet undetermined timeframe to allow for system maintenance and updates.
- The system will be stable, minimizing application crashes, freezes, or hang-ups.
- The system must be energy efficient with respect to the mobile device.
- The system must be efficient with respect with to cellular data consumption.
- If Facebook is down the account integration feature in the app will be expected to have downtime as well

4.2.3 Platform

- The system will be developed using Android v5 (Lollipop).
- The system will be developed in Android Studio.
- The system will employ the Facebook API.

4.2.4 Domain Analysis

4.2.4.1 Extensions

- Increased social networking functionality.
- Broaden integration for other social network sites such as Twitter and Instagram.

4.2.4.2 General Domain Knowledge

While many individuals like to share their workout information on social media sites, there does not currently exist an integrated fitness oriented social networking option.

4.2.4.3 Clients and Users

- Targets users capable of operating a compatible device that this application could be installed on.
- Facebook Users.

4.2.4.4 Computing Environment

Primary functionality will be coded in Android 5.0 (Lollipop) and as such will be unsupported on older versions of Android. Specific system requirements are not yet determined. No additional client requirements will be necessary for database access, other than an active connection to internet.

4.2.4.5 Tasks and Procedures

- Linking Facebook account to the app: The app will ask if the user has a Facebook account they would like to link upon account creation.

  This will prompt Facebook to authorize permission for the app to access its functionality such as sharing posts straight from the app.

- User Challenges a record time on the leaderboard for a route: User chooses a route within their area, which then lists the top times that the route was completed.

  User has option of picking what time they would like to challenge.

  Upon completion of the route, if the user who challenged the time completes the course in a shorter time, then the user that was challenged would receive a notification that their time was beaten.

4.2.4.6 Competing Software

The following applications/software may fulfill similar roles or offer similar functionality:

- MyTracks - Offers only basic data collection (Lat/Long position, elevation) dependent on device sensors, and no analysis options.
- FitBit App - Offers data collection and basic analysis of user performance.
- iOS Health - Offers basic data collection and limited analysis of performance.
- LG Health - Offers route tracking and calorie journal. Still lacks any social networking component.

4.2.4.7 Similar Domains

Many applications are similar to Pathway in the sense that they make use of geospatial data location and/or social networking user accounts. While no other application we found combines all aspects of our project in the same way we do, it will be worthwhile to look at other applications, such as Waze, Facebook, and other apps to gather some understanding of how to efficiently implement these systems.

4.2.5 System Analysis

4.2.5.1 Introduction

Since the User Accounts subsystem will be focusing on implementing the choice to create a profile through the app, the Facebook API will be the majority of what we will be using in this subsystem to help authorize a users profile and also gain access to the users profile information given the option to control what permissions the user will be able to access from the app whether it be from their personal info to their friends lists.

In order to do this, we will first need to utilize a program called OpenSSL. OpenSSL is a software library for applications that secure communications over computer networks from being accessed by unauthorized users. It contains an open-source implementation of SSL and TLS protocols. It's core library implements cryptographic functions and provides various utility functions. OpenSSL also utilizes wrappers which allows its library to be used across many different computer languages so that there are no compatibility issues. With OpenSSL, we will be able to obtain a key hash. A key hash is a 28-character string that Facebook uses to authenticate any type of exchanges of information in between Facebook and the Pathway app. If a key hash is not obtained, there will be no possibility of Facebook integration within the app.

After establishing the connection to the users Facebook account, we will be implementing the CallbackManager class provided by the Facebook API which determines how methods are called and what values are returned from

those methods which will interact with other classes within the Social subsystem while also interacting with each of the other subsystems. During this process, we have the option to list what permissions the app will have access to such as the user's basic profile information, email, or list of friends. This is how we plan to get access to the user's list of friends that also have the app installed which is already determined through the Facebook API.

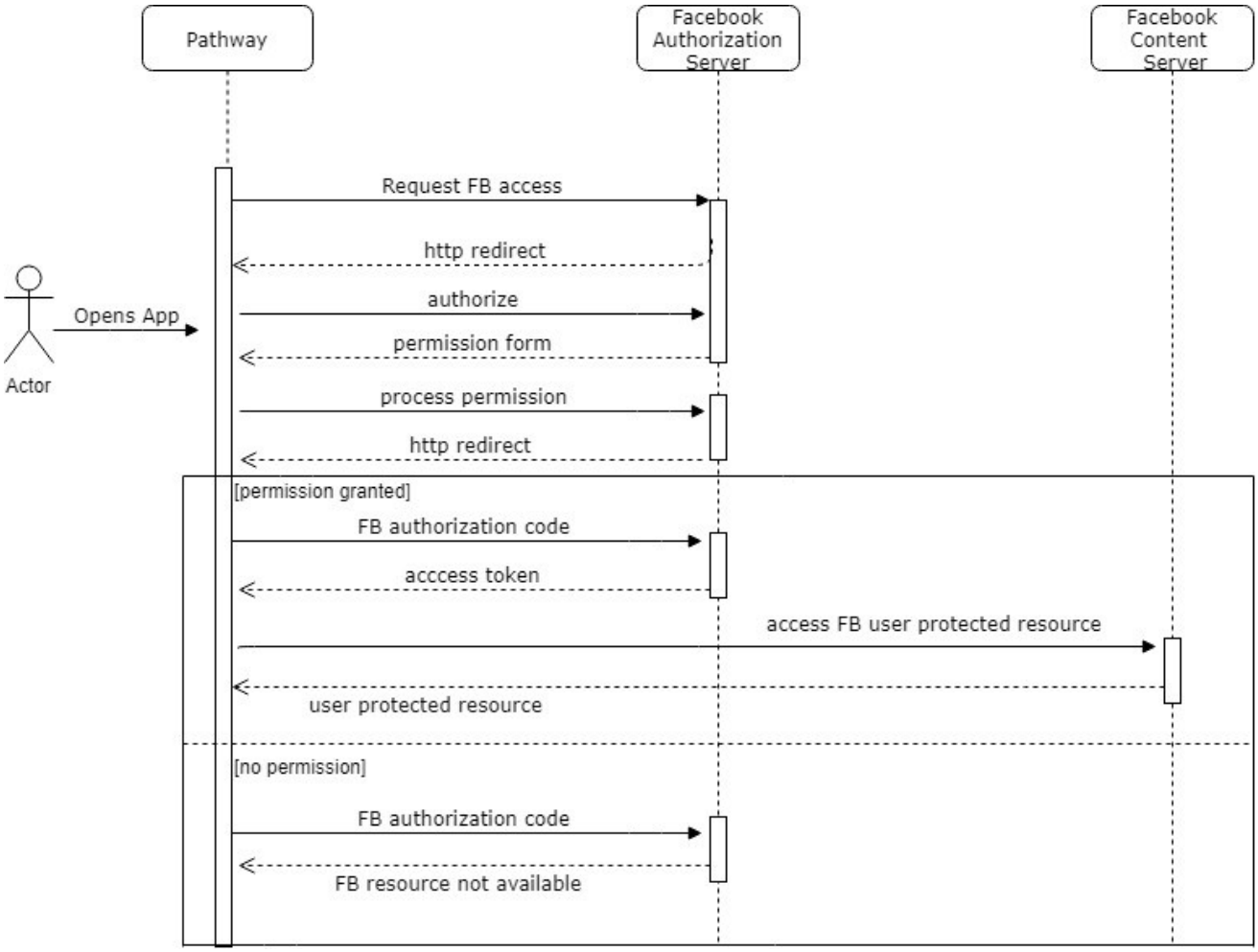


Figure 5: Pathway social sequence diagram

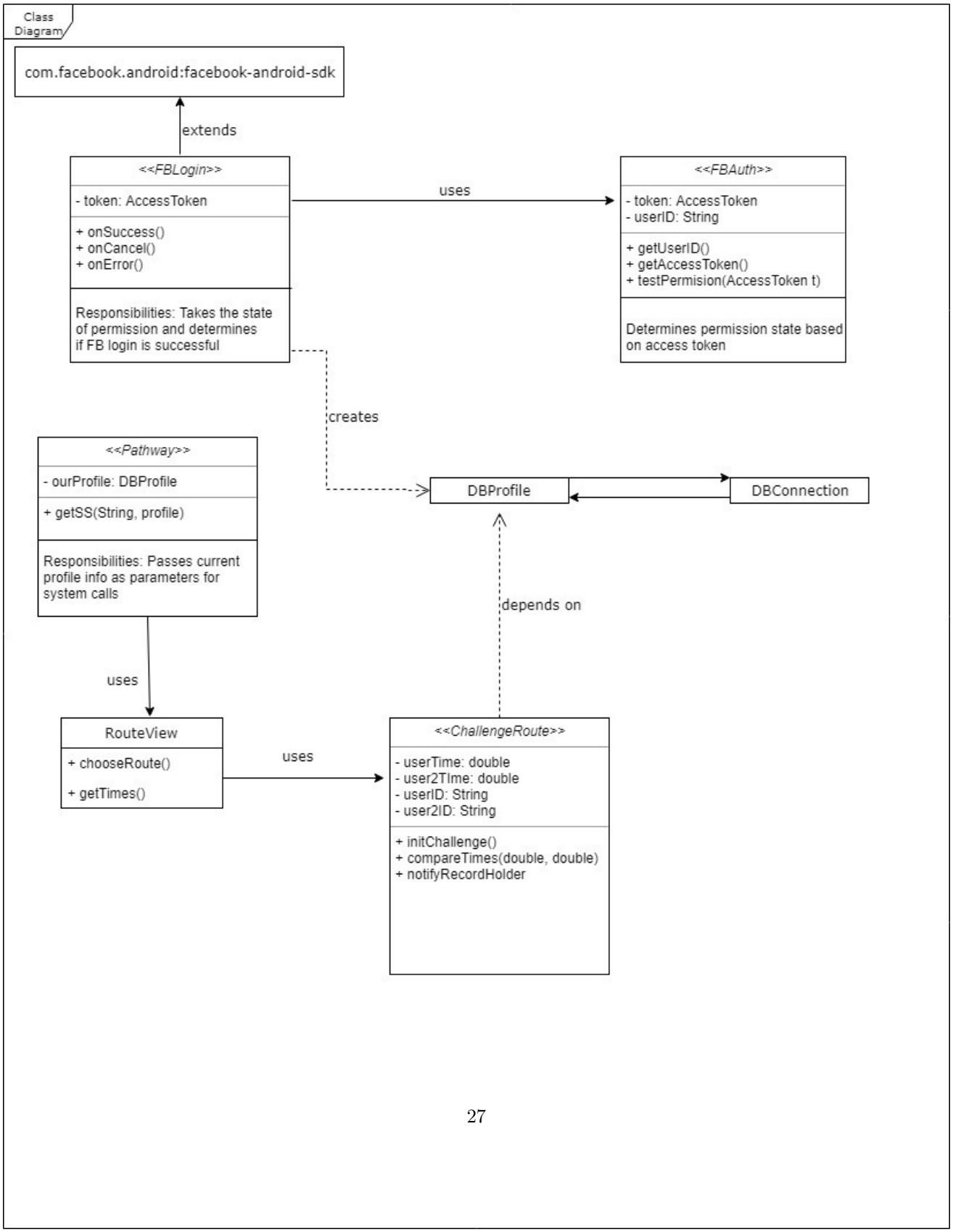


Figure 6: Pathway social UML diagram

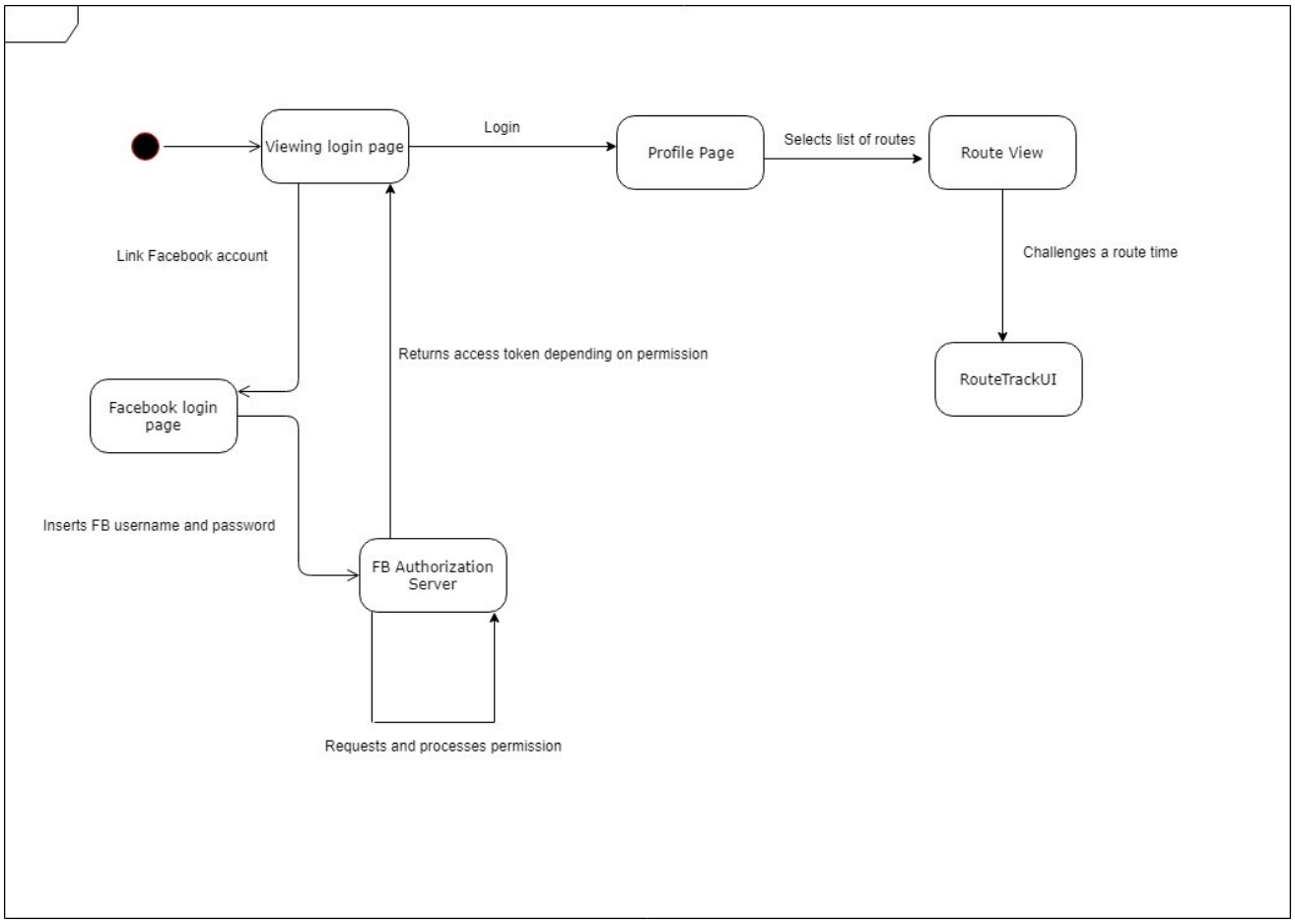


Figure 7: Pathway social activity diagram

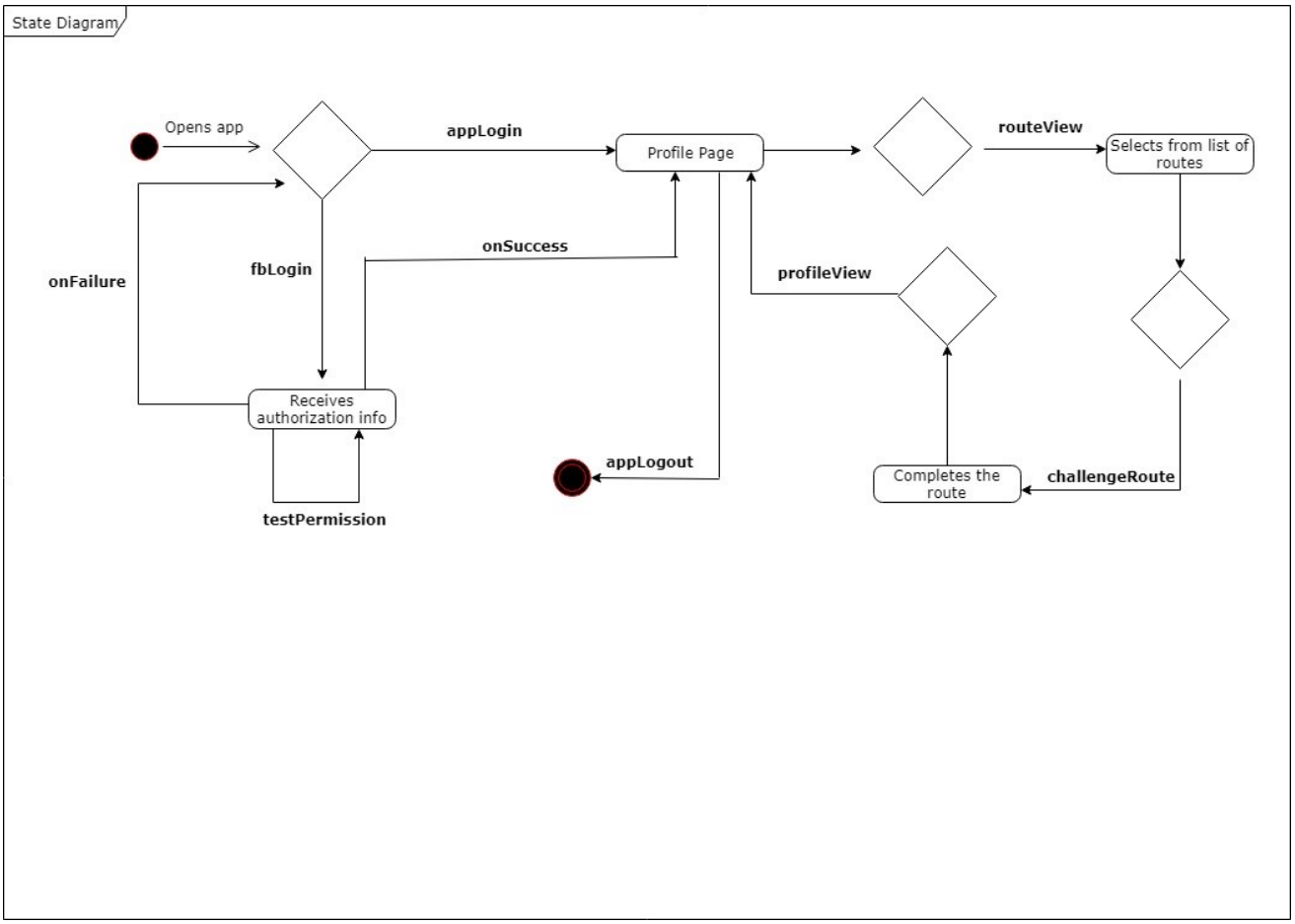


Figure 8: Pathway social state diagram

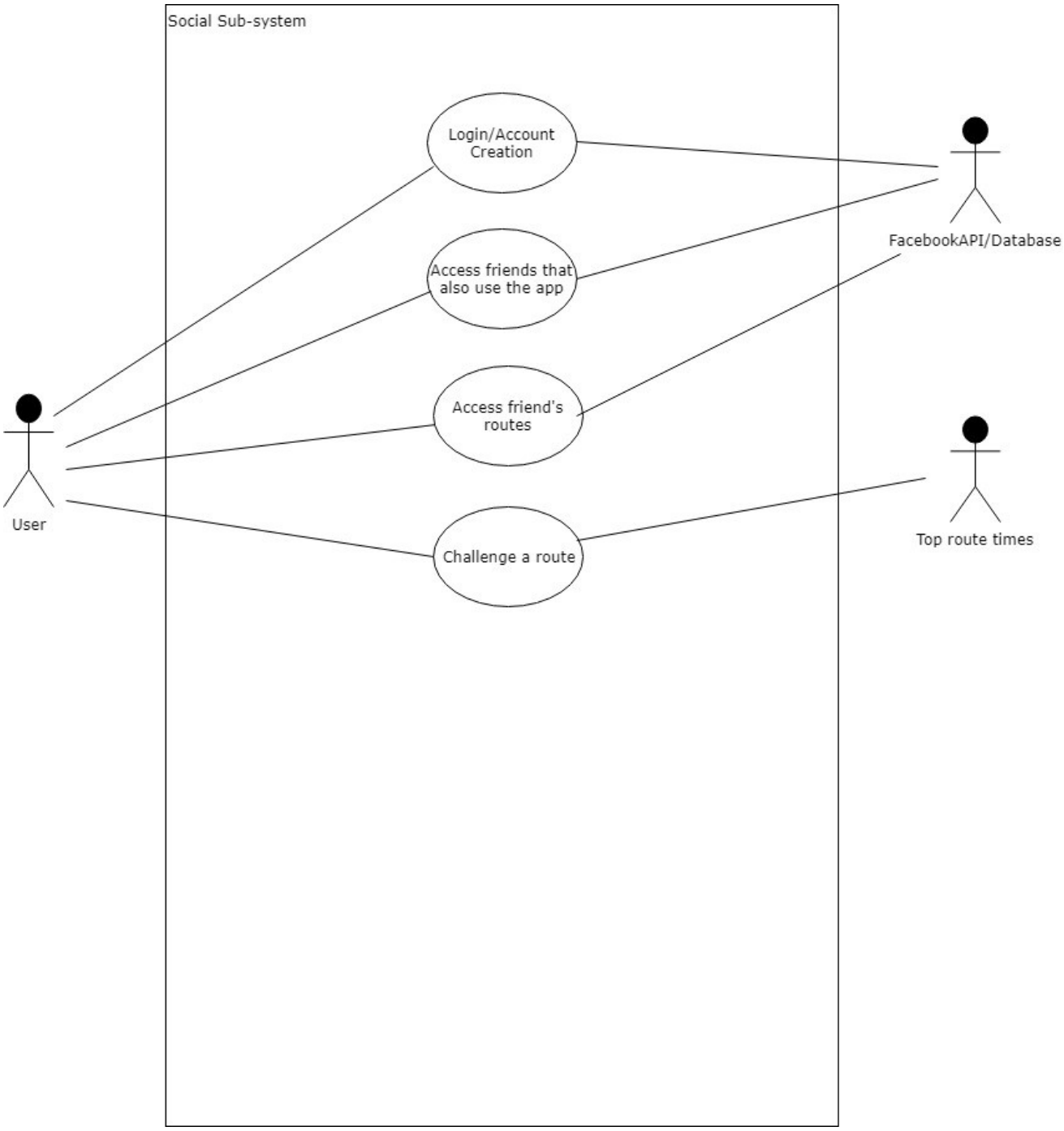


Figure 9: Pathway social use case diagram

4.2.5.2 Use Cases

1. The user creates an account:
   (a) The user wishes to use Pathway and creates an account either through linking their facebook account or by making an account through our system.
2. The user logs in:
   (a) The user has an account and wishes to login, they either enter their credentials or select to login using their facebook.
3. The user wishes to challenge a top time on a route:
   (a) The user wants challenge a user that holds a record time for a route. After selecting a route from a list, the users can display the top times for that route and will have the option to challenge it
4. The user wishes to see friends from Facebook that also have the app:
   (a) The user hits the "Friends" tab, and it loads all of their friends that also have the app installed on their device
5. The user wishes to see the achievements they unlocked:
   (a) The user hits the "Achievements" tab, and it loads all of the application's achievements and displays what achievements the user has unlocked

+

4.2.6 Test Cases

- TC-01: Linking Facebook profile to app

  Requirements Fulfilled:

  The user will be able to link their Facebook account with the app to create an account.

  The app will be able to access the user's Facebook data in order to utilize other social functionalities

  Test Steps:

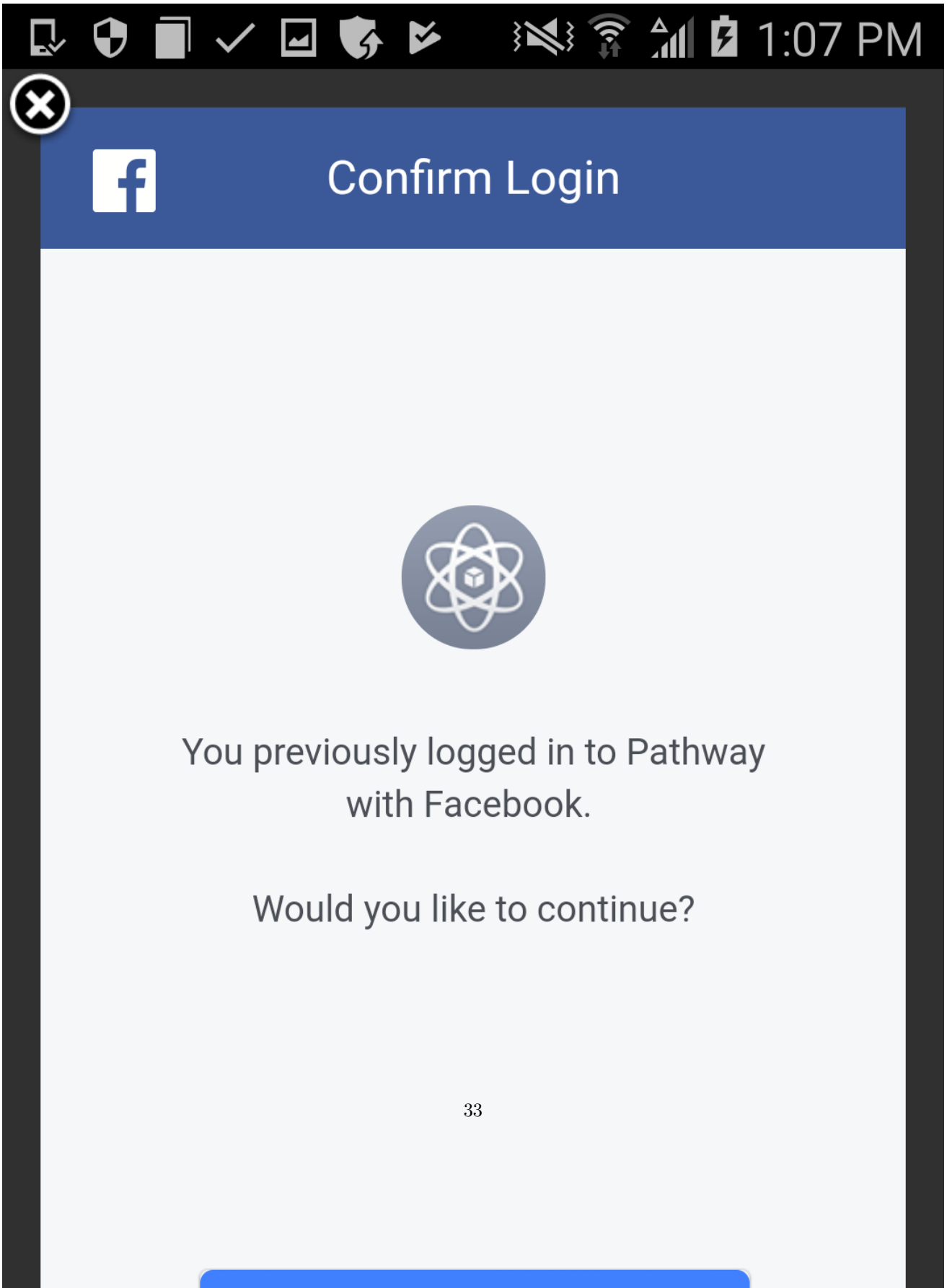
  1. App is started.

  2. Press "Connect with Facebook" button when prompted to login.

  3. Webview of Facebook login page appears where the user will be prompted to enter their account email and password.

  4. User will be directed back to app depending on whether the login was successful or not.

  Expected Result: User is logged in/registered and is added to the database with accessibility to core data such as name, email, and friend's list from Facebook account (other data can be accessed optionally).


1:07 PM
Confirm Login
You previously logged in to Pathway with Facebook.
Would you like to continue?

- TC-02: Viewing Achievements

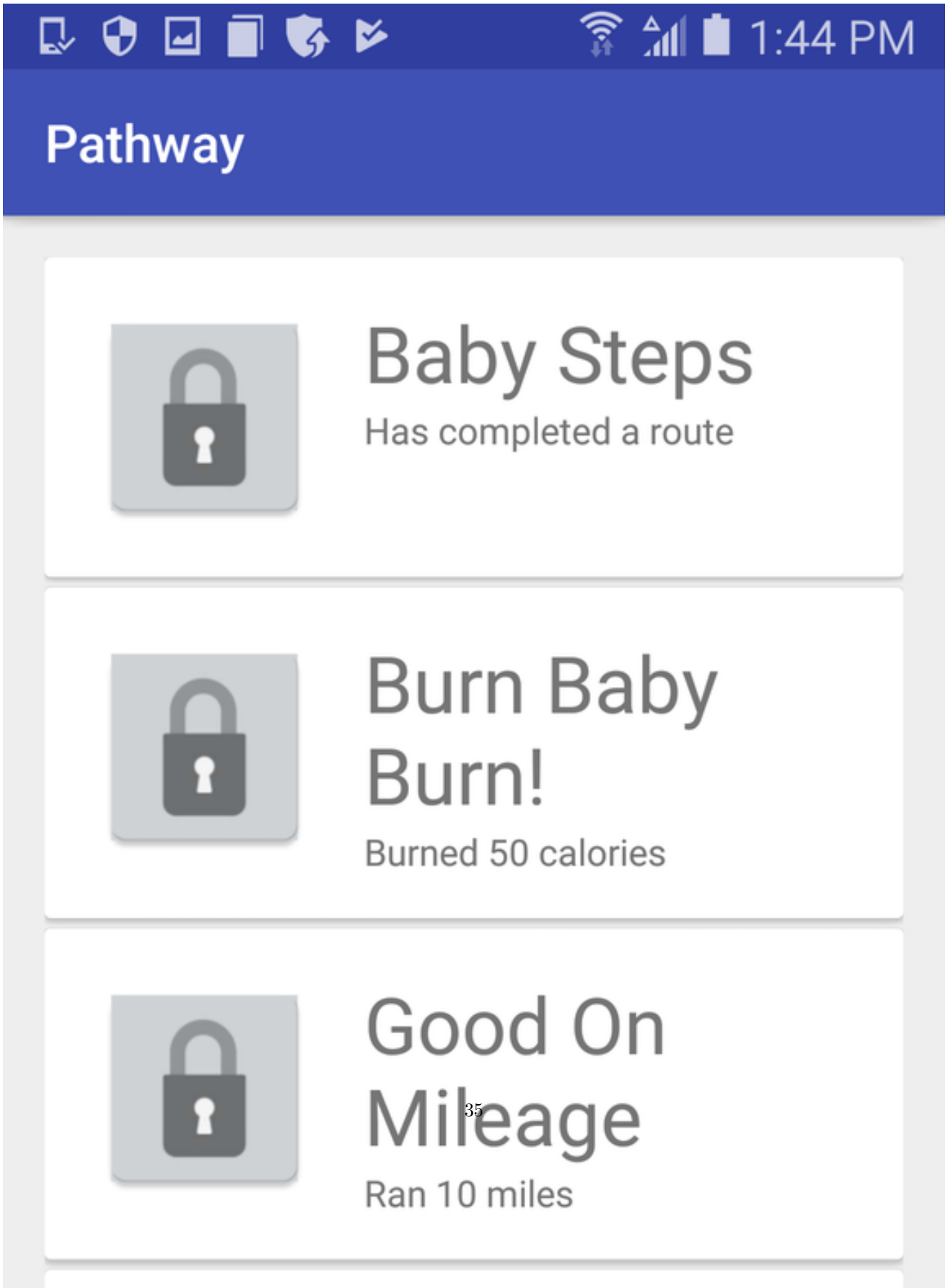
  Requirements Fulfilled:

  The user will be able to see what achievements they have unlocked

  Test Steps:

  1. App is started.
  2. After logging in open the navigation drawer
  3. Tap "Achievements"
  4. User will be directed to the achievements page to see what achievements they have unlocked

  Expected Result: User displays list of achievements which will show if they are unlocked or not


1:44 PM
Pathway
Baby Steps
Has completed a route
Burn Baby Burn!
Burned 50 calories
Good On Mileage
Ran 10 miles

- TC-02: Viewing Friends

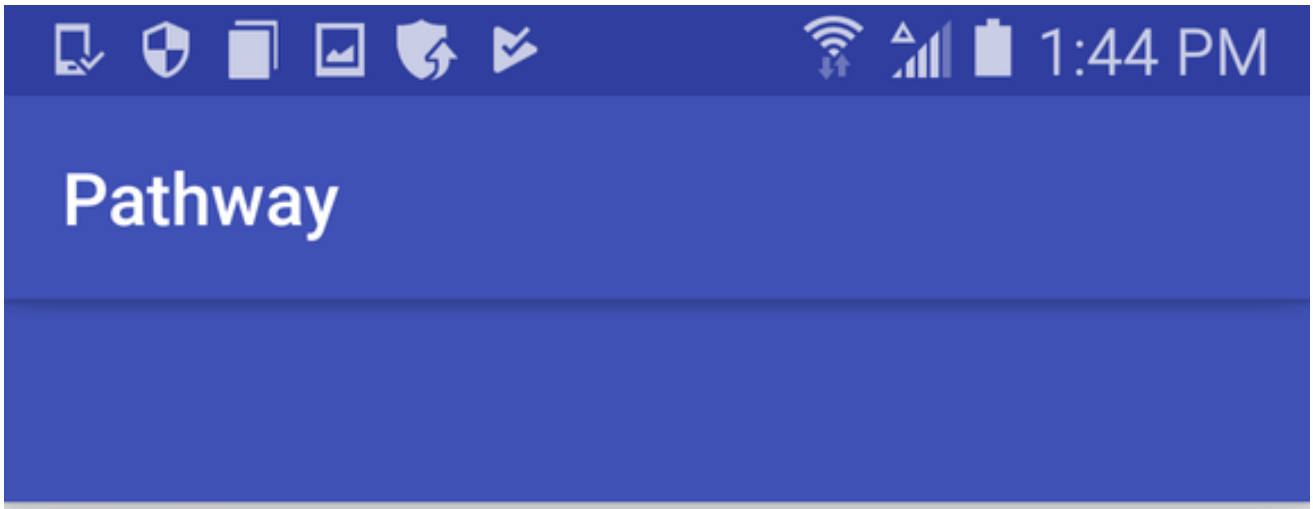
  Requirements Fulfilled:

  The user will be able to see their Facebook friends that also have the app downloaded

  Test Steps:

  1. App is started.
  2. Log in through Facebook
  3. Will be redirected to friends list after the login is complete

  Expected Result: User displays list of Facebook friends who also use the app


1:44 PM
Pathway
Chris Carney
Bob John

4.2.7 Social Subsystem Code

4.2.7.1 Facebook Login Code

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import com.facebook.AccessToken;
import com.facebook.AccessTokenTracker;
import com.facebook.CallbackManager;
import com.facebook.FacebookCallback;
import com.facebook.FacebookException;
import com.facebook.login.LoginResult;
import com.facebook.login.widget.LoginButton;

public class MainActivity extends AppCompatActivity {

LoginButton loginButton;
CallbackManager callbackManager;

protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

//bind UI component to LoginButton variable
loginButton = (LoginButton)findViewById(R.id.login_button);
callbackManager = CallbackManager.Factory.create();

//sets permissions so that data from user's profile can be accessed
loginButton.setReadPermissions("email", "public_profile", "user_friends");

loginButton.registerCallback(callbackManager, new FacebookCallback<LoginResult>(

//When login is successful
@Override
public void onSuccess(LoginResult loginResult) {
//code goes here
}

//When user cancels login
@Override
public void onCancel() {
//code goes here
}
```

```
//When an error occurs during login
@Override
public void onError(FacebookException error) {
//code goes here
}

});

}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
super.onActivityResult(requestCode, resultCode, data);

//callback manager gets response code to determine what next activity is
callbackManager.onActivityResult(requestCode, resultCode, data);
}

}
```

4.2.8 Design Alternatives

For the social subsystem, the original plan was to utilize the Facebook API to set up an achievement system and a challenge function that would just essentially send a Facebook notification to the user being challenged. Even though the API has these utilities, they are not fully accessible to those creating an Android app. The notifications function was only allowed its by a web canvas app and since our project is a native app on an android device, it was not possible to implement.

4.2.8.1 First Design Alternative: Achievements

Seeing as how the Facebook API gives applications in development such a restricted area to make implementations, we decided to create our own achievement system from the ground up. We did this by first creating a local database that stored set achievements with their names and a value that shows the activation of the achievement which means it has been unlocked. We then pull user data from the Statistics subsystem to help set conditions in which the user will unlock certain achievements once the requirements are met for each individual achievement. After the requirements are met, the value in the table for the corresponding achievement also changes to signify that it has been unlocked.

```
/*
code will access a local database that has the achievements stored manually.
Then we create an object of the class that the local database is created in to p
user report data so that we can set if statements which will check and see if re
*/
```

```
if user data meets the condition set{
set achievement to "unlocked"
}
```

4.2.8.2 Second Design Alternative: Challenging

Since the API also allows little room for development when trying to send notifications to users once they are challenged, we decided to take another approach. We agreed upon an implementation that instead of sending the notification to the user through Facebook, we will just use the SMS feature that is provided through Android Studios which is called the SMSManager. In order to utilize this function we had to include imports and permissions that would let us use it. So now when a user challenges someone, the application will make a GET request to the server for the phone number of the person that is being challenged and it will send a preset message notifying them that a user has challenged a route of theirs.

```
/*
called after user chooses route to challenge and request is made to server to re
*/
public void sendText(){
//request to use SMS permission during runtime
requestSmsPermission();
try {

sendSms("User's number", "Your route has been Challenged!");

} catch (Exception e) {
e.printStackTrace();
}

}

private void sendSms(String phoneNumber, String message){
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(phoneNumber, null, message, null, null);
```

4.3 Statistics/Metrics - Eric West

4.3.1 Introduction

The statistics sub-system will perform analysis of user data generated by the Geolocation sub-system and data stored within the Infrastructure sub-system's database. This sub-system will take data generated from the use of the route tracking of the application and analyze it. It will then access the database by methods defined by the Infrastructure sub-system to retrieve any stored results that are related to this analysis compare and compare them to determine the user's performance. Once this is done the data will be organized into a formatted report and delivered to the User Accounts sub-system for reporting the details to the user. It will also allow the user to compare themselves to other users who have similar profile features. This sub-system will also generate any graphs of user data that are required by the user or by other sub-system.

4.3.2 User Requirement

- The system will use collected geospatial data and user demographic information to present analysis of the users performance.
- The system will take user data and generate leaderboards for heavily used routes in certain areas, in order to allow users to compete for bragging rights on those routes.
- The system will generate a report after a run detailing the users performance with comparisons of previous runs.
- The system must generate reports that should be easily readable and detailed while avoiding being confusing to users.

4.3.3 Platform

- The system will be developed using Python 3.
- The system will be a sub-service housed along with the server application.

4.3.4 Domain Analysis

4.3.4.1 Competing Software

The following applications/software may fulfill similar roles or offer similar functionality:

- MyTracks - Offers only basic data collection (Lat/Long position, elevation) dependent on device sensors, and no analysis options.
- FitBit App - Offers data collection and basic analysis of user performance.
- iOS Health - Offers basic data collection and limited analysis of performance.

- LG Health - Offers route tracking and calorie journal. Still lacks any social networking component.

4.3.4.2 Extensions

For later versions of this service we may try to increase the variety of analysis. We may also offer a web service allowing a more complete analysis of the users performance and physical activities.

4.3.4.3 General Domain Knowledge

Few people have the knowledge or desire to manually formulate statistics about their fitness progress.

4.3.4.4 Tasks and Procedures

- The user finishes a route.
- The users device uploads the route details up to the database on the server
- The server calls the statistic subsystem analysis function
- The function will be passed a way to access the data uploaded by the user
- The function will perform the analysis
- Determine average speed, distance, steps, etc.
- Analyze the routes gps points to determine the data points needed to plot a graph of distance over time
- Any extra data analysis involving previous route data
- Prepare information to be sent back to user
- Server sends the analysis data back to the users device
- Device will process the analysis data to generate a report for the user
- The report will display the average speed, steps, etc. for the route
- The devices system will take the data points and plot a graph of distance over time.

4.3.4.5 Clients and Users

This sub-system will be used by any device with the Pathway application and a user account.

4.3.4.6 Similar Domains

There are many programs and applications that offer similar functionality to this sub-system. The nature of the application means that such sub-systems are required to fulfill the purpose of fitness tracking and personal improvement. It will be important to examine these other domains to learn how to improve and leverage more out of the sub-system.

4.3.5 System Analysis

4.3.5.1 Introduction

For the statistics/metrics subsystem, the plan to utilize a python module specifically built to perform the necessary statistical operations, such as numpy. This will be the most appropriate way to run the analysis as such modules are both efficient, fast, and well tested. This reduces the need to develop mathematical and statistical algorithms. This system will also use the JSON module to parse any JSON formatted data that is necessary to perform the calculations required of the system. The Python library Geopy will be used to determine distances for GPS points. On the client side, we will utilize a widely used library called Android Plot to render plots. However the need to develop an efficient approach for handling the reports is necessary. For this reason we plan to implement the Factory method design pattern. This will allow the server to simply pass a small amount of data to an object called a factory which contains methods that generate objects. This factory then gathers the necessary data from the database and performs the various statistical calculations, such as determining averages and standard deviations, and passes this report to the server.

4.3.5.2 Use Cases

Generate Report

1. User Requests report
    (a) App determines type of report to generate
2. Request is sent to server
3. Server processes request
    (a) The userID, routeID, and type of report are collected
    (b) These values are passed to a ReportRequest object
4. ReportRequest is sent to ReportFactory
5. ReportFactory produces report with appropriate fields and plot values
    (a) ReportFactory determines behavior based on ReportRequest
    (b) Requisite data is gathered by ReportFactory

(c) All necessary calculations are performed

(d) ReportFactory passes these values in a corresponding report-type object

6. Server sends report back to user

   (a) Server will audit report to ensure the report contains valid data.

   (b) Sends the report

7. Pathway app processes the report

   (a) Report data is processed and passed into a structure to store and organize.

8. Processed report is displayed to the user.

   (a) Android activity will pull data from storage structure and fill in appropriate GUI components.

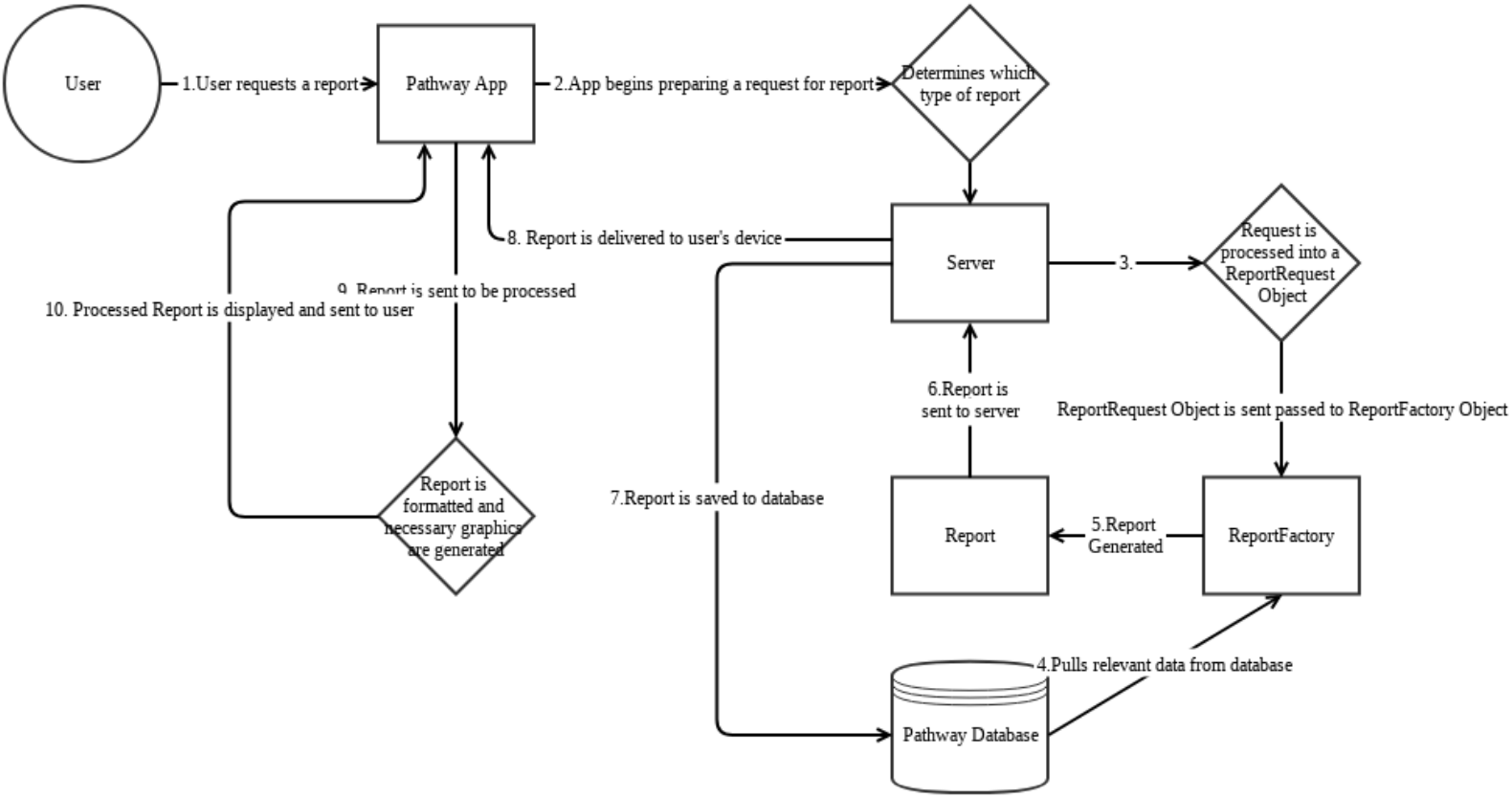


Figure 13: Statistics system data flow

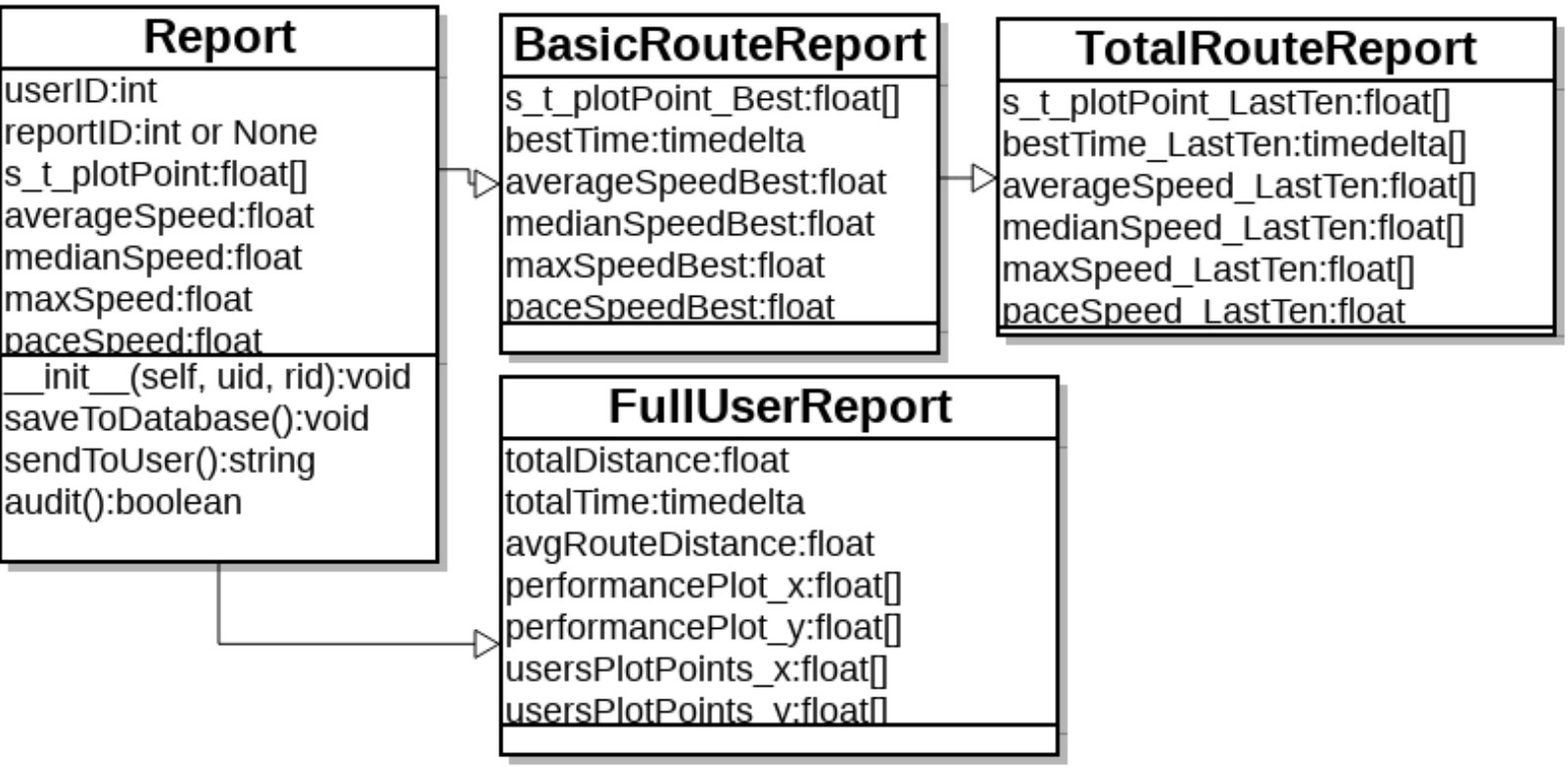


Figure 14: Statistics system UML:Report Classes

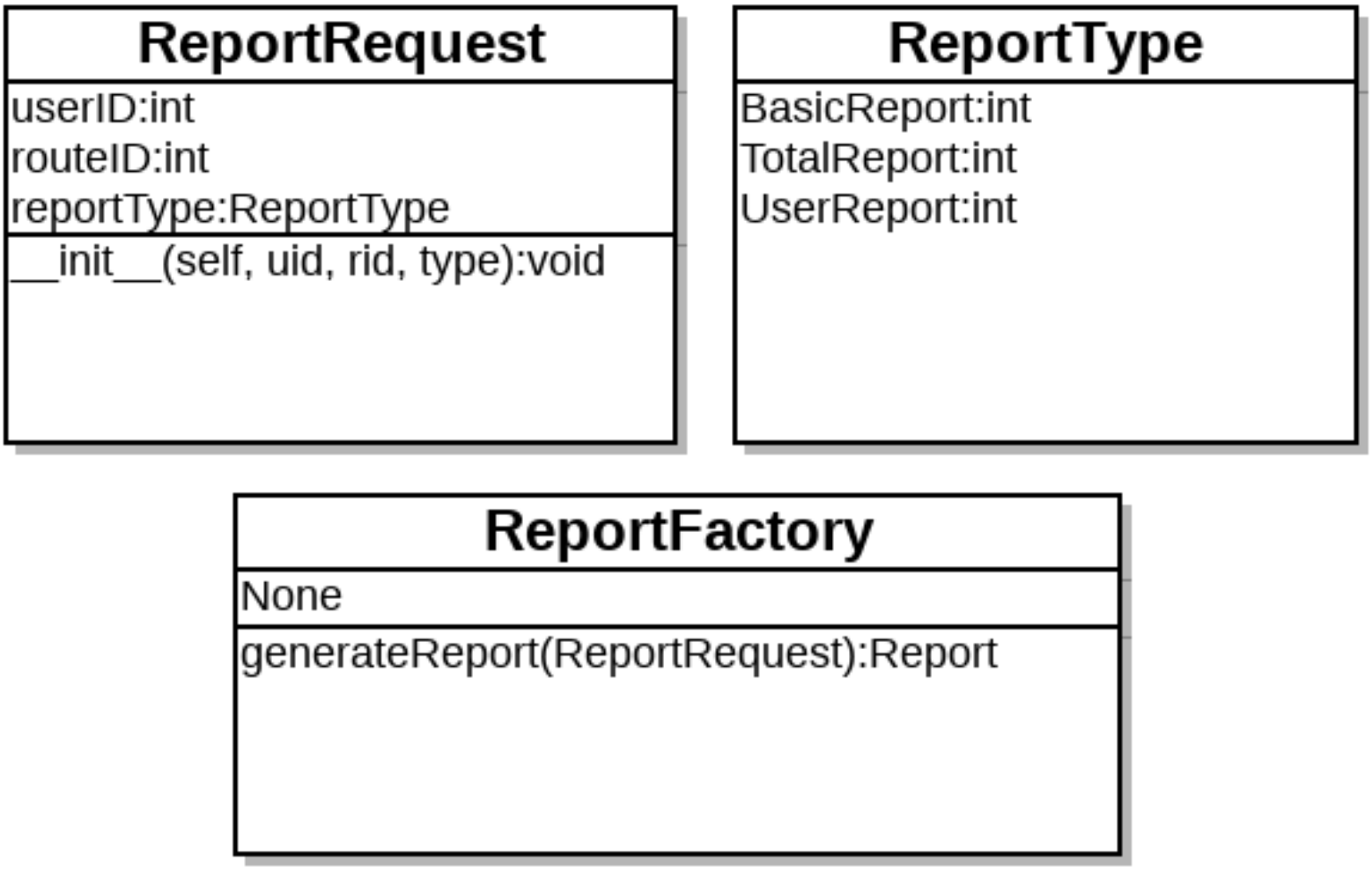


Figure 15: Statistics system UML: Utility Classes

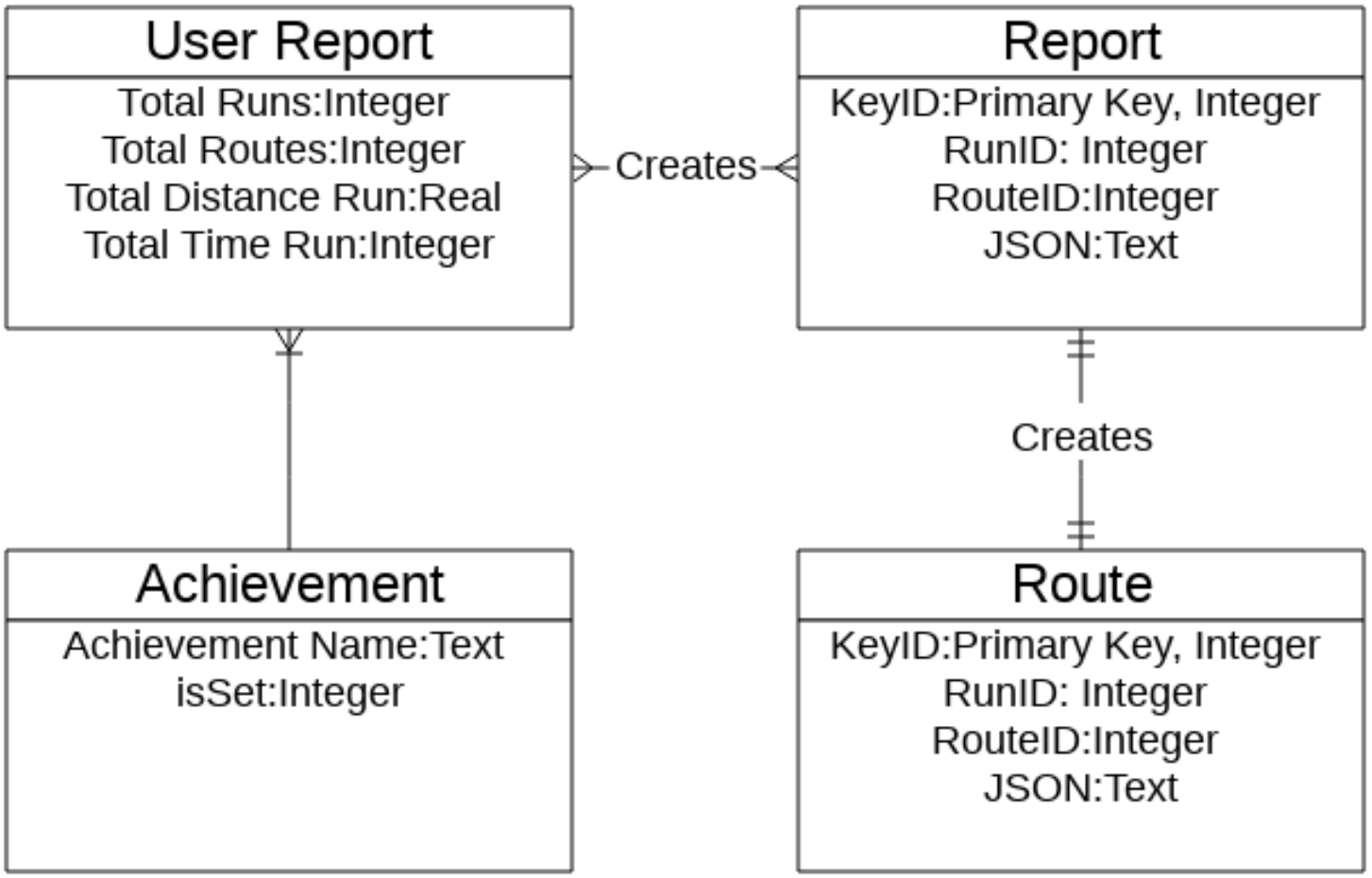


Figure 16: Statistics system ER: Local SQLite Database

4.3.6 Statistics Subsystem Testing

4.3.6.1 Introduction

Our plan for testing the Statistics subsystem has been divided into three separate phases, which are listed and described in more detail in their own subsections below. These have been divided into separate parts based upon the domains involved in each part. This is partly due to the issue of the data be-

ing passed through two different systems, one being a server the other being a mobile device. This also allows for testing to build off of one another.

4.3.6.2 Test Plan Parts

1. Generating Reports
2. Auditing Reports and Delivery
3. Parsing and Presenting Reports

4.3.6.3 Testing Phase: Generating Reports

To test the functionality of this subsystem on the server we have designed three test cases and are currently planning to begin by testing them against using simulated route data to test the functionality these three cases are listed below and have a section that will follow describing them in more detail. After all test cases have passed using simulated data the subsystem will be tested against route data generated by our application using the same tests.

Test Cases

- TC-01 Generate Basic Route Report

  Requirements Fulfilled:

  1. System can generate a basic report by using user generated data
  2. The system will use collected geospatial data and user demographic information to present analysis of the user's performance.
  3. The system will generate a report after a run detailing the user's performance with comparisons of previous runs.

- TC-02 Generate Total Route Report

  Requirements Fulfilled:

  1. System can generate a total report by using user generated data
  2. The system will use collected geospatial data and user demographic information to present analysis of the user's performance.
  3. The system will take user data and generate leaderboards for heavily used routes in certain areas, in order to allow users to compete for bragging rights on those routes.
  4. The system will generate a report after a run detailing the user's performance with comparisons of previous runs.

- TC-03 Generate Full User Report

  Requirements Fulfilled:

1. System can generate a complete user report by using user generated data
2. The system will use collected geospatial data and user demographic information to present analysis of the user's performance.
3. The system will take user data and generate leaderboards for heavily used routes in certain areas, in order to allow users to compete for bragging rights on those routes.
4. The system will generate a report after a run detailing the user's performance with comparisons of previous runs.

TC-01 Generate Basic Route Report This test case will involve simply taking a user route data from their most recent run and their best run of the same route. It will then compare the two after performing statistical analysis to derive some basic performance data from. The resulting analysis will then be used to create an object derived from the base Report class. The Report derived object will then call the audit function to ensure all fields contain valid format and content.

TC-02 Generate Total Route Report This test case will involve simply taking a user route data from their most recent run and all of their previous runs of that route. It will then compare them after performing statistical analysis to derive a more detailed set of performance data. The resulting analysis will then be used to create an object derived from the base Report class. The Report derived object will then call the audit function to ensure all fields contain valid format and content.

TC-03 Generate Full User Report This test case will involve taking all route and usage data for a user and analyzing it to give that user a detailed performance and usage analysis. This analysis will then be stored in a Report derived object which will then be audited to ensure all fields contain valid format and content.

4.3.6.4 Testing Phase: Auditing Reports and Delivery

This next part will be testing the functionality of auditing a report and delivering the report to the user's device. The first part of this test, auditing of the report, is necessary to ensure that the user is being given report data that is valid. Any inconsistencies will result in misleading data or issues with plotting graphs, so it is essential for reports to be complete and contain valid data and types in their respective fields. The second part will be to ensure that the report data can be sent to the user's device in a format that their device will be able to extract all the report data from. As the reports are generated using Python and the language of the device we have been developing our application on executing Java, this will be an important test.

Test Cases

- TC-04 Auditing Reports

  Requirements Fulfilled:

  1. Ensure reports are properly formatted before delivery
  2. Ensure all report fields contain valid data
  3. The system must generate reports that should be easily readable and detailed while avoiding being confusing to users.

- TC-05 Delivering Reports

  Requirements Fulfilled:

  1. Ensure reports objects can be converted to Java objects
  2. Ensures that report can be delivered to the user's device
  3. The system must generate reports that should be easily readable and detailed while avoiding being confusing to users.
  4. The system will generate a report after a run detailing the user's performance with comparisons of previous runs.

TC-04 Auditing Reports This test case will involve taking reports that have been generated by the system and auditing them. The test will involve taking reports and formatting them into a appropriate data type, likely a string, for delivery to our user's device. Then it will then audit the report by checking to make sure that all the fields that were in the Report object are in the object. We will begin the testing process by attempting this by using valid reports, reports with all their fields filled, then on invalid reports, reports missing one or more fields or has invalid fields.

TC-05 Delivering Reports This part of this testing phase will involve taking the data type we generated and converting it to a Java equivalent object. Our first test will begin by reading in files containing the report data from storage on the device and converting them into their equivalent Java objects. We will then move to testing when report data is delivered by sending them through the server to client devices.

4.3.6.5 Testing Phase: Presenting Report Data

This final phase of testing will involve taking the user reports and processing them for presentation to the user. This will involve generating graphs and listing the report data for the user.

Test Cases

- TC-06 Presenting Report Data

  Requirements Fulfilled:

  1. Display report data to the user
  2. Give user the visual aids and an interface to interact with report
  3. The system must generate reports that should be easily readable and detailed while avoiding being confusing to users.
  4. The system will generate a report after a run detailing the user's performance with comparisons of previous runs.

4.3.7 Statistics Subsystem Code

4.3.7.1 ReportFactory Code

This is a snippet below is of the ReportFactory class which will be used to generate reports for a user when delivered a ReportRequest object. This class is the factory portion of our implementation of the factory method design pattern. It will generate a report of the requested type at runtime for a user.

```
import report as rep;
import json;
import numpy as np;
import statistics as stats;
from geopy.distance import vincenty;

class ReportFactory:
    def generate_report(self, request):
        if request.report_type == rep.report_type.basic_report:
            return self.doBasicReport(request.userid, request.routeid);
        elif report_request.report_type == rep.report_type.total_report:
            return self.doTotalReport(request.userid, request.routeid);
        elif report_request.report_type == rep.report_type.full_report:
            return self.doFullReport(request.userid, request.routeid);
        return None;

    def doTotalReport(self, uid, rid):
        #code here...

    def doBasicReport(self, uid, rid):
        #code here...

    def doFullReport(self, uid, rid):
        #code here...
```

4.3.7.2 Report Class Code

This is code snippet below is for the Python Report class. It is the base class from which all of our report types inherit from.

```
class report:
    def __init__(self, uid, rid, s_t_plots, pace_speed, avgSpd,
    medSpd, maxSpd, currTime, calories_burned):
        self.userid = uid;
        self.routeid = rid;
        self.s_t_plotsx = s_t_plots;
        self.pace_speed = pace_speed;
        self.avgSpeed = avgSpd;
        self.medSpeed = medSpd;
        self.maxSpeed = maxSpd;

    def audit(self):
        return true;

    def sendToDatabase(self):
        return "";

    def sendToUser(self):
        return str(self.userid) + "," + str(self.routeid) + "," + ...;
```

4.3.8 System Conversion and Implementation

In the process of implementing the Statistics subsystem there was some changes to the system. One of these was to implement the report factory that was the primary part of the system on the android device rather than on the server. This change resulted in the subsystem having to be coded in Java and utilize several features built in to the android system libraries.

It has also resulted in the addition of another feature to this subsystem, managing the local database on the android device. It was originally part of the Geolocation subsystem until the redesign of this system required access to it. When it was migrated to the Statistics subsystem it was significantly expanded and additional features were added to handle the demands of both Statistics and the Geolocation subsystems as well as allow better integration with the other subsystems. The additional features required for the Statistics subsystem were to store basic and user reports. The Social and User subsystems required access to these reports to implement some of the features of offered by them.

4.3.9 Design Alternatives

We considered two different designs for this subsystem. One approach, the original, was to design and implement this subsystem on the server using Python.

The second approach, was to implement the subsystem on the android device using Java.

4.3.9.1 First Design Alternative

The original approach was to integrate the subsystem on the server to be called when data for a route was being saved to the server's database. The subsystem would have called during the saving to create a report and then save the report to the server's database as well. The report would be saved for future requests. The subsystem was going to be implemented using a combination of the Factory Method and the Builder design patterns.

4.3.9.2 Second Design Alternative

The second approach, which we decided to implement, was to implement the system on the android device. The system was implemented in very much the same manner as the original, however the reports were generated and stored on the android device's local SQLite database. This approach offered easier access and better security for the User's route data. It also made it easier for the Statistics subsystem to be intergrated with the other subsystem.

4.4 Geolocation Collection and Plotting - Johnny Bragg

This sub-system, referred to as the Geolocation sub-system, will manage the collection, plotting, and storage of geospatial data that will comprise routes traversed by the user. This data will include: WGS 1984 Latitude and Longitude coordinates, expressed in decimal degrees, Elevation as measured in measures, 0 relative time-stamp information in a 24Hr hh:mm:ss format, and additional derived attributes from the aforementioned data, such as speed, direction, etc.

The sub-system will collect data and write it into a standardized JSON format called GeoJSON[1] for use in the statistics sub-system and storage in the database. Route data will also be accessible by other systems as needed whenever a mapped route needs to be displayed. Additionally, the sub-system will employ two main UI fragments. The first UI fragment will allow a user to track their path as it appears on a web service provided background map in approximately real-time. The second UI will provide a more data oriented synopsis of the user's current speed, elevation, and coordinates, and will be accessed using a slide up panel visible during data collection. User's will have the option of exporting data in a text file for personal use. Other information may be included as the project is revised.

The Geolocation sub-system will transmit data to the database back-end for storage and store a local copy of the user's route data on a SQLite database instance on the device to ensure functionality is retained when the User loses service. The DeviceDBHandler class will handle setup and interaction with the database on the device and will synchronize with the external database, by tracking time stamps, when needed. During development, it was determined

that the statistics subsystem would benefit from local storage capabilities, so the device database was expanded to accommodate the needs of the statistics subsystem, as well as added tables for Users and Achievements. Please see the statistics subsystem for details regarding implementation of the local database system.

Sending and Receiving of Route information To and From the database server is accomplished by extending Java's AsyncTask[2] class to two new classes: FetchData and SendData. Both classes make use of an interface and Callback methods to send and retrieve data. An implementation of FetchData was found online and modified to handle POST requests as well as GETs.[3] The main map activity will implement interfaces for both classes to avoid blocking the UI during requests and while the application awaits a response from the server.

4.4.1 User Requirements

- System will use device sensors to record GPS data in a Latitude/Longitude format.
- System will use device sensors to record user elevation or device altitude.
- System will plot Geolocation data against a Google provided background map.
- System will record relative time stamps to use in calculation of speed and direction.
- System will display user activity duration, speed, distance, current elevation, and coordinates during data collection.
- System will display user routes clearly and in real-time as data is being collected.
- System will allow the user to save individual routes to a file on the device for personal use.
- System will display traversed routes in the User's immediate vicinity.
- System will allow a user to tap on a route to retrieve and view information associated with that route.
- System will store a user's routes in a particular format for use in Pathway.
- System will maintain a local device database to store a user's route data.
- System will be available as long as a persistent data connection is allowed.
- System will be made available in a limited capacity with reduced accuracy and no map service while no data connection exists.
- System will accurately and precisely locate the user's position within an acceptable margin of error, allowing for a larger margin where service is weak or none exists.

4.4.2 Process

The system will be completed an tested over the course of the Fall 2017 semester, with regular updates to follow.

4.4.3 Domain Analysis

4.4.3.1 Introduction

This document conveys a brief overview of GPS based data plotting and collection as is commonly used in mapping applications. This is a well-established domain, with many existing applications employing mapping systems to some degree.

4.4.3.2 Extensions

Possible extensions may include extension of data review to a website interface. Functionality beyond the mapping capabilities is unnecessary and will be constrained to only those updates required to maintain functionality as updates occur to the API or updates to correct any bugs discovered in operation.

4.4.3.3 General Domain Knowledge

- Most mobile devices are GPS capable.
- Most mobile devices include sensors to count steps and track speed.
- Many people who walk/run for fitness maintain some records of times/speeds to track progress.
- Fewer people maintain information regarding where they perform fitness activities outside of a gym or similar facility.
- Generally, fitness programs are more successful if personal improvement is apparent. (Times noticeably decrease, weight is lost, strength measurably improves, etc.)
- Fitness journals and programs that monitor progress help people stick with a program.

4.4.3.4 Clients and Users

The primary clients of the mapping sub-system will be the other sub-systems comprising Pathway, who will depend on the mapping sub-system to collect and store the data used in those sub-systems. Users of Pathway will interact with the Geolocation sub-system to initiate data collection, describe routes, and save completed routes.

4.4.3.5 The Environment

All actors in this process will need access to a GPS enabled tablet or cell phone running Android v5 or later. The final product will require an active data connection to write and retrieve data from the database and to retrieve base-map information from Google servers. Offline functionality will be limited to viewing of User owned routes saved on the device.

4.4.3.6 Tasks and Procedures

- Finding user location using GPS: The sensors perform an initial ping of satellite locations to establish a start location.
- GPS accuracy is limited by device and location constraints. Specifically, mountainous or very built-up urban areas may produce signal "bouncing", causing the GPS to lose accuracy.[4]
- Subsequent positions are found by pinging the sensors every x seconds.
- GPS location may be supplemented using triangulation through wi-fi networks and cell towers.
- Recording the current user position: The information collected by the device sensors is in the form of Lat/Long coordinates and an elevation value, typically measured in feet.
- Storing user location over a period of time: User information (Lat/Long, Elev, etc.) is stored in a text file or data structure to allow review at a later date.
- Additional functions may be performed using the data collected in the above methods. This may include: Plotting of data in real-time.
- Calculation of derived information that may be useful to the user. This includes: speed, direction.

4.4.3.7 Competing Software

Our research has discovered that many applications incorporate mapping information into their functionality. Due to the widespread usefulness of the system, the general nature of the software, and the fact that the API used is offered free of charge, it can be argued that there are no direct competitors. Similar software products exist but are marketed on their utility either outside of mapping or by some unique feature in tandem with mapping.

4.4.3.8 Domain Similarities

Mapping applications like the one we envision are common, and the usefulness of geographic data is apparent in many fields and industries. There is no shortage

of similar systems available for use, so documentation is both plentiful and current. Due to the general nature of this sub-system, we are not concerned that any similarities would constitute a duplication of an existing system. The widespread usage of similar systems does however provide us with a wealth of information to be used towards development of our own system.

4.4.4 System Analysis

4.4.4.1 Introduction

The geolocation subsystem is relatively isolated from other subsystem within the Pathway project. The primary interaction between the geolocation subsystem and other subsystems will be in the passing of an instance of the Route class, which will extend functionality of the JSONObject datatype. The mapping functionality of the subsystem will be implemented by leveraging existing classes and objects within the Google Maps API and as such will not require extensive coding to achieve the desired functionality and layout. This functionality would include map display, panning, zoom, and drawing of features on the map.

4.4.4.2 Design Analysis: Algorithms

To implement the route comparison feature, we have reviewed three algorithms that break the problem down into a comparison of ordered sets of pairs: Hausdorff distance and Fréchet distance, and a third algorithm that we have devised that will provide a simple comparison measure.

4.4.4.3 Simple Line Comparison

We have created a basic route comparison algorithm that will run first, saving computation time. The simple version of the algorithm takes two Routes as input and recursively compares start and end points of line segments within the route, limiting the calls to five recursions, in a fashion similar to how a binary search is performed, beginning with the total route.
Note: n is the last element in the Route.

```
match = false
if counter >= 5 then
    match = true
end
else if len(Route1) == 0 OR len(Route2) == 0 then
    return false
end
else if len(Route1) == 1 AND len(Route2) == 1 then
    if start Route1 near start Route2 AND end Route1 near end Route2
     then
        match = true
    end
end
else if start Route1 near start Route2 AND
end Route1 near end Route2 then
    counter ←counter + 1
    k ←floor(n/2)
    match = (basicCompare(Route1[0 : k], Route2[0 : k], counter) AND
    basicCompare(Route1[k + 1 : n], Route2[k + 1 : n], counter))
end
return match
```

Algorithm 1: basicCompare(Route 1, Route 2, counter)

The above algorithm uses a counter to limit the number of times a recursive call is made, and therefore runs in O(1) time. Some possible drawbacks to the above algorithm are that two lines that intersect at key places but are otherwise very different may be considered a match. The Basic Line Comparison algorithm is unlikely to be useful for long or complex routes, but will serve as a good first comparison for less complex features.

4.4.4.4 Hausdorff Distance

The Hausdorff distance, named after Felix Hausdorff, measures how far two subsets of a metric are from each other. [5] The Hausdorff distance compares sets of points, and as such, is unaffected by the ordering of the sets. This makes it especially suitable for use in Pathway to determine if one traversed Route is the same as another, regardless of starting point along that Route. The following pseudocode was adapted from an online source.[6]

```
h ← 0
for point a_i in A do
    shortest ← ∞
    for point b_j in B do
        d_ij ← d(a_i, b_j)
        if d_ij shortest then
            shortest ← d_ij
        end
    end
    if shortest h then
        h ← shortest
    end
end
```

Algorithm 2: Hausdorff distance for ordered sets of coordinate pairs.

Hausdorff distance is the best candidate for our purposes, as it treats lines as sets of points with no particular order.It is also worth noting that H(A,B) != H(B,A), necessarily.[6] The Hausdorff algorithm shown here clearly runs in O(mn) time. While the run time of the algorithm is expensive, the simplicity of the implementation and the relatively small data sets that will be compared make the run time an acceptable trade-off.

4.4.4.5 Fréchet Distance

The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves.[7] It is more complex and harder to implement, but generally more versatile and supports backtracking. Unlike Hausdorff distance, Fréchet distance does consider ordering of points of when performing comparisons.

dF(P,Q): real;
Data: Polygonal Curves P = (u_1,...,u_p) and Q = (u_1,...,u_q)
Result: dist(P,Q)
ca: array[1..p,1..q] of real;
c(i,j): real;
for *i=1 to p* **do**
  for *j=1 to q* **do**
    ca(i,j) $\leftarrow -1.0$**end**
  end
  if $ca(i,j) > -1$ **then**
    return ca(i,j)
  end
  else if $i = 1$ *AND* $j = 1$ **then**
    ca(i,j) $\leftarrow d(u_1, v_1)$
  end
  else if $i > 1$ *AND* $j = 1$ **then**
    ca(i,j) $\leftarrow max(c(i-1,1), d(u_1, v_1))$
  end
  else if $i = 1$ *AND* $j > 1$ **then**
    ca(i,j) $\leftarrow max(c(1,j-1), d(u_1, v_1))$
  end
  else if $i > 1$ *AND* $j > 1$ **then**
    ca(i,j) $\leftarrow max(min(c(i-1,j), c(i-1,j-1), c(i,j-1)), d(u_1, v_1))$
  end

Algorithm 3: Fréchet distance for two polygonal curves.

The above implementation of the discrete Fréchet distance, as described by Eiter and Mannila[8], runs in O(pq) for initial assignment and O(pq) for calculation of ca(p,q) for a total of O(pq) time.

4.4.4.6 Algorithms Testing

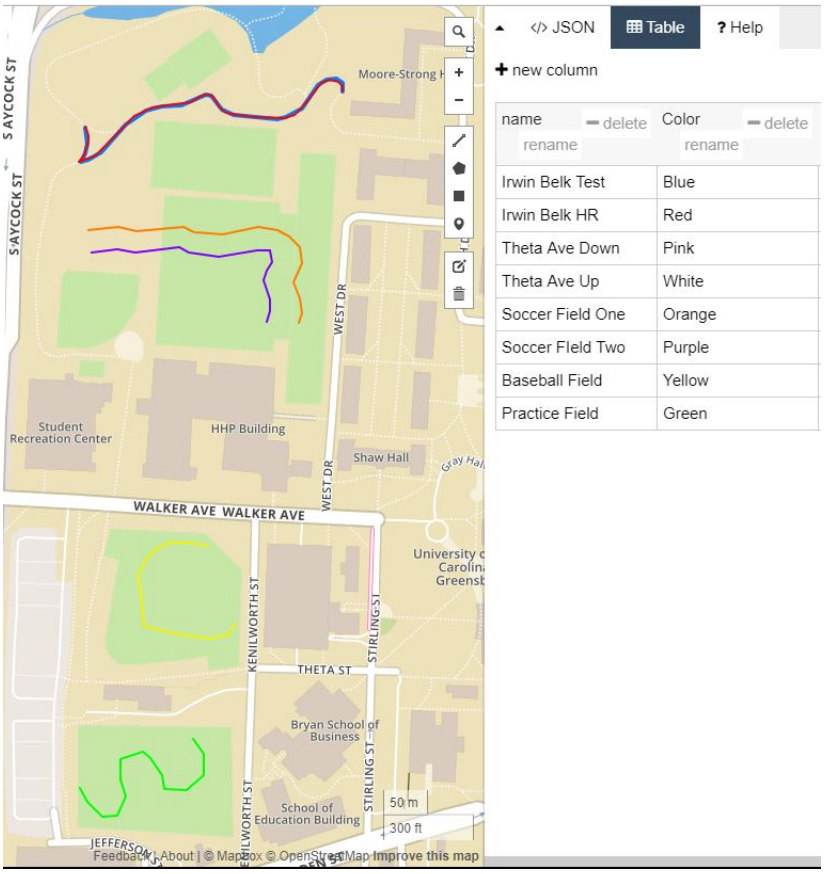


Figure 17: Test Routes For Comparison Algorithms

The previously mentioned algorithms were tested against the above routes. Results are summarized in table form on the next page. Cells marked in grey represent where Routes were compared to themselves and are trivially true. Comparisons were performed with a 25m tolerance for comparison acceptance of points. Hausdorff and Fréchet algorithms return a distance value, which was then accepted if the distance was within the 25m tolerance.

Simple	Blue	Red	Pink	White	Orange	Purple	Yellow	Green
Blue	TRUE	TRUE						
Red	TRUE	TRUE						
Pink			TRUE	FALSE				
White			FALSE	TRUE				
Orange					TRUE	TRUE		
Purple					TRUE	TRUE		
Yellow							TRUE	FALSE
Green							FALSE	TRUE

Figure 18: Results for Simple Comparison

Hausdorff	Blue	Red	Pink	White	Orange	Purple	Yellow	Green
Blue	TRUE	TRUE						
Red	TRUE	TRUE						
Pink			TRUE	TRUE				
White			TRUE	TRUE				
Orange					TRUE	TRUE		
Purple					TRUE	TRUE		
Yellow							TRUE	FALSE
Green							FALSE	TRUE

Figure 19: Results for Hausdorff Comparison

Frechet	Blue	Red	Pink	White	Orange	Purple	Yellow	Green
Blue	TRUE	TRUE						
Red	TRUE	TRUE						
Pink			TRUE	FALSE				
White			FALSE	TRUE				
Orange					TRUE	TRUE		
Purple					TRUE	TRUE		
Yellow							TRUE	FALSE
Green							FALSE	TRUE

Figure 20: Results for Fréchet Comparison

4.4.4.7 Algorithm Results Summary

The Simple Comparison works as expected, correctly identifying similar routes Red and Blue and Routes Orange and Purple as similar. It also correctly marked Routes Green and Yellow as not matching. Routes Pink and White however were incorrectly marked as not matching. One obvious shortcoming of the simple comparison is that it adheres to a strict ordering of pairs, and so would not match a route to the reverse of itself.

The Hausdorff comparison works as expected in all cases, correctly matching Red and Blue, Orange and Purple, and Pink and White as matching, and Yellow and Green as not matching. Further, the Hausdorff algorithm correctly identifies route similarity in cases where Routes are reversed. This most closely matches the needed functionality for Pathway's purposes.

The Frechet comparison does a better job matching less linear features correctly, but also pays attention to point order within Routes. This gives it the same shortcoming as the Simple Comparison algorithm. While still useful, this shortcoming makes it less than ideal for use in Pathway.

4.4.4.8 Tool Selection

Due to the reliance of the geolocation system on an existing API, our tool selection will include the Google Maps API[9], Google Elevation API[10], and code provided by the Google Maps Utility Library[11], and the development tools and language that were selected for the project as a whole. The system will be coded entirely in Java, primarily using the Google Maps API and leveraging Android Studio to more easily create the application layout. A SQLite database will be created on the local device to store information until a connection can be established with the Pathway server. The local database will closely mimic the structure of the external server. The above mentioned algorithms will be coded as methods within the Route class, which will store all representations of route data. Currently, the geolocation system design will employ at least 5 classes, with additional classes likely being added in the course of development. These classes may be altered or replaced as research into the API continues.

4.4.4.9 Use Cases

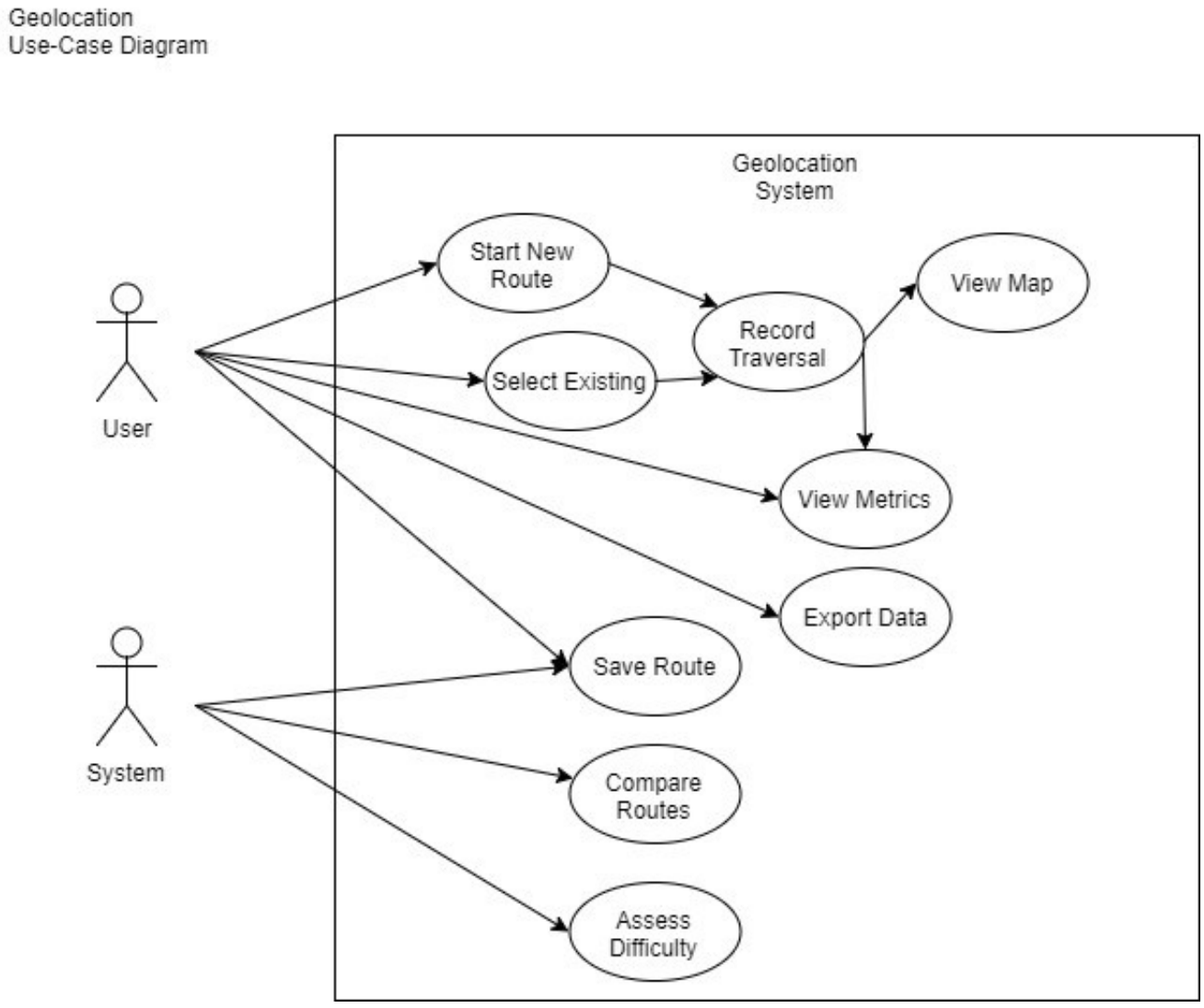


Figure 21: Geographic system use case diagram

1. User starts a new route.
    (a) Actors: User
    (b) Preconditions: System is idle and no existing route is selected.
    (c) Flow:
        i. User presses start route.
        ii. System prompts user to select activity.
        iii. Route begins recording.
    (d) Postcondition: User is recording a route.
2. User starts an existing route.
    (a) Actors: User
    (b) Preconditions: User has selected an existing route from the map and is currently near the start of the route.
    (c) Flow:
        i. User presses start existing route.
        ii. Route begins recording.
    (d) Postcondition: User is recording a new traversal of existing route
3. User completes and saves a traversal.
    (a) Actors: User and System.
    (b) Preconditions: User is currently recording a traversal.
    (c) Flow One: New Route.
        i. User presses Stop and is prompted to name and save.
        ii. System stores Route information in database.
        iii. System returns to idle.
    (d) Flow Two: Existing Route.
        i. User presses Stop and is prompted to name and save.
        ii. System verifies that traversal matches the selected Route.
        iii. System stores Route information in database.
        iv. System returns to idle.
    (e) Postconditions: System is idle.
4. User wants to view track information.
    (a) Actors: User and System.
    (b) Preconditions: Two Cases
        i. User has selected an existing map and is not currently recording.
        ii. User is currently recording a new or existing.

(c) Flow:

i. User presses the info button.

ii. A new view fragment appears to display past and/or current user performance.

iii. User returns to map screen.

(d) Postconditions: None

5. User wants to export data.

(a) Actors: User, System

(b) Preconditions: User has selected an existing or completed a new or existing route.

(c) Flow:

i. User selects Export.

ii. System prompts user for file destination.

iii. System saves csv onto local device.

(d) Postconditions: A file exists on the local device.

6. System wants to compare two routes.

(a) Actors: System

(b) Preconditions:

i. Routes have overlapping bounding boxes.

ii. Routes have similar total distances.

(c) Flow:

i. System checks for preconditions to be true before continuing.

ii. System calculates a similarity score using two routes as input.

iii. System decides if routes are similar based on score.

iv. If similar, store route information using ID of original track.

v. If not, store new route information.

(d) Postconditions: Route is stored in database.

7. System wants to determine difficulty of Route.

(a) Actors: System

(b) Preconditions: Route has been traversed and has not yet been saved.

(c) Flow:

i. System uses a naive method to assign difficulty rating, based on distance.

ii. System uses a naive method to assign difficulty rating, based on length of route and elevation variability, which will be defined as the difference between maximum and minimum elevations.

(d) Postconditions: Route has an estimated difficulty assigned.

4.4.4.10 Classes

Classes, their interaction, and the general flow of the subsystem are depicted in the following diagrams.

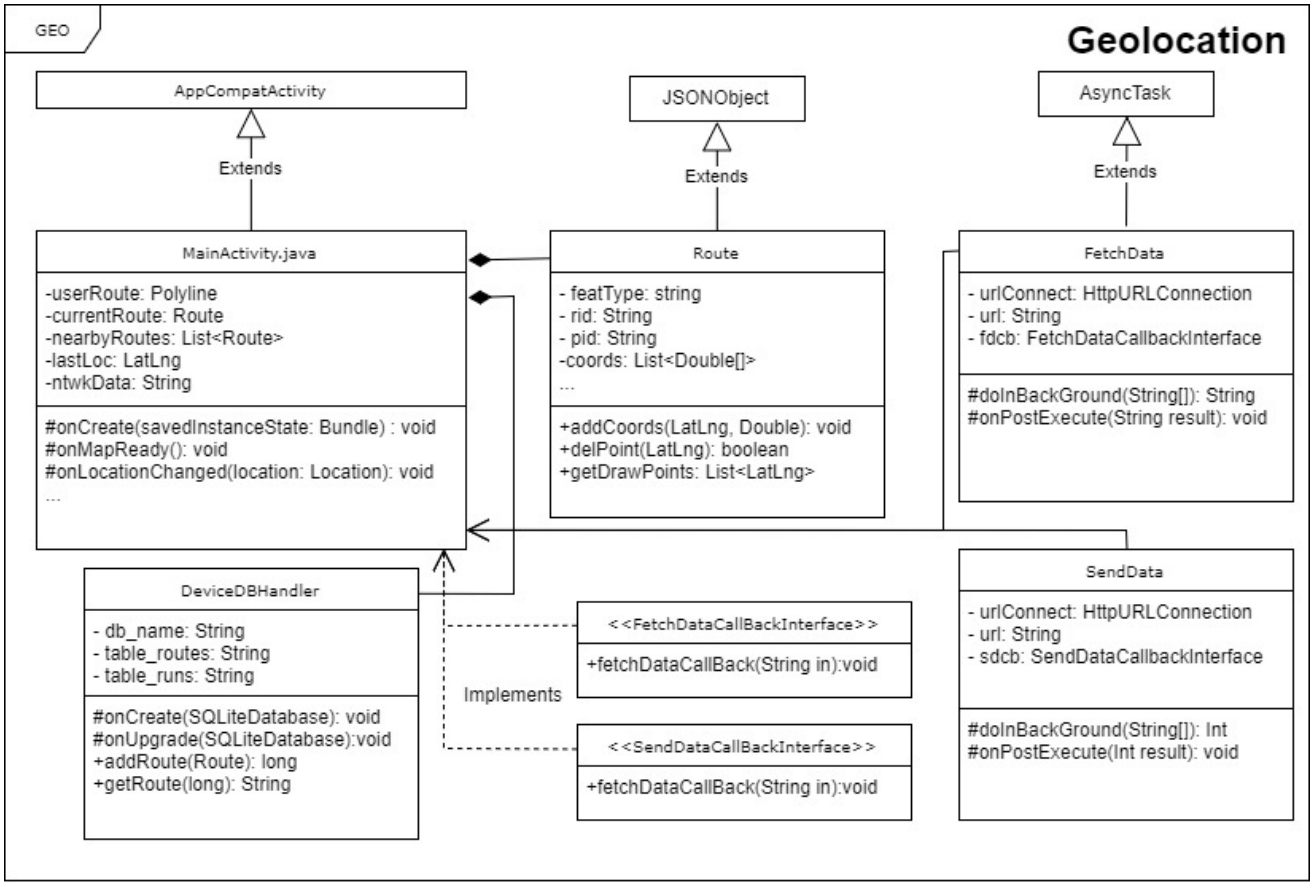


Figure 22: Geolocation UML diagram

The system will be comprised of 5 classes, aside from strictly UI classes. Classes marked with * are classes from the Maps API. The MainActivity will maintain instances of the Location* class and GoogleMap* class, which will be used to retrieve user location and display the map, respectively. The Route class will handle storage and retrieval of information in conjunction with Google's Polyline class. Each Route object will contain identifying information, location data, and additional attributes as necessary. Routes will be retrieved from the database based on current geographic location. Route will inherit from the JSONObject datatype and will follow geojson conventions as described by Butler et al [1], allowing it to store geolocation information in a standard format.

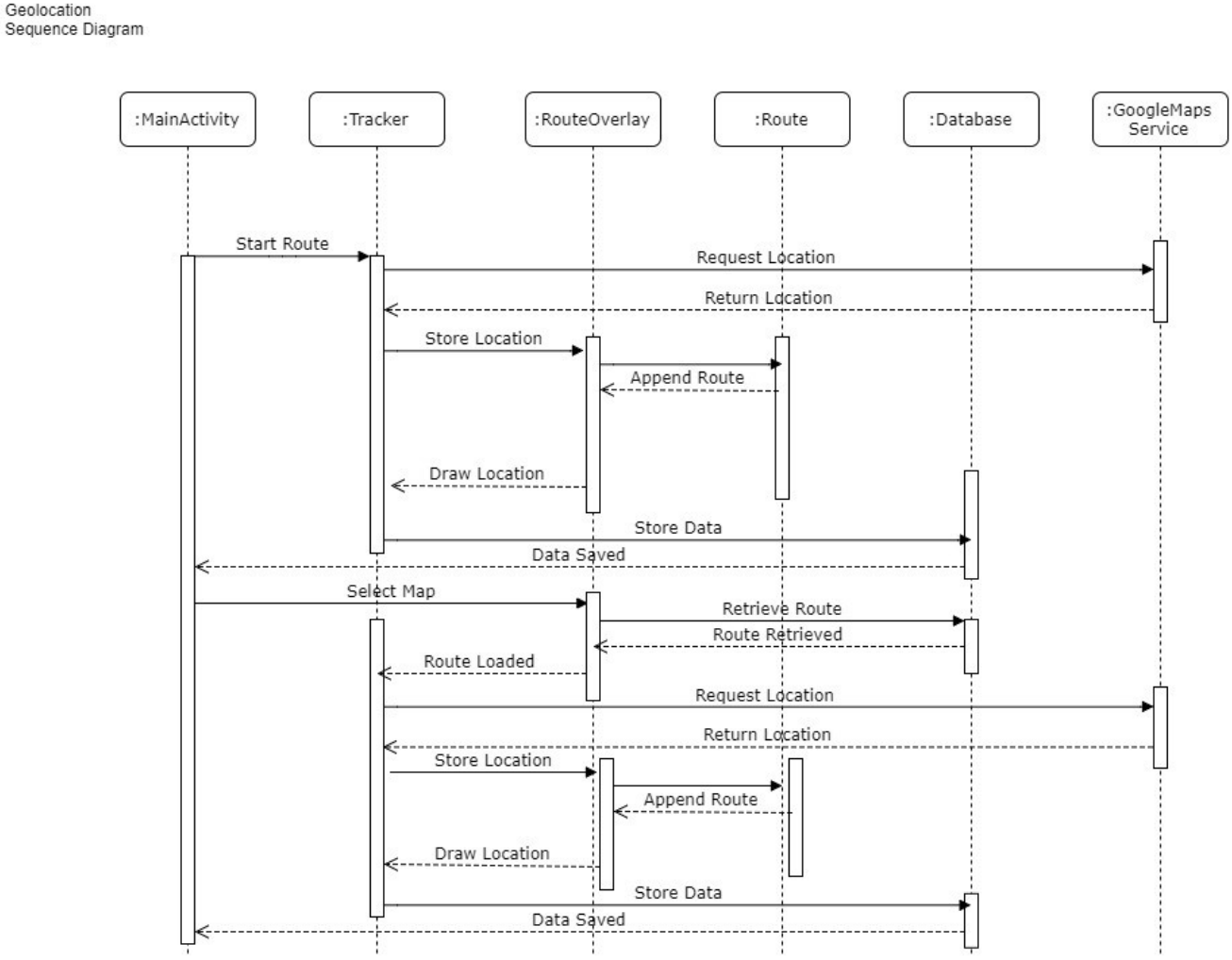


Figure 23: Geolocation sequence diagram

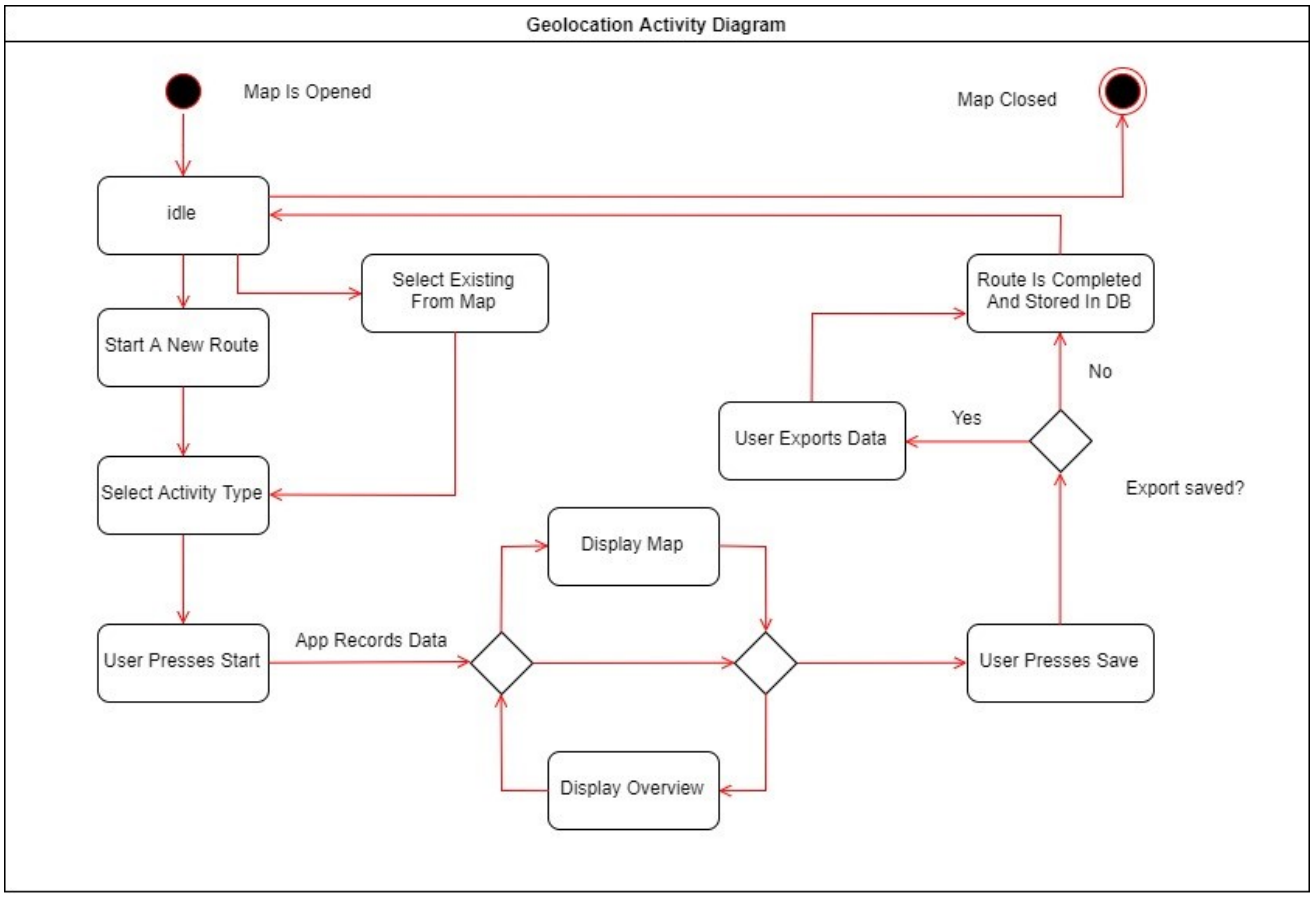


Figure 24: Geolocation activity diagram

4.5 Testing Plan

4.5.1 Test Overview

In order to test the geolocation functionality of our application, we have derived a series of test cases, corresponding to specific requirements in the User Requirements section of the Geolocation subsystem. Tests were performed on an emulated Nexus 5 running Android 5.0 and on an LG G3 device running Android 6.0. The primary goal of the tests were to ensure correct operation of GPS sensors and utilization of device location data. The following tests do not comprise an exhaustive list of all tests to be performed and do not satisfy all User Requirements of the Subsystem. Additional tests will be performed as development continues to satisfy all remaining requirements. Test Cases:

- TC-01 - Get Device Location
- TC-02 - Draw User Route
- TC-03 - Functionality With No Data Access

The following terms are used in describing these tests:

- Camera - Refers to the overhead view within the map frame.
- MyLocation - Standard Map API control to query for device location.

4.5.2 Test Cases

- TC-01: Get Device Location

  Requirement(s) Fulfilled:

  System will use device sensors to record GPS data in a Latitude and Longitude format.

  System will plot Geolocation data against a Google provided background map.

  Description: The device retrieves the current location and centers the map on that position. For the test, the map begins centered on a marker placed at Sydney Australia. Upon pressing the My Location button in the upper corner, the map is recentered to the current location.

  Precondition: User has logged into Pathway.

  Test Steps:

  1. Application is started.
  2. Press MyLocation Button
  3. Map pans to user.
  3. Observe Results Of Location Change.

  Expected Result: Current Location of the User is shown.

Actual Result: Map view pans to current location, as shown in Figures 17 and 18.

Additionally, manually moving camera and clicking MyLocation successfully recenters view on the User's location.

Related Code:

```
public void onMapReady(GoogleMap googleMap) {
    // Add a marker in Sydney, Australia,
    // and move the map's camera to the same location.
    mMap = googleMap;
    LatLng sydney = new LatLng(-33.852, 151.211);
    googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
    googleMap.addMarker(new MarkerOptions().position(sydney)
    .title("Marker in Sydney"));
    googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_FINE_LOCATION)
    != PackageManager.PERMISSION_GRANTED) {

    ActivityCompat.requestPermissions(this,
        new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
    return;
    }
    googleMap.setOnMyLocationButtonClickListener(this);
    googleMap.setMyLocationEnabled(true);
}
```

- TC-02: Draw User Route

  Requirement(s) Fulfilled:

  System will plot Geolocation data against a Google provided background map.

  System will display user routes clearly and in real-time as data is being collected.

  Description: Draws the users route on the map as a path is traversed, updating the current position at regular intervals.

  Precondition: User's location has been found, and the Start Button has been pressed. GPS must be enabled.

  Test Steps:

  1. User presses Start
  2. User begins traversing an area.

  Expected Result: Route is drawn on screen.

  Actual Result: Route is successfully drawn on screen. Figures 18 and 19 show traversals in a localized area (roughly 100 ft) and across approximately 4 miles, respectively.

Related Code: Code fires when Start button is pressed.

```
public void onStartPressed(View v) {
    if (runState == RunStates.OFF) {
        btnStart.setText("Stop");
        Snackbar.make(v, "Recording...", Snackbar.LENGTH_LONG)
        .setAction("Action", null).show();

        runState = RunStates.RUN;
        PolylineOptions routeOptions = new PolylineOptions()
        .color(Color.RED)
        .width(12)
        .startCap(new RoundCap())
        .endCap(new RoundCap());
        timerRoute.setBase(SystemClock.elapsedRealtime());
        timerRoute.start();
        userRoute = mMap.addPolyline(routeOptions);
        startLocationUpdates();
    }
    else if(runState == RunStates.RUN) {
        btnStart.setText("Start");
        Snackbar.make(v, "Recording Stopped.", Snackbar.LENGTH_LONG)
        .setAction("Action", null).show();

        runState = RunStates.OFF;
        userRoute.remove();
        userRoute = null;
        timerRoute.stop();
        stopLocationUpdates();
    }
}
```

Code responsible for adding most recent location to Polyline and displaying on screen.

```
public void onLocationChanged(Location location) {
    lastLoc = new LatLng(
    location.getLatitude(),
    location.getLongitude()
    );
    if (userRoute != null) {
        List<LatLng> points = userRoute.getPoints();
        points.add(lastLoc);
        userRoute.setPoints(points);
    }
    coordMsg = String.format("XY: %s.", lastLoc.toString());
    mMap.animateCamera(CameraUpdateFactory.newLatLng(lastLoc));
    mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(lastLoc, 16));
}
```

- TC-03: Functionality With No Data Access

Requirement(s) Fulfilled:

System will be available as long as a persistent data connection is allowed.

System will be made available in a limited capacity with reduced accuracy and no map service while no data connection exists.

Description: The system will make use of GPS sensors when a data connection isn't available. In the event of sudden data loss, the system continues to track and draw User location.

Precondition: System has a GPS signal and Route is being recorded.

Test Steps:

1. User presses Start.

2. System begins recording with no data connection, or data connection is lost during recording.

Expected Result: Application continues collecting GPS data and plotting route. Background map will not be available while data is unavailable.

Actual Result: Route continued to record while data was unavailable (Figures 19 and 20). Quality of data suffered, and was noisy when the User was stationary, due to resolution of GPS sensor, which produced false movement along the route. (Figure 21).

Related Code: No specific code was needed to ensure this functionality.

4.5.3 Test Case Figures

Figure 25: Initial Location - Sydney, Australia

Figure 26: Current Location - Includes depiction of test route.

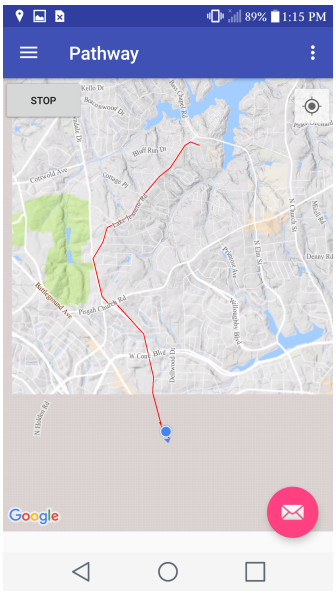


Figure 27: Map showing traversal through area after data loss.

Figure 28: Traversed same area several times to show variance along route.

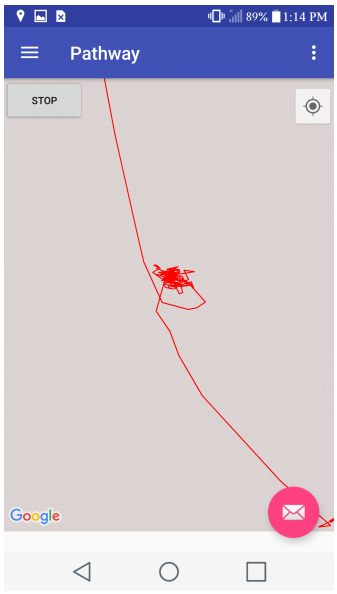


Figure 29: "Bunched" data, collected while stationary without data connection. Shows GPS variance.

4.5.4 Test Summary

Tests performed to ensure accuracy of geolocation on a handheld device were largely successful. Latitude and Longitude values were within expected deviations from actual values as provided by Google Maps each time. While generally accurate, situations where data connection is lost produce less than ideal coordinate values and as such shall be avoided when possible. Prolonged loss of data connection would likely produce unusable data sets, but temporary outages should be able to be handled by the system. B

4.6 User Accounts - Daniel Cregan

This Sub-System focuses on user accounts and the aspects that go along with such a sub-system, such as security for the user and the storing of that users info for their use/consumption in the moment or for later. The sub-system itself will store all tracked and analyzed data from the statistical analysis sub-system and present the information to the user through their account page on the application when they move to their profile page via the GUI. The User's account will display such things as their average walk, run, bike, etc. speed as well as their best time amongst all their recorded routes and types of exercises. It will also display to them total amount of steps taken in lifetime use of the application, and it will also show them a graph of their improvement over their time using the application on either certain routes or just overall, and lastly it will also store all of their recorded routes that they wish to save on a routes page in their profile so that they may view specific statistics about their performance on a course or so that they may publish their route to their friends either using facebook or by using a native groups feature(this part is undecided as of yet as to which well use to publish route data and issue challenges to those in your area).

4.6.1 User Requirements

- The System will allow the user to create an account that will be kept up to date by the system.
- The system for user accounts will keep track of a user's life time statistics and display them in an always up to date state, statistics include, life time steps, distance, best course, best time, etc.
- The user accounts system will save all user routes as buttons that will produce a popup view of the selected route.
- The system will allow the user to pull up these saved routes and see their best time or if that time has been beaten by anyone that they have shared it with.
- The system will also create graphs that display the users progression over time of running a route, that user can see.
- The system will be designed to accommodate future updates and additional functionality as needed.
- The system will be available for constant use, except during a small as of yet undetermined timeframe to allow for system maintenance and updates.
- The system will be stable, minimizing application crashes, freezes, or hang-ups.
- The system must be energy efficient with respect to the mobile device.

- The system must be efficient with respect with to cellular data consumption.
- The system must generate reports that should be easily readable and detailed while avoiding being confusing to users.

4.6.2 Domain Analysis

4.6.2.1 Extensions

- Increased social networking functionality.
- More detailed performance analytics.
- A companion website for profile management, advanced performance display options and tools.
- Additional functionality to allow a user to track nutrition intake and plan meals.

4.6.2.2 General Domain Knowledge

- Users that have used other services will generally choose this one over others because of its much more robust security protocols.
- The analysis of the user data will provide a good frame of reference for the user to see how to improve through the use of graphs.
- Users will be able to quickly access saved routes through their profile and see their statistics for each route.
- Routes will be protected unless a user authorizes the sharing of their route through security protocols.

4.6.2.3 Clients and Users

Target users would include all individuals that: 1. Are capable of operating a compatible device that this application could be installed on, and 2. Are interested in exercising and making connections. 3. Are interested in detailed analysis of their progress in their attempt to exercise.

4.6.2.4 Computing Environment

Primary functionality will be coded in Android 5.0 (Lollipop) and as such will be unsupported on older versions of Android. Specific system requirements are not yet determined. No additional client requirements will be necessary for database access, other than an active connection to internet.

4.6.2.5 Tasks and Procedures

- The general process for traversing user profiles and viewing all info there is and activating a route to run.
- User selects their account in the upper left corner of the application.

  Upon selecting this if an active connection is available it will take the user to their main profile page which contains all of their personal info which can be edited here as well.
- User selects their saved routes page in the tabs at the top center of their profile main page or statistics page.
- The users saved routes page opens if a connection is active and displays the user's routes to the user for them to select and start running, walking, etc. or view their specific stats on them.
- They can also share them by selecting to after selecting the specific route.

4.6.2.6 Similar Domains

Many applications are similar to Pathways user accounts in the sense that they make use of security protocols to keep users safe, and have analyzed data to show users their performance on the application. While we plan to make our User accounts much more robust than those others. Our system would definitely benefit from looking into other applications such as Facebook and other fitness tracking apps.

4.6.3 System Analysis

For the user accounts sub-system, since it relies heavily on the database and the statistical sub-systems for its information that it will display to the user. The matter of analysis will not be so much on the algorithms that will be implemented, but more on what tools through Android 5(Java) will be used/more useful. As I mentioned before the only real real decision algorithm wise will be the decision on what encryption algorithm will be used to encrypt user credentials for storage in the database. For the sake of this project I wanted to select an algorithm that was both secure and used widely throughout the field of security as a way of preparing myself for the field at large, and also wanted an algorithm that would be able to be easily and timely implemented in the time remaining for the project. In light of all of these necessities I have chosen to use HTTP/HTTPS for the handling of the encryption of messages to the server.

4.6.3.1 Use Cases

1. The user creates an account:

(a) The user wishes to use Pathway and creates an account either through linking their facebook account or by making an account through our system.

2. The user logs in:

   (a) The user has an account and wishes to login, they either enter their credentials or select to login using their linked facebook.

3. The user wishes to see their saved routes:

   (a) The user hits the routes tab, and it loads all their saved routes from the local system and displayed to the user.

4. The user wishes to view one of the saved routes:

   (a) The system pulls the route forward, and shows all relevant information to the user.

5. The user wishes to edit their account info:

   (a) The system loads an editable user profile and when done the user hits save this new info is then sent to the database and changed.

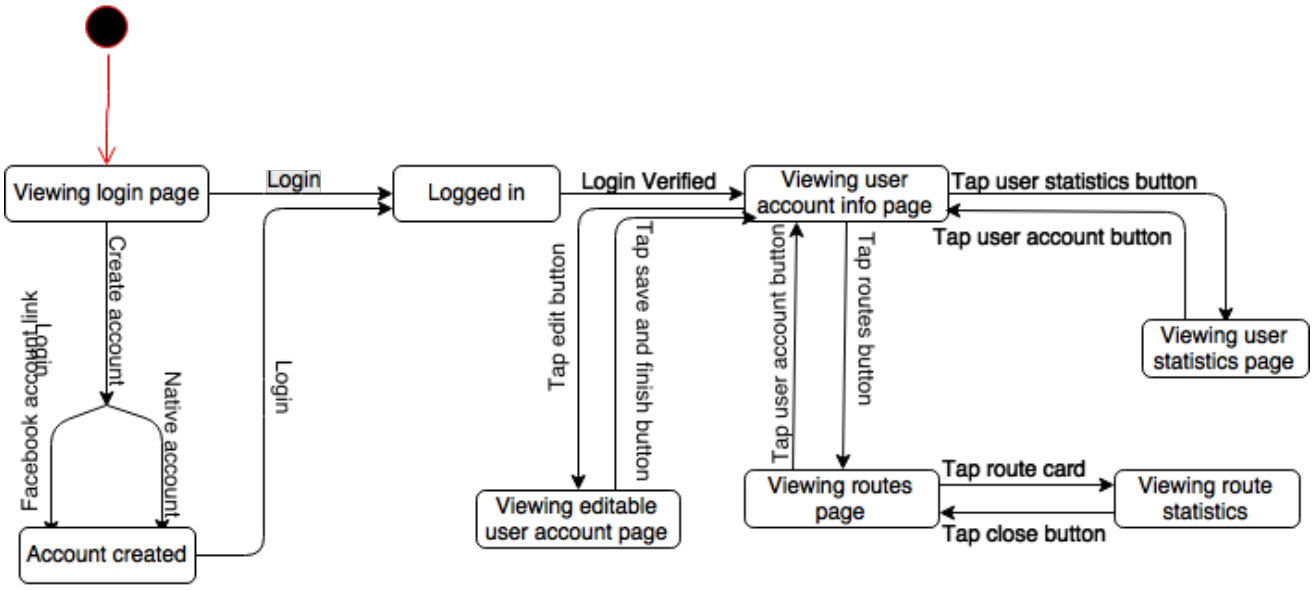


Figure 30: User account system activity diagram

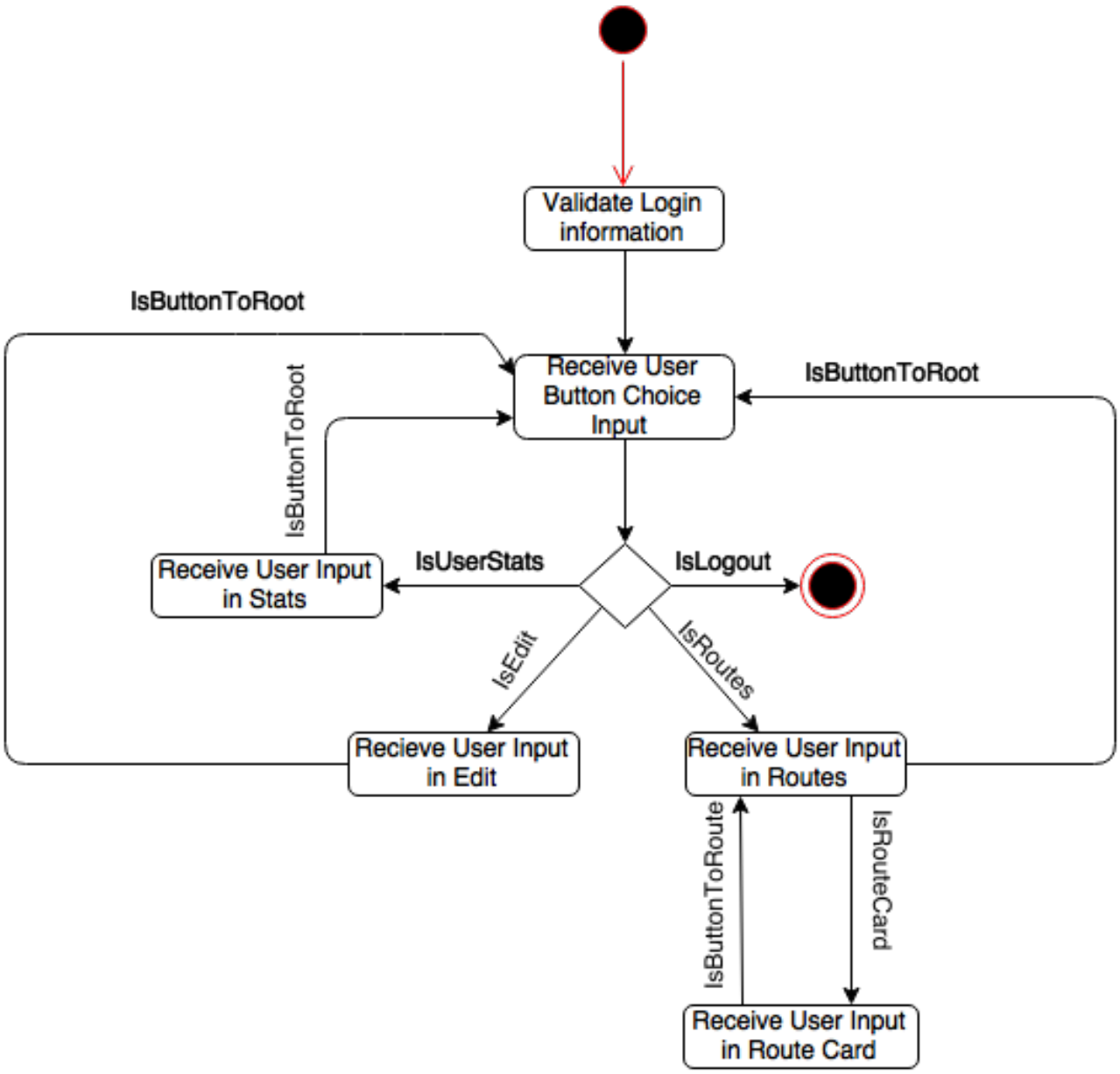


Figure 31: User account dataflow diagram

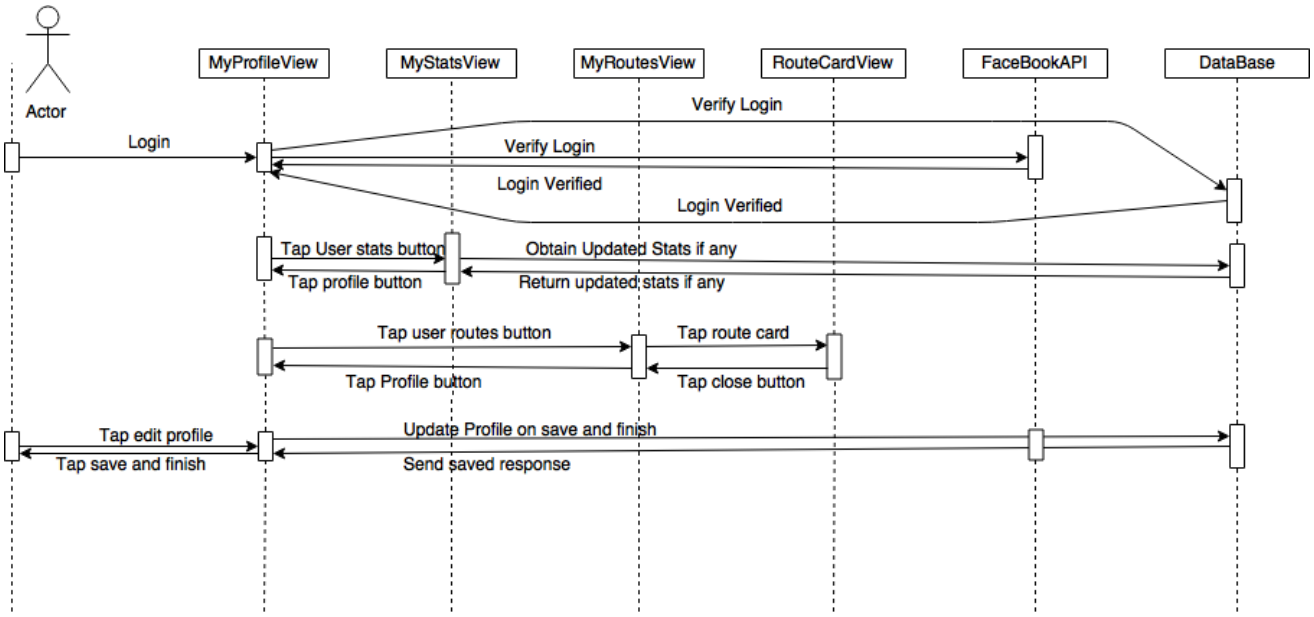


Figure 32: User account sequence diagram

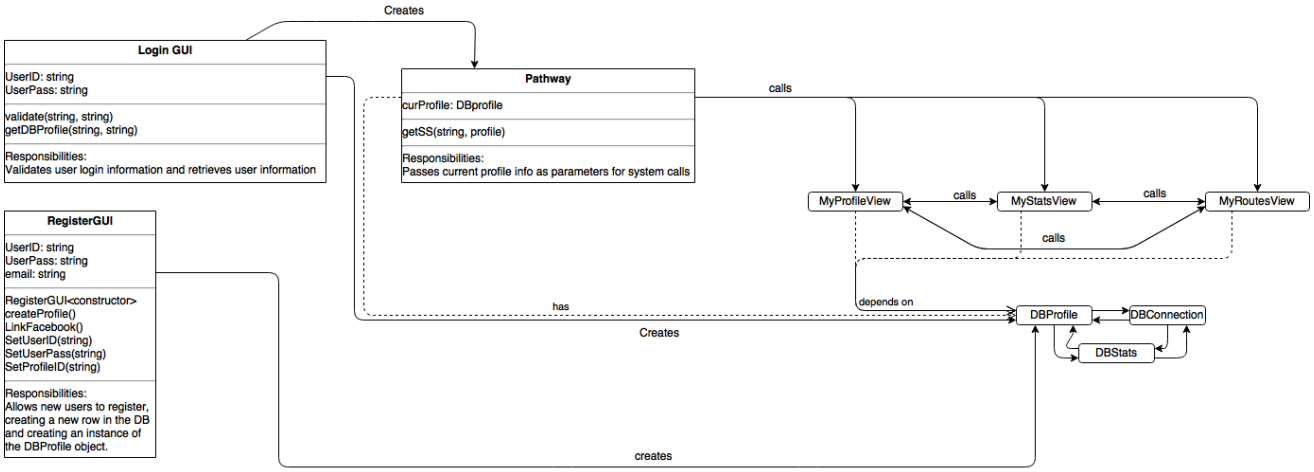


Figure 33: User account UML diagram

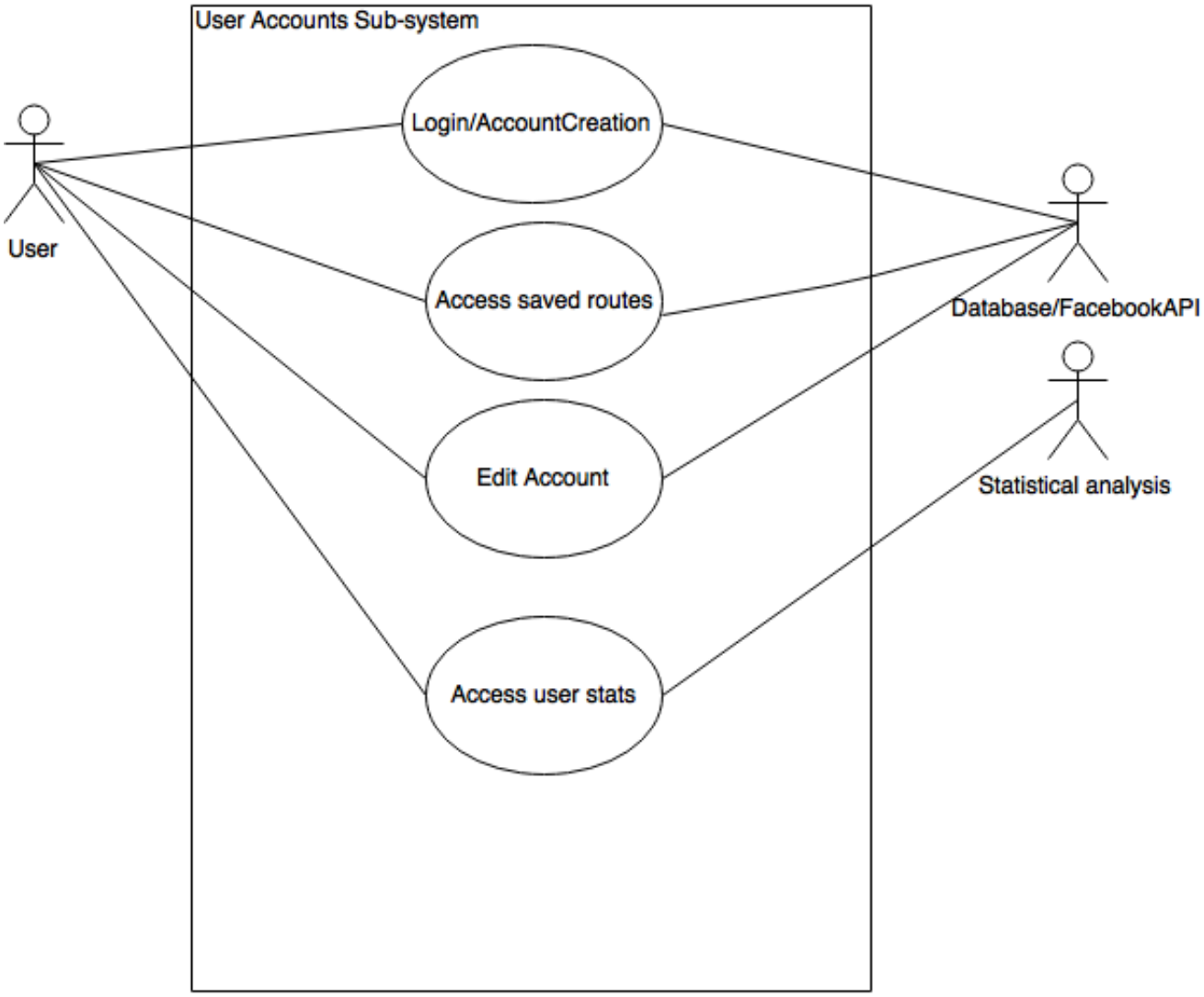


Figure 34: User account use case diagram

4.6.4 Test Cases

- TC-01: Login/Create account

  Requirements Fulfilled:

  System has implemented UI for login screen and account creation screen and will be connected to singleton authentication method

  Singleton authentication system still needs to be linked to database and account creation/login

  Test Steps:

  1. App is started.

  2. Press login after filling valid fields, or press the register link to register then fill the valid fields there and press the register button.

  3. Server checks the database for user specified by login and returns an accept message to the user or if new user registers their info into the database then at which point they can then log in.

  Expected Result: User is logged in/registered and is added to the database.

  Actual Result: The User is logged in/registered and is added to the database.

- TC-02: User account

  Requirements Fulfilled:

  User info, and routes page UI all exist.

  User info page will display pertinent user account info and stats.

  Routes page has code for route pop up selection will connect to mapping sub-system to allow user to save routes to it as buttons that produce pop ups.

  Test Steps:

  1. App starts.

  2. User logs in/registers.

  3. User is shown account info and stats on the account page.

  4. User selects routes page and can tap a saved route veiw a pop up version of it if they have any saved

  Expected Result: the above functionality of the test steps works.

  Actual Result: The above functionality of the test steps works.

4.6.5 Login Snippet

```
package com.example.daniel.loginregister;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class LoginActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        final EditText etUsername = (EditText) findViewById(R.id.etUsername);
        final EditText etPassword = (EditText) findViewById(R.id.etPassword);
        final Button bLogin = (Button) findViewById(R.id.bLogin);
        final TextView registerLink = (TextView) findViewById(R.id.tvRegisterHer

        registerLink.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v){
                Intent registerIntent = new Intent(LoginActivity.this, RegisterA
                LoginActivity.this.startActivity(registerIntent);
            }
        });...
```

4.6.6 Registration Snippet

```
package com.example.daniel.loginregister;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;

public class RegisterActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_register);

        final EditText etUsername = (EditText) findViewById(R.id.etUsername);
        final EditText etAge = (EditText) findViewById(R.id.etAge);
        final EditText etEmail = (EditText) findViewById(R.id.etEmail);
        final EditText etWeight = (EditText) findViewById(R.id.etWeight);
        final EditText etSex = (EditText) findViewById(R.id.etSex);
        final EditText etPassword = (EditText) findViewById(R.id.etPassword);
        final Button bRegister = (Button) findViewById(R.id.bRegister);...
```

4.6.7 User Account Info Page snippet

```
package com.example.daniel.loginregister;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;

public class UserPage extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_page);

        final EditText etEmail = (EditText) findViewById(R.id.etEmail);
        final EditText etUsername = (EditText) findViewById(R.id.etUsername);
        final EditText etAge = (EditText) findViewById(R.id.etAge);
        final EditText etSex = (EditText) findViewById(R.id.etSex);
        final EditText etWeight = (EditText) findViewById(R.id.etWeight);...
```

4.6.8 Routes Page and Button Pop Up Snippet

```
package com.pathway.pathway_android;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class routespopup extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_routespopup);
```

```
        Button route = (Button) findViewById(R.id.route);
        route.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v){
                startActivity(new Intent(routespopup.this, routepop.class));
            }
        });...
```

```
package com.pathway.pathway_android;

import android.app.Activity;
import android.os.Bundle;
import android.util.DisplayMetrics;

/**
 * Created by Daniel on 10/20/2017.
 */

public class routepop extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.routepopwin);
        DisplayMetrics dm = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(dm);
        int width = dm.widthPixels;
        int height = dm.heightPixels;
        getWindow().setLayout((int)(width*.8), (int)(height*.6));...
```

4.7 System Conversion and Implementation

During the process of designing the user account sub-system a couple of things changed. For one the creation of the user statistics page didn't happen due to time constraint and therefore was split up to be integrated into the user account page and the user routes page displaying necessary statistics on each. And lastly the designing of the singleton pattern was nixed and instead implemented on the server side by Cory's sub-system in order to take care of authentication. Other than those two things the implemenation of the user accounts sub-system remaind the same.

4.8 Design Alternatives

There were two design ideas for this sub-system. The first of which was to pull all relavent information for the user from the server datbase. The second of which was to pull some data from the servers database and some from the the local database on the device.

4.8.0.1 First Design

The idea with the first design was to have all the user account pages make requests to the servers database to get and display all pertinent user information. The login/register between the two design choices are the same.

4.8.0.2 Second Design

The idea for the second design, which is what was used for the sub-system, was that each user page would make appropriate calls to both the server database and a local device side database and then display that information appropriately to the user. The login/register between the two design choices are the same.

5 Pathway - Platform

- The system will be developed using Android v5 (Lollipop).
- The system will be developed in Android Studio.
- The system will employ the Google Maps API in development.

6 Pathway - Domain Analysis

This document serves as an overview for the planned development of a software system, named Pathway, that is intended to address the difficulties in staying motivated to exercise by incorporating elements of social interaction and friendly competition as a means to motivate through a mobile application interface.

6.0.0.1 Pathway - Extensions

- Increased social networking functionality.
- More detailed performance analytics.
- A companion website for profile management, advanced performance display options and tools.
- Additional functionality to allow a user to track nutrition intake and plan meals.
- Addition of workout tracking and planning such as weight lifting routines.

6.0.0.2 Pathway - General Domain Knowledge

- Users interested in tracking fitness routes have other options, but those applications are often limited in functionality beyond data collection.

- Meaningful analysis requires exporting of data or manual input into an outside system.
- While many individuals like to share their workout information on social media sites, there does not currently exist an integrated fitness oriented social networking option.
- Finding new routes to walk/run/cycle for a user, especially in a new area, can be difficult. Users may solicit advice for where to try these activities on social media.
- Users may also elect to try new areas with no prior knowledge.

6.0.0.3 Pathway - Clients and Users

Target users would include all individuals that:

- Are capable of operating a device that is compatible with Pathway.
- Are interested in exercising and making connections.

6.0.0.4 Pathway - Computing Environment

Primary functionality will be developed for Android 5.0 (Lollipop) and as such will be unsupported on version of Android prior to 5.0. Specific system requirements are not yet determined. No additional client requirements will be necessary for database and server access, other than an active network connection.

6.1 Pathway - Tasks and Procedures

Currently the general process of selecting a route to traverse (walk, run, or cycle) and then sharing information goes as follows:

1. A user solicits route information from friends and acquaintances.
    - If successful, the user may use an existing application to log route and speed information.
    - This information can be exported to an application, such as excel, and analyzed.
    - A user may elect to also post performance information through social media, such as Facebook.
2. A user can elect to try a new route.
    - A route could be to the user's liking and fitness level, making it more likely that that the route will be completed and traversed again in the future .

- If the route is difficult or unpleasant, the user may not finish their planned traversal of the route.

3. A user may use a 3^rd party application to find a group with which to walk or run or bike with.
    - This method has the advantage of added support from the group.
    - The user is more likely to finish and even more likely to perform better than if alone.

6.2 Pathway - Competing Software

The following applications/software may fulfill similar roles or offer similar functionality as that proposed in this document for Pathway:

- MyTracks - Offers only basic data collection (Lat/Long position, elevation) dependent on device sensors, and no analysis options.
- FitBit App - Offers data collection and basic analysis of user performance.
- iOS Health - Offers basic data collection and limited analysis of performance.
- LG Health - Offers route tracking and calorie journal. Still lacks any social networking component.

6.3 Pathway - Similar Domains

Many applications are similar to Pathway in the sense that they make use of geospatial data location and/or social networking user accounts. While no other application we found combines all aspects of our project in the same way we do, it will be worthwhile to look at other applications, such as Waze, Facebook, etc to gather some understanding of how to efficiently implement these systems.

7 Glossary

- **Route** - A Route in Pathway will be the geometric representation of a users tracked traversal path and will include geographic data that defines an exact geographic position through XY coordinates and elevation.
- **Challenge** - A Challenge in Pathway will be the primary means of social interaction between users, acting as a prompt from User A to User B to encourage participation in an exercise activity.
- REST API - Representational State Transfer, provides interoperability between systems through the web.
- **Python** - Python is an interpreted, object-oriented programming language similar to PERL.

- **Flask** - Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions.
- **MySQL** - MySQL is an open source relational database management system (RDBMS) based on Structured Query Language (SQL).
- **Object Relational Mapping (ORM)** - Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages.
- **Docker** - The Docker Engine is the underlying client-server tool that supports container technology to handle the tasks and workflows involved in building container-based applications.
- **Amazon Web Services (AWS)** - Amazon Web Services (AWS) is a comprehensive, evolving cloud computing platform provided by Amazon.com.
- **scalable** - It is the ability of a computer application or product (hardware or software) to continue to function well when it (or its context) is changed in size or volume in order to meet a user need.
- **dockerize** - The act of containing a piece of software within a docker container.
- **Route** - A Route in Pathway will be the geometric representation of a users tracked traversal path and will include data that defines an exact geographic position through XY coordinates and elevation.
- **Challenge** - A Challenge in Pathway will be the primary means of social interaction between users, acting as a prompt from User A to User B to encourage participation in an exercise activity.
- **Statistics** - branch of mathematics dealing with the collection, analysis, interpretation, presentation, and organization of data.
- **Database** - application that contains organized collections of data in tables
- **Geolocation** A specific pair of x and y coordinates as plotted on a spherical surface, allowing the description of an exact geographic location on that sphere.
- **Latitude** The angular distance of a location in degrees, minutes, and seconds relative to the earths equator. Lat is shorthand.
- **Longitude** The angular distance of a location in degrees, minutes, and seconds relative to the meridian at Greenwich, England. Long is shorthand.

- **Elevation** A measurement of height above some arbitrary baseline, usually sea-level, expressed most often in feet or meters. Elev. is shorthand.
- **User** An individual person who maintains a Pathway account.
- **Route** A series of two or more data points in sequence that comprise the path taken by a user and showing the users traversal across an area.
- **Point** A recorded location, being comprised of Latitude, Longitude, Elevation, and Time, relative to the first recorded point in the series.

8 Bibliography

- - Django.readthedocs.io. (2017). Using the Django authentication system Django 2.1.dev20171017153446 documentation. [online] Available at: http://django.readthedocs.io/en/latest/topics/auth/default.htmlhow-to-log-a-user-in [Accessed 17 Oct. 2017].
- - Cohen, U. (2017). Write your own Android Authenticator. [online] Blog.udinic.com. Available at: http://blog.udinic.com/2013/04/24/write-your-own-android-authenticator/ [Accessed 17 Oct. 2017].
- - Docs.djangoproject.com. (2017). django.contrib.auth — Django documentation — Django. [online] Available at: https://docs.djangoproject.com/en/1.11/ref/contrib/auth/ [Accessed 17 Oct. 2017].
- - Python-3-patterns-idioms-test.readthedocs.io. (2017). The Singleton Python 3 Patterns, Recipes and Idioms. [online] Available at: http://python-3-patterns-idioms-test.readthedocs.io/en/latest/Singleton.html [Accessed 17 Oct. 2017].
- - Developer.android.com. (2017). HttpURLConnection — Android Developers. [online] Available at: https://developer.android.com/reference/java/net/HttpURLConnection.htm [Accessed 13 Nov. 2017].
- - www.hackernoon.com. (2017). AWS vs. DigitalOcean. [online] Available at: https://hackernoon.com/aws-vs-digitalocean-which-cloud-server-is-better-1386499a6664 [Accessed 19 Nov. 2017].

References

[1] e. a. Butler, "The geojson format." Online, aug 2016. ISSN: 2070-1721.

[2] "AsyncTask | Android Developers," nov 2017.

[3] S. Contini, "Android development, fetch data from api and execute a callback.," nov 2017.

[4] R. Czerniak, R. Genrich, N. C. H. R. Program, A. A. of State Highway, T. Officials, U. S. F. H. Administration, and N. R. C. U. T. R. Board, *Collecting, Processing, and Integrating GPS Data Into GIS*. NCHRP synthesis, National Academy Press, 2002.

[5] "Hausdorff distance," Sept. 2017. Wikipedia, Page Version ID: 802180359.

[6] N. Gregoire and M. Bouillot, "Hausdorff distance between convex polygons." Online, 1998. Accessed Nov, 2017.

[7] "Fréchet distance," Sept. 2017. Wikipedia, Page Version ID: 800573982.

[8] T. Eiter and H. Mannila, "Computing discrete fréchet distance," 05 1994.

[9] "Map Objects | Google Maps Android API."

[10] "Elevation | Google Maps Android API."

[11] "Google Maps Android GeoJSON Utility | Google Maps Android API."