Maddox Scott

Mr. Pierson

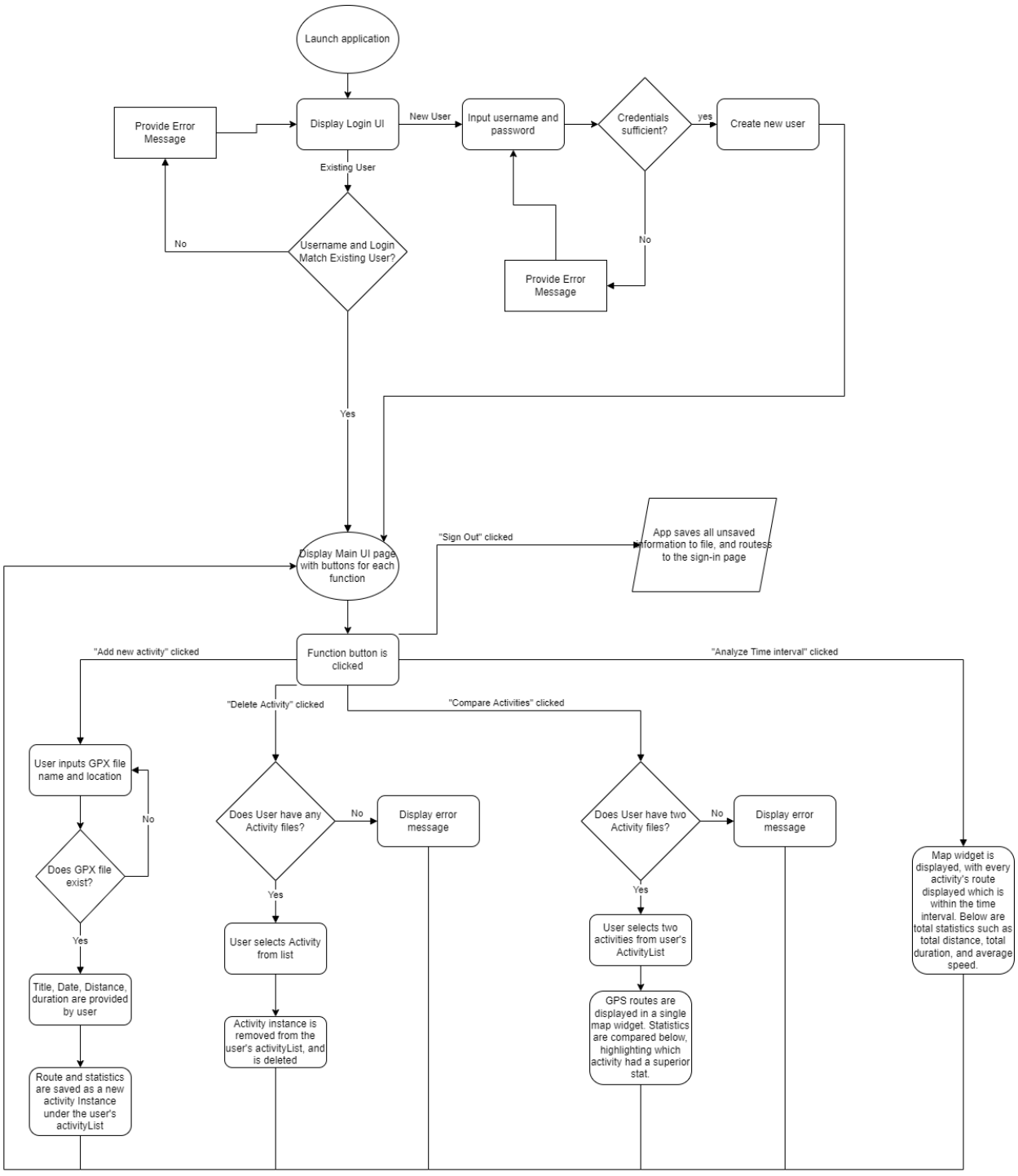
IB Computer Science, Period 1

11 February 2022

Criterion B: Design

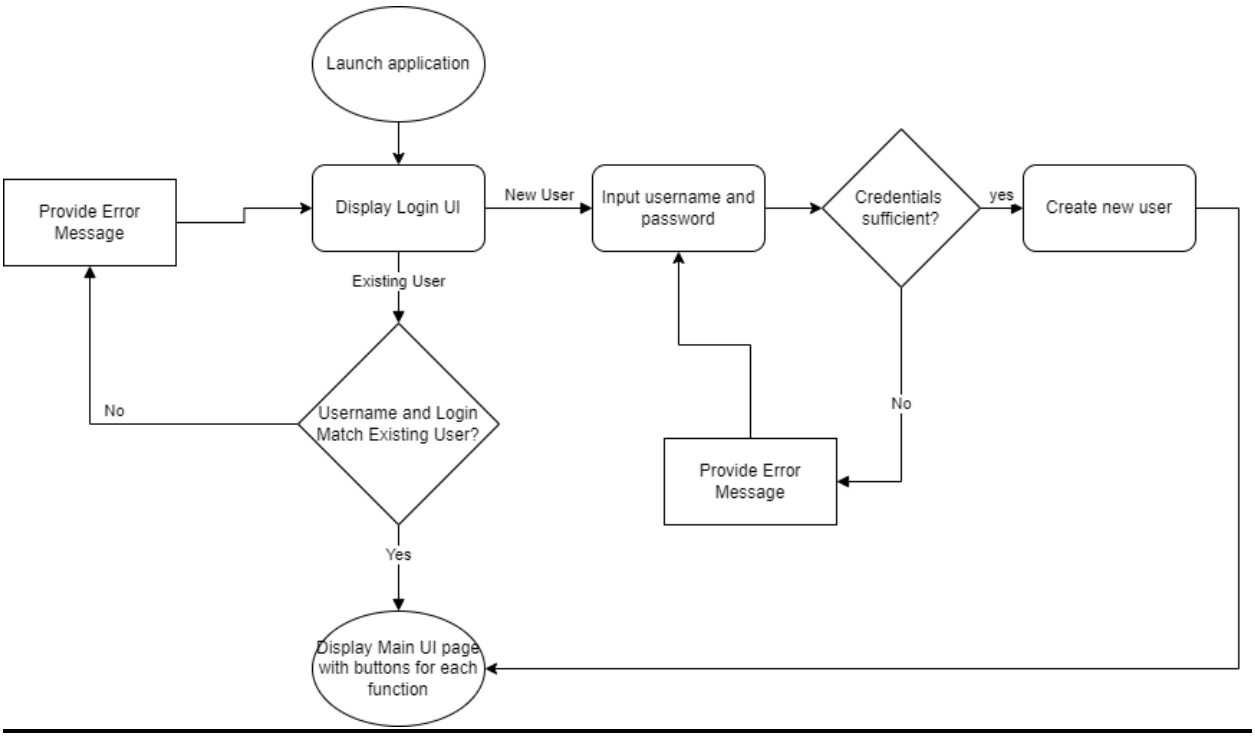
Flowchart(s)

Application Flowchart: refer to Appendix (D.1) for initial iteration

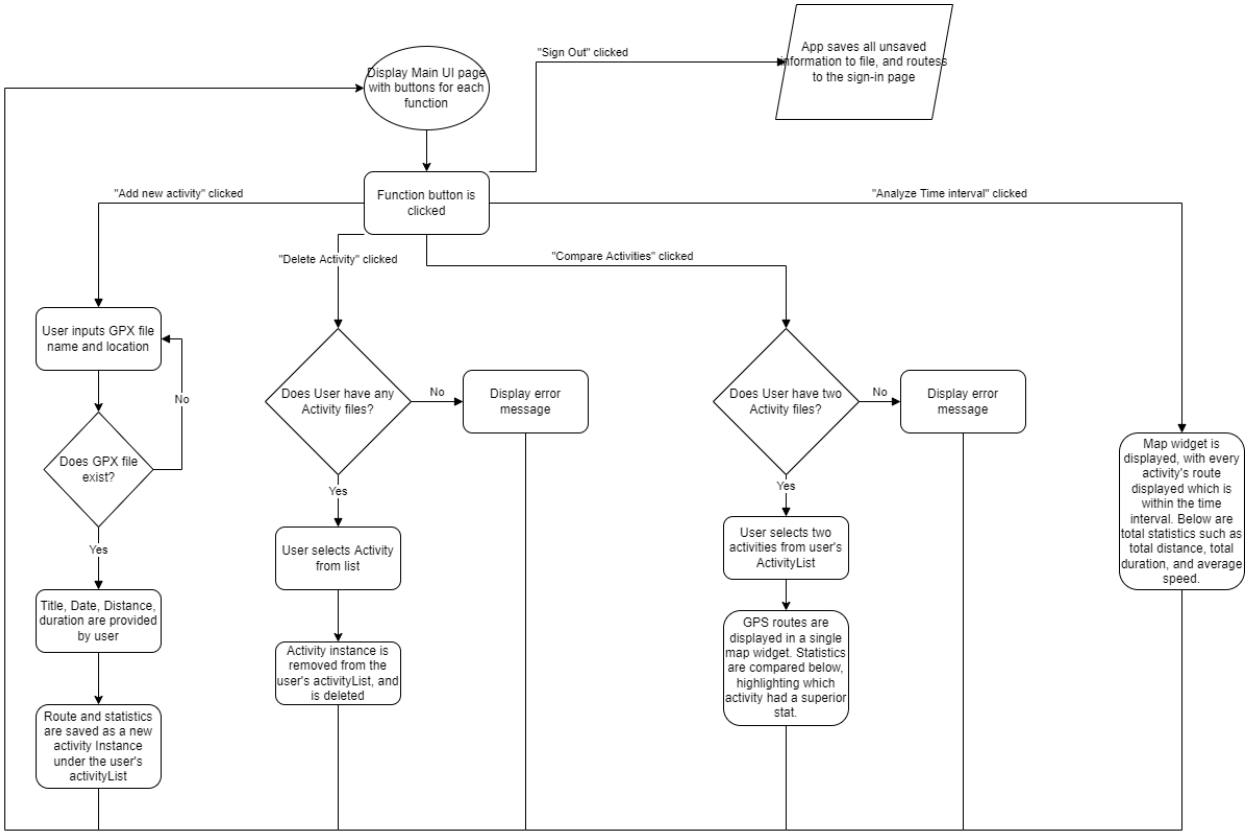


Login Algorithm: refer to Appendix (D.2) for initial iteration. When Starting the application, the user must sign in using their username and password. If they have not created a new user, they are routed to the ‘new user” page to do so. Whenever a sign-in or new user creation fails, the

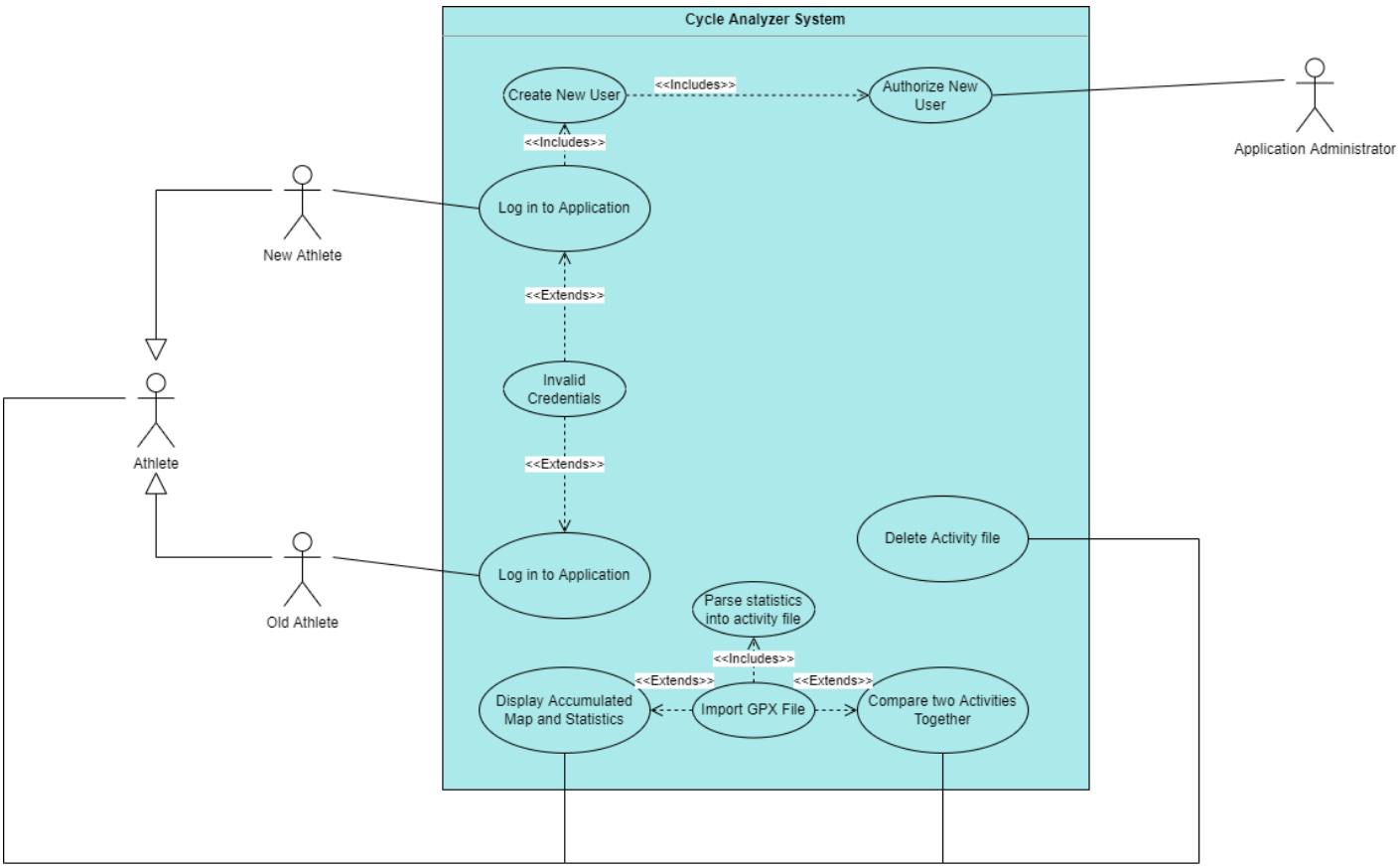
user is routed to the start of the algorithm, the Login UI. This loop repeats until a successful login is performed.



Home-Screen Algorithm: This algorithm handles all visual elements once the user is signed in. Each button routes them to the corresponding view, after checking if the user has all necessary object instances before deciding to route them back to the hub or display the function view. When the user is done, they are routed back to the home view. Algorithms are finite, so when the user is finished, they log-out of the application, ending the loop



Use Case Diagram



Data Structures

User and Activity data will be stored as Maps under Shared Preferences. The imported API SharedPreferences allows for any dart class within the app to locate and read stored information, regardless of its file path. It also makes updating user information simple, as the methods for storing user information for the first time as well as updating user information are equivalent.

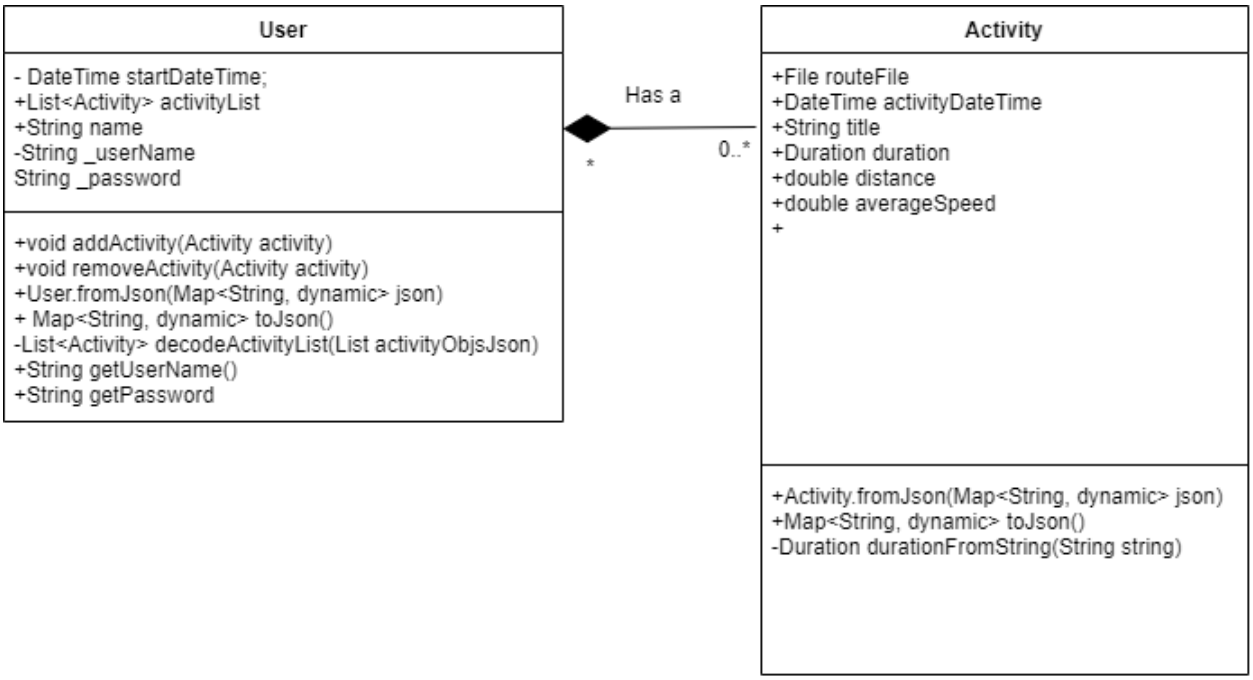
```
//Method used to save/update a user's information to SharedPreferences, and is saved locally.
void saveUser(User user, SharedPreferences prefs) async {
  String json = jsonEncode(user.toJson());
  prefs.setString(user.getUserName(), json);
}
```

Information stored here persists between restarts as well. Prior to being saved in Shared Preferences, the user is converted into a Map. A map is a set of unordered Strings, which each have a key used to identify each String. Using a Map to store data means that pulling data from

Shared Preferences does not require a loop to search a list, and the time required to locate user data is always constant.

Each User instance has a list of Activity instances, representing bike rides. They are stored in a list so that each Activity can be easily called from using its corresponding index. Lists can also change their lengths or number of elements, making them superior to arrays for this application. Lists are included in the default Flutter library, so no imports are needed.

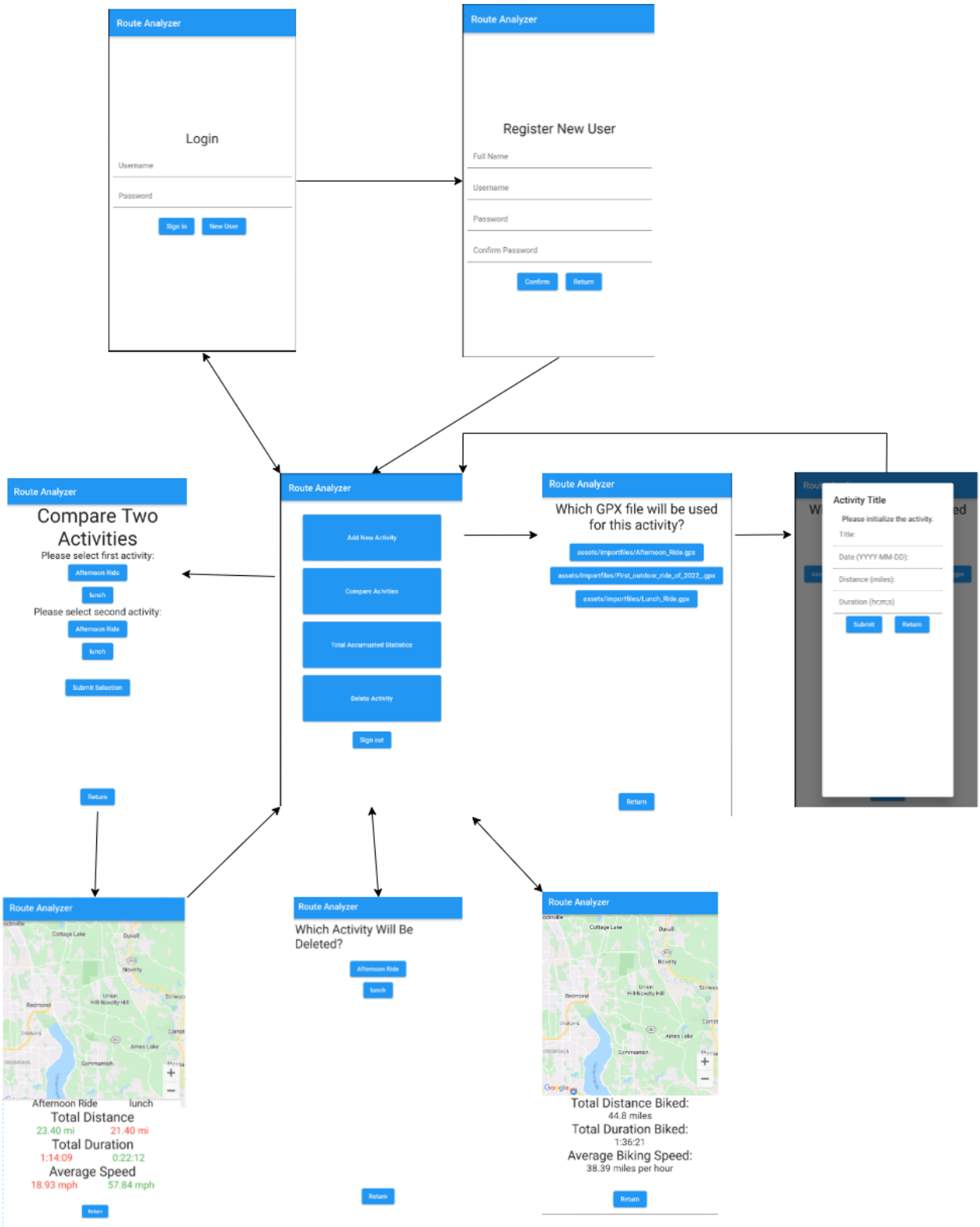
Objects / UML Diagram: For earlier iterations, Refer to Appendix (D.3). Changes were made to initial UML as the required methods for each class changed. Heart Rate and elevation were removed from activity, as there was no way to collect them from the GPX.



Class	**Description**
user	Represents a user of the application, which can create or remove activities from their activityList. They possess a name, username and password.

	_username and _password are needed for authentication to the application, and are private so that they cannot be tampered with or obtained externally. Getter methods are provided so that the login algorithm can utilize them. toJson() is used to convert each User instance to a Map string, which is saved locally in Shared Preferences. fromJson() converts a Map String back into a User instance. With addActivity() and removeActivity(), the user instance can alter their activityList whenever necessary.
Activity	Each Activity instance has a File variable which represents its corresponding GPX file. This variable is called upon during map creation. The duration of a ride is saved as a Duration variable, which automatically formats itself into hh/mm/ss format, which makes displaying the duration simple. Just like User, Activities must be able to be converted and retrieved from Map Strings, so Activity also has toJson() and fromJson() methods.

UI Flows: For earlier iterations and revision feedback, Refer to Appendix (C.1)


Route Analyzer
Login
Username
Password
Sign In
New User
Route Analyzer
Register New User
Full Name
Username
Password
Confirm Password
Confirm
Return
Route Analyzer
Add New Activity
Compare Activities
Total Accumulated Statistics
Delete Activity
Sign out
Route Analyzer
Compare Two Activities
Please select first activity:
Afternoon Ride
lunch
Please select second activity:
Afternoon Ride
lunch
Submit Selection
Return
Route Analyzer
Which GPX file will be used for this activity?
assets/importfiles/Afternoon_Ride.gpx
assets/importfiles/First_outdoor_ride_of_2022_.gpx
assets/importfiles/Lunch_Ride.gpx
Return
Activity Title
Please initialize the activity.
Title:
Date (YYYY-MM-DD):
Distance (miles):
Duration (hr:mm:s)
Submit
Return
Route Analyzer
Afternoon Ride
lunch
Total Distance
23.40 mi
21.40 mi
Total Duration
1:14:09
0:22:12
Average Speed
18.93 mph
57.84 mph
Route Analyzer
Which Activity Will Be Deleted?
Afternoon Ride
lunch
Return
Route Analyzer
Total Distance Biked:
44.8 miles
Total Duration Biked:
1:36:21
Average Biking Speed:
38.39 miles per hour
Return

Test Plan

Success Criteria	Test Plan
1	**Plan:** The gpx file “Afternoon_Ride” will be placed in the app’s directory and the application will attempt to locate, read, and extract all its contents into an Activity instance. This Activity must also have a distance, duration and speed, all of which are not null. I will print the instance’s toJson method to see if all necessary information was collected. **Positive Outcome:** The Activity’s toJson() method returns the activity’s title, data, duration, distance, and speed. **Negative Outcome:** When running the toJson() method, if the application detects any statistic field to be null, the app will handle the NullPointerException by printing an error message.
2	**Plan:** Using the Activity created from “Afternoon_Ride.gpx”, the “display total accumulated statistics” button will attempt to display the activity’s route in a map widget. **Positive Outcome:** All portions of the bike route are successfully displayed in the google maps widget at an appropriate size. The user has the ability to zoom in or out within the map widget. **Negative Outcome:** The google maps widget can’t convert the GPX file to a visual map, displaying nothing.
3	**Plan:** A second gpx file will be used to create another Activity instance, with a different route, distance, duration, and average speed. Both activities will be displayed, with the higher statistic of the two displayed in green, and lower in red. **Positive Outcome:** All statistics are displayed, in the correct color according to their comparison. The routes are displayed in a Map **Negative Outcome:** Statistics are displayed in the incorrect colors, or both routes do not display.
4	**Plan:** A hypothetical “user” will be created, with the username “Tester”. Upon creation, a simple String such as “password123” will be used as a password. **Positive Outcome:** Tester can log into the application using its password and username declared for the Tester user instance. Attempting to log in with a different password fails as intended, and Tester is still able to login after an application restart.

	Negative Outcome: The Tester cannot login, and the user cannot access the main page of the app. The credentials fail to save locally and do not work upon restarting the application.
5	**Plan:** Using both of the Activity instances created under the "Tester" user in previous steps, the application will display the sum of each individual statistic of both activities. The routes of both rides will also attempt to be displayed using a map widget. **Positive Outcome:** The application displays the correct sums for each statistic between the two rides. "Total Distance Traveled" must equal the sum of the distances traveled for both rides. The map displays both routes in a single map, proving that the app is scalable and can display many more routes at once. **Negative Outcome:** The application incorrectly displays the sums of accumulated states, providing a NullPointerException. The map cannot display two routes at a single time, or requires two map widgets to do so.
6	**Plan:** The Tester user will attempt to delete the second Activity instance, and run the "accumulated statistics" test from the previous step again. **Positive Outcome:** The application will display the statistics of Activity one rather than the sum of Activity one and two. The map only displays the route of Activity one as well. **Negative Outcome:** The application still displays the accumulated statistics between the two activities. The Map still displays Activity two's route.
7	**Plan:** Search the entire application for any low contrast or small text. **Positive Outcome:** Every piece of text within the app is above 12px in size. High contrast ratio color combinations are used for text, such as blue on white. Widgets implement simple animations on their entrance and exit. **Negative Outcome:** Legible text, either created by myself or implemented from a third party widget, is smaller than 12px, and difficult to read. Poor color combination contrast rations such as gray on white are used, and there are no animations whatsoever.