CS 491 Senior Design Project I

Low Level Design Report

Project short-name: Pigeon's Map

Berdan Akyürek 21600904

Ömer Olkun 21100999

Tanay Toksoy 21703919

Abdullah Ayberk Görgün 21201986

Ekin Üstündağ 21602770

Supervisor: Özcan Öztürk

pigeon-s-map.github.io

1. Introduction

Choosing an efficient path to a destination point is usually not easy. If there are multiple destinations, it is even more complicated. Postal workers have to think of tracks with various delivery points every day, and if they choose a path that is close to the optimal option, they can save both time and fuel.

Moreover, the Covid-19 pandemic increased the number of online orders, so more packages are delivered [1]. Being more efficient is essential nowadays due to the increased workload.

Anyone can use Pigeon's Map to find an efficient path, but it mainly helps postal workers find efficient routes in their delivery with multiple destinations. This android application takes a start point and multiple goals with priorities to find a delivery order with the shortest distance in total. Pigeon's Map dramatically reduces the time to think of an efficient path and delivery.

1.1 Object design trade-offs

Complexity vs. Usability

It is better to have complex processes to save time, but Pigeon's Map needs to be used by as many users as possible. Anyone with minimum knowledge about technology should be able to use it easily. Usability is more critical for Pigeon's Map.

Cost vs. Performance

It would be better to have a server handle pathfinding with high computational power, but we cannot afford such servers. The device of the users will handle the pathfinding computations.

Memory vs. Maintainability

It would be good to use as low memory as possible, but finding an efficient path for multiple destinations can be complicated. Memory is still essential but maintaining the lower time in calculations is more critical. We will focus on maintainability on this project.

1.2 Interface documentation guidelines

We used a table format for the class interfaces such that Pascal Case is used for class names, and the Camel Case is used for attribute and function names.

Name of the class	ClassName
Description of the class	Brief description
List of the attributes	attributeName: AttributeClass
List of the functions	myFunction(ClassName1, ClassName2): ReturnTypeClass

1.3 Engineering standards

We used UML [2] guidelines in previous reports for class models and scenarios. We used the IEEE format for referencing in every report.

1.4 Definitions, acronyms, and abbreviations

UML: Unified Modeling Language

Pascal Case: Naming convention in which the first letter of every word is in capital

Camel Case: Naming convention in which the first letter of every word except the first one is in capital

2. Packages

2.1 Model

Model classes represent the data structures that we have in the server.

User.java
UserSettings.java
Route.java
Warning.java
Location.java
Street.java
Database.java

2.2 Activities

These are controller classes; they extend the Activity base class from Android SDK.

MainActivity.java
LoginActivity.java
RegisterActivity.java
SelectAddressActivity.java
MapActivity.java

3. Class Interfaces

User
This class holds information for a user like settings, location, login information and routes
- username: String = “guest” - settings: UserSettings - warnings: List<Warning> - location: Location - route: route - savedRoutes: List<Route>
+ setSettings(bool, bool, String, int, String): void + setSettings(UserSettings): void + login(String):bool + updateLocation(Location): bool + createWarning(String, Location, bool): bool

UserSettings
This class holds the list of settings for a user.
- autoLogin: bool - savePastRoutes: bool - loginToken: string - maxRoutesToStore: int - units: String
+ setAutoLogin(bool): void + setSavePastRoutes(bool): void + setLoginToken(String): void + setMaxRoutesToStore(int): void + setUnits(String): void

Route
This class represents a route.
- circular: bool - locations: List<Location> - length: float - noOfLocations: int - streets: List<Street> - warnings: List<Warning>
+ calculate(): bool + addLocation(Location): void + removeLocation(Location): void + update(): bool

Warning
This class holds information for a single warning
- text: String - user: User - location: Location - isPrivate: bool
+ getText(): String + getUser(): User + getLocation(): Location + getPrivate(): bool + setText(String): void + setUser(User): void + setLocation(Location): void + setPrivate(bool): void

Location
This class represents the coordinates of the location and street.
- xCoordinate : float - yCoordinate : float - street : Street
+ setXCoordinate(float) : void + setYCoordinate(float) : void + getXCoordinate() : float + getYCoordinate() : float

Street
This class represents the rate of the street where user's vote.
- rate: int - name: String
+ setRating(int): void + setName(String): void + getRating(): int + getName(): String

Database
This is the class that manage the connection between Client side and database.
- databaseConnectionInfo: String
+ fetchUserInfo() + checkLoginInfo(String, String): bool + addWarning(Location, String): void + getWarnings(Route): List<Warning> + getStreets(User, List<Street>): List<Street>

Map
This is a map object that hold the information about the current map
- currentRoute: Route
+ showWarningDetails(Warning): void + showSubmittedMessage(): void + update(): void + showCreateWarningDialog(): void + display(): void

Activities

MainActivity
This is the welcome screen of the application
- loginButton: Button - registerButton: Button - continueButton: Button
+ void onCreate(Bundle)

LoginActivity
This is UI of the login screen.
- emailTextField: EditText - pwdTextField: EditText - submitButton: Button - googleButton: Button
+ void onCreate(Bundle) + submit():

RegisterActivity
This is the UI of the register screen.
- emailTextField: EditText - pwdTextField: EditText - pwdTextField2: EditText - submitButton: Button - googleButton: Button
+ void onCreate(Bundle)

SelectAddressActivity
This is the UI where the user will select the addresses.
- addressFieldList: List<EditText>
+ void onCreate(Bundle)

MapActivity
This is the UI where the route and map will be displayed to the user.
- map: Map - createWarningButton: Button - rateStreetButton: Button - warningButtons: List<Button> - reportButtons: List<Button>
+ void onCreate(Bundle) + createWarning(Warning) + rateStreet(Street, int) + reportWarning(String)

4. Glossary

Optimal Path: The most efficient path possible.

Efficient Path: A path close to the optimal way of saving time.

Destination: An address to stop by to deliver a package.

Point: Users can point to a road's circumstances to be considered good or bad in future searches.

Circumstances: Infrastructure or environment of the neighborhood about a road.

5. References

[1] "Pandemic increases workload, health risks for postal and delivery employees", *PBS NewsHour*. [Online]. Available: https://www.pbs.org/newshour/show/pandemic-increases-workload-health-risks-for-postal-and-delivery-employees. [Accessed: 12- Oct- 2020].

[2] “What is UML,” *What is UML | Unified Modeling Language*. [Online]. Available: https://www.uml.org/what-is-uml.htm. [Accessed: 08-Feb-2021].