TaxiNow

2024

Created by
TCE1 Group TREE

Table of Contents

Problem

Private Hire Vehicles

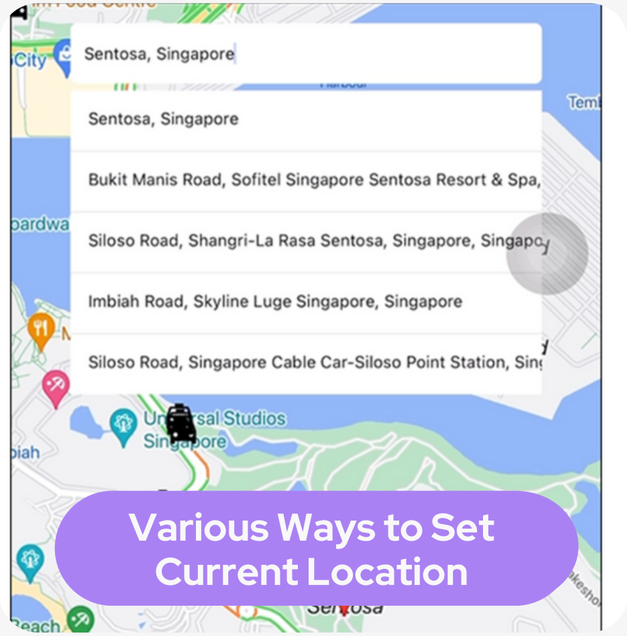
- High Cost 
- Might take a long time to find a hire

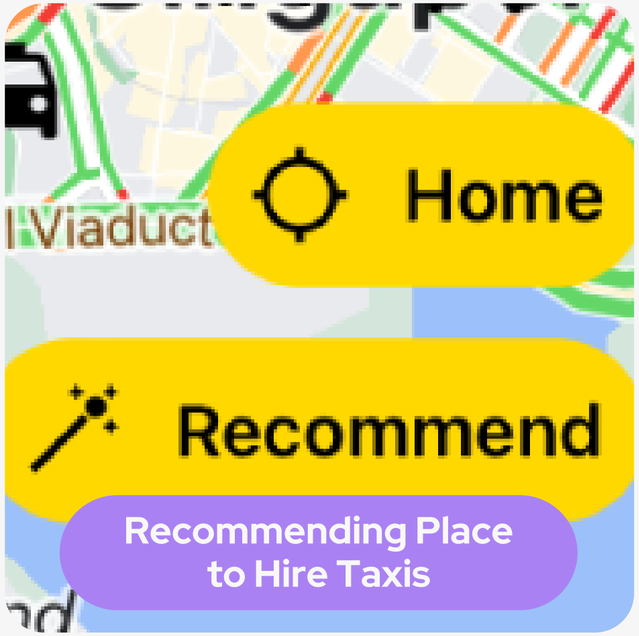
- Needs personal information and credit card details

Solution - TaxiNow

- Provides users with accurate real-time information on available taxis, and recommends an optimal location to flag one, based on current traffic conditions

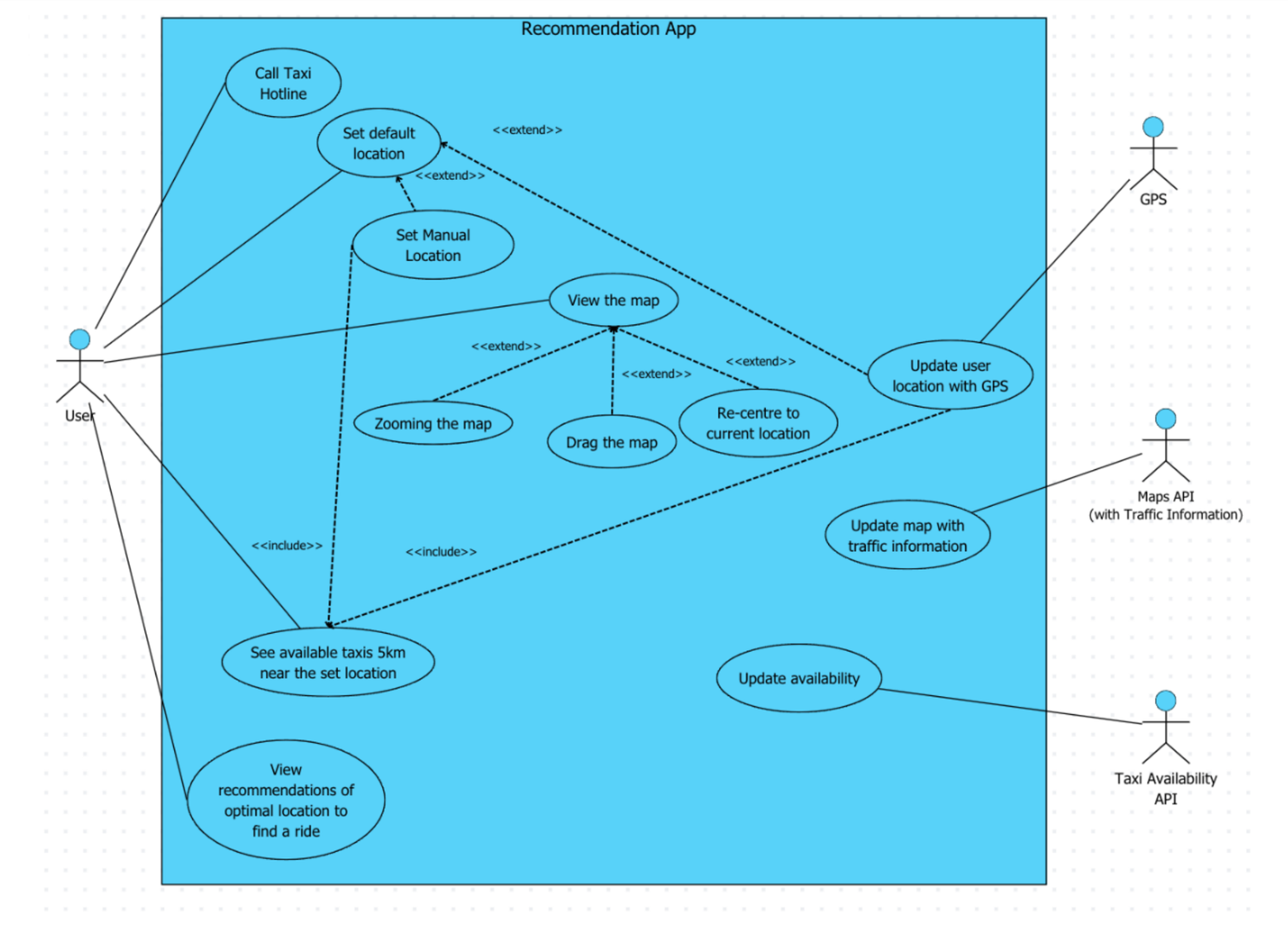
- Saves time and money without requiring account registration or sensitive information



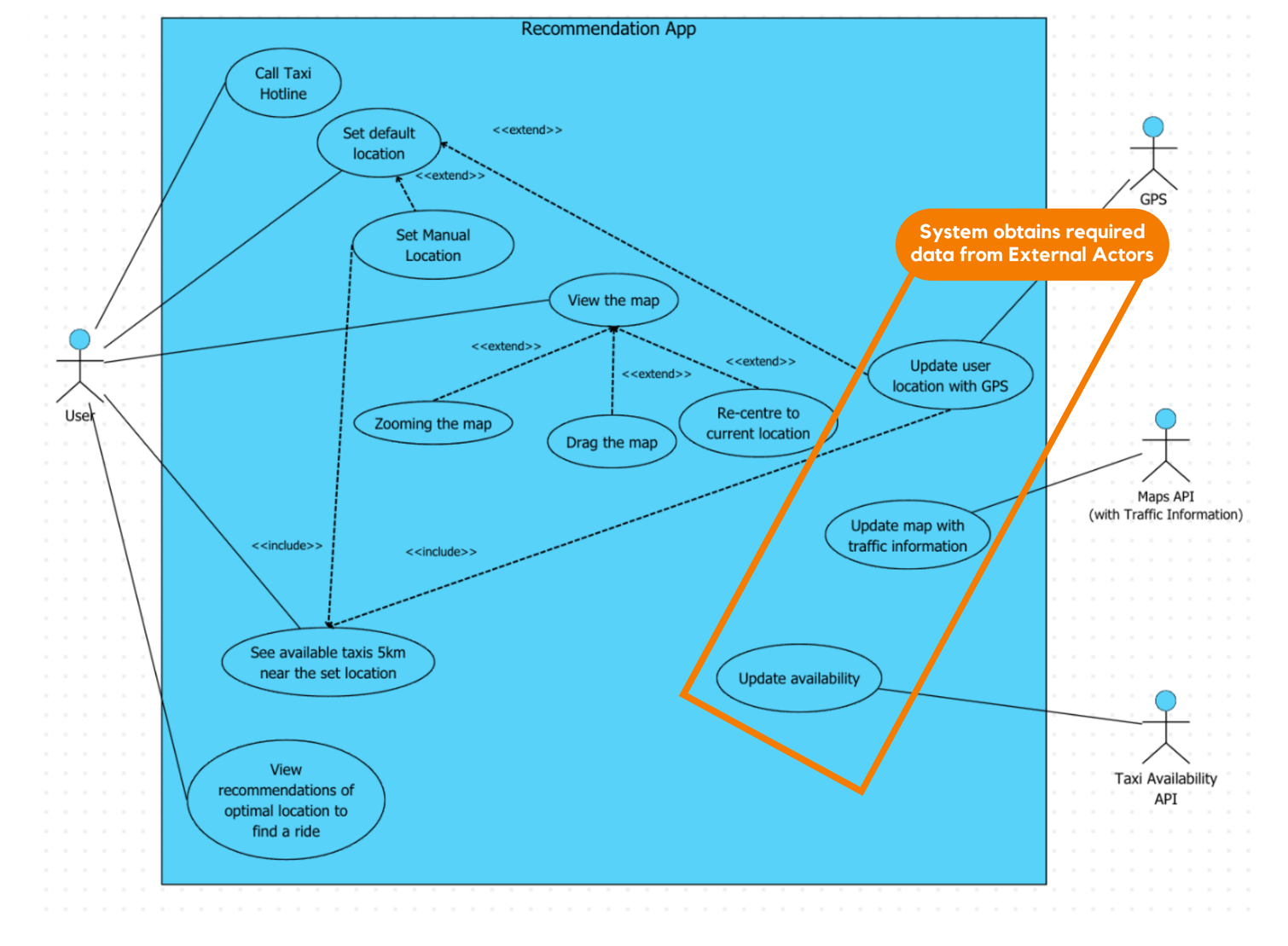


Key Features

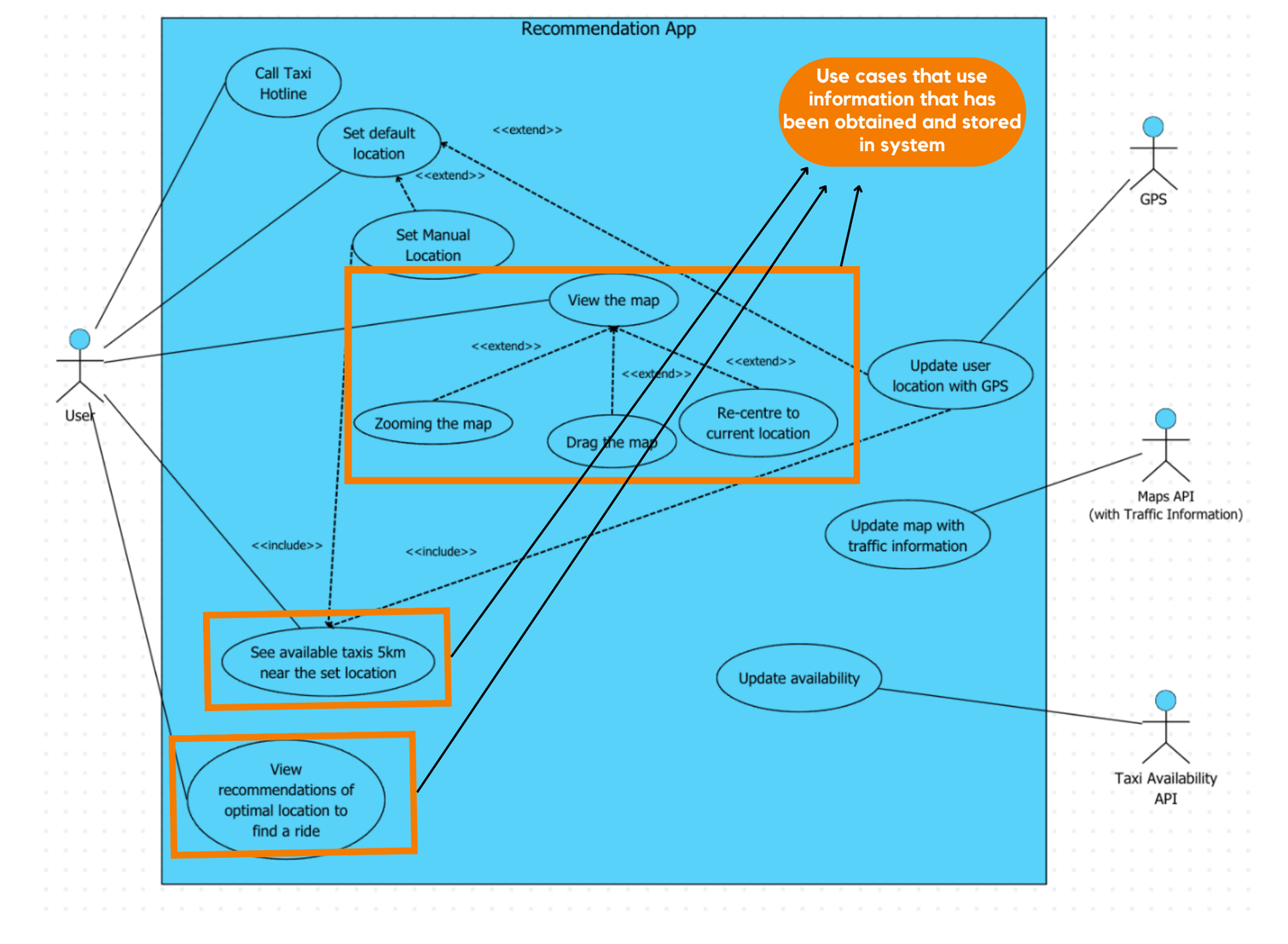
Use Case Diagram



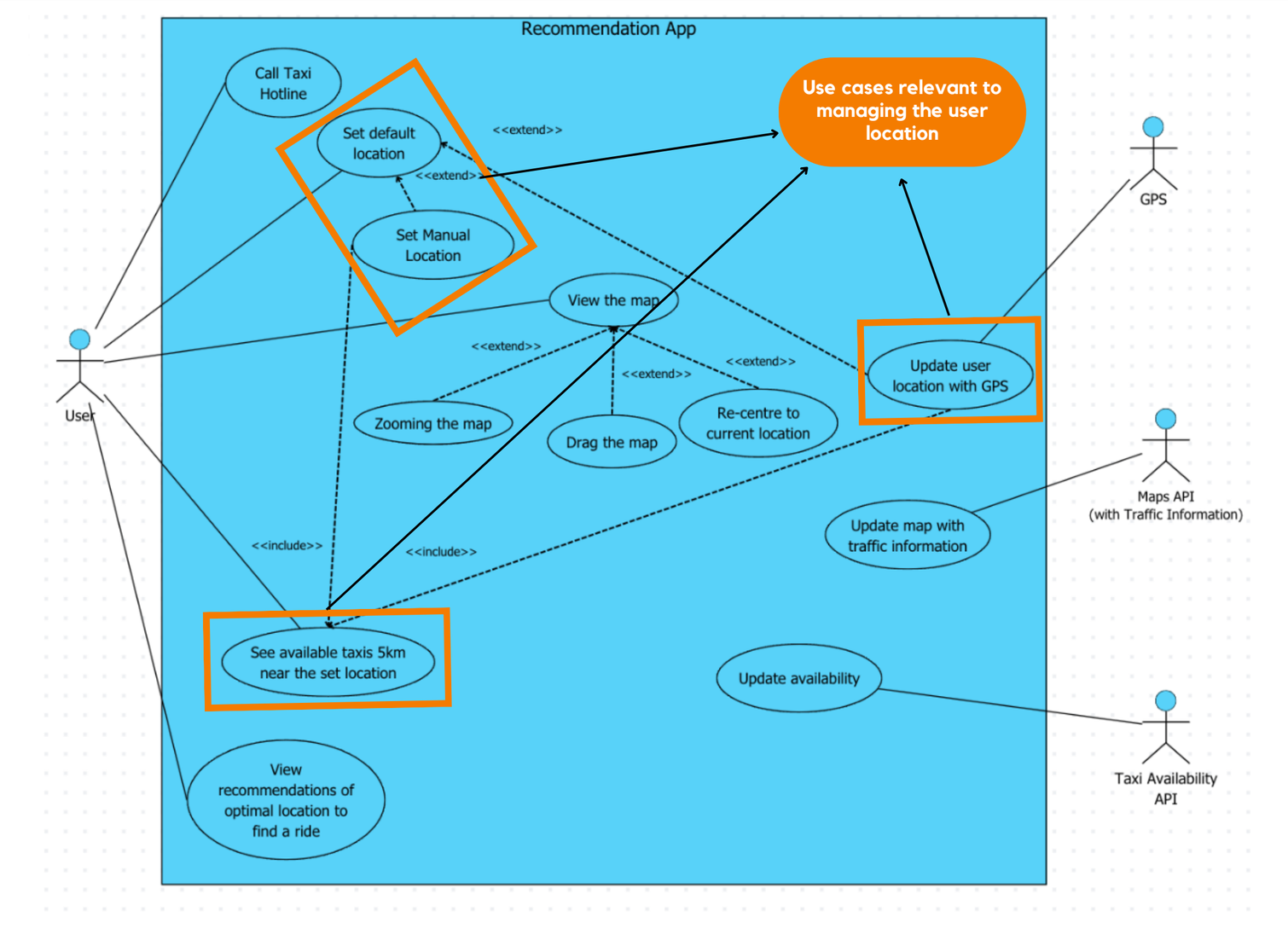
Use Case Diagram



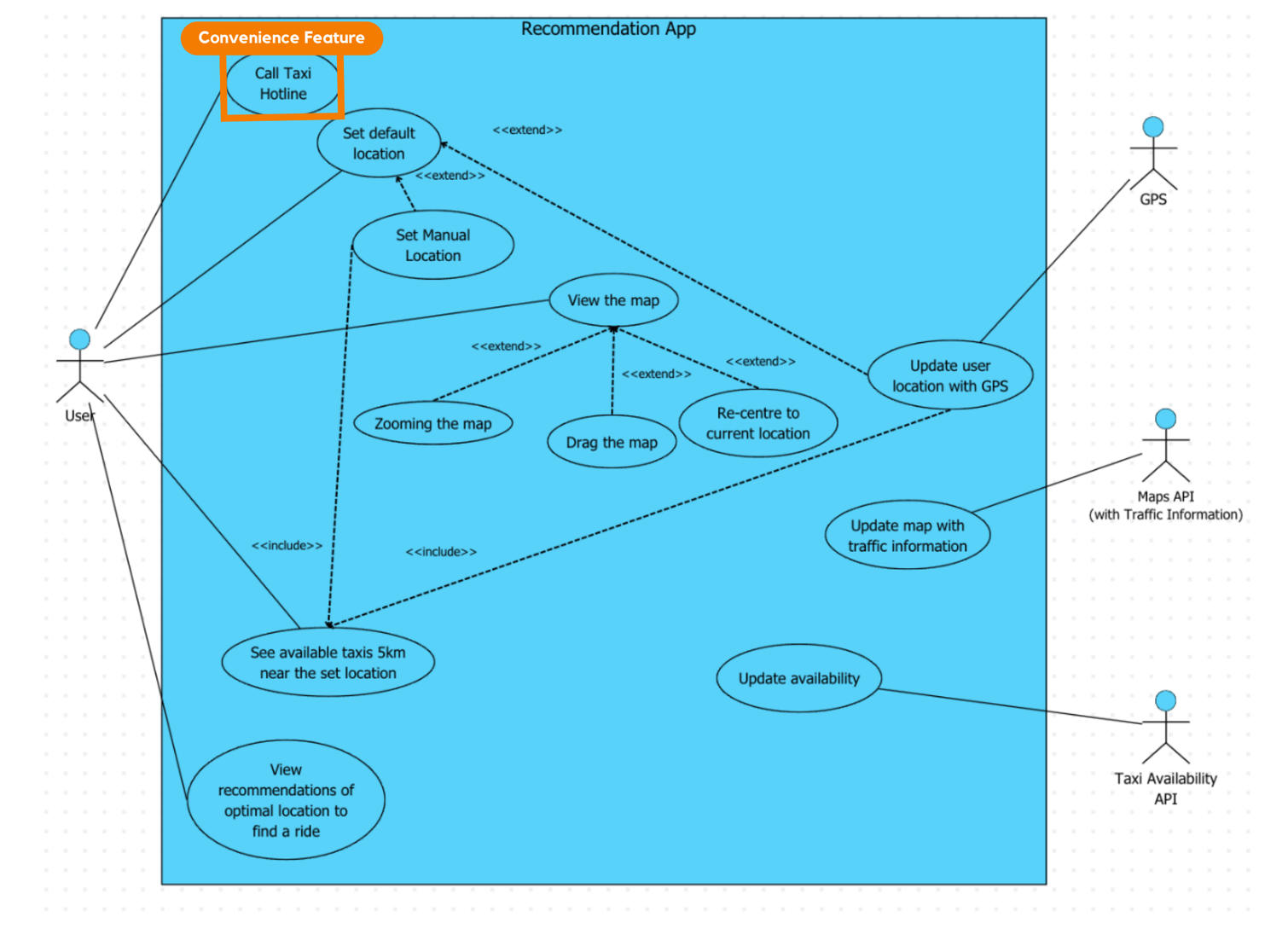
Use Case Diagram



Use Case Diagram



Use Case Diagram



Expected Users

Users who are:

- Looking for taxi locations
- Looking for a recommended location to hail a cab
- Struggling to find drivers on private hire applications
- Want to be able to get find a ride without keying in sensitive information on private hire applications

Product Demo

Good SWE Practices

1. SOLID PRINCIPLES
2. MOLDULAR & ORGANISED CODE
3. ERROR HANDLING & LOGGING
4. CONCURRENCY & SYNCHRONISATION

SOLID PRINCIPLES

Single Responsibility Principle

- Each package and function in our codebase has a single responsibility, making the code more modular and easier to understand.
- Example: locationAPI.go handles location-related functionality, while mapsAPI.go handles map-related functionality.

```go
// locationAPI.go
func handleLocation(c *gin.Context) {
    switch c.Request.Method {
    case "GET":
        handleGetLocation(c)
    case "PUT":
        handleUpdateLocation(c)
    default:
        c.String(http.StatusMethodNotAllowed, "Method not allowed")
    }
}
```

Open-Closed Principle

- Our code is open for extension but closed for modification. Permissible to add new functionality without modifying existing code.
- Example: Adding a new API endpoint can be done by creating a new function and route without modifying existing code.

```go
// server.go
r.GET("/taxi", handleTaxiCoordinates)
r.GET("/recommendations", getRecommendedLocationsHandler)
```

SOLID PRINCIPLES

Liskov Substitution Principle

- Our code uses interfaces and abstractions to allow for substitutability of objects.
- Example: The getRecommendedLocations function in recomendation.go depends on the getNearbyTaxiLocations function, which returns a slice of Coordinate structs. Any implementation that satisfies the Coordinate struct can be used

```go
// recommendation.go
func getRecommendedLocations(lat float64, lon float64, numOfRandomCoordinates int,
    var locations [][]float64
    taxiLocations := getNearbyTaxiLocations(lat, lon)
    // ...
}
```

Interface Segregation Principle

- Our code defines small and focused interfaces to avoid unnecessary dependencies.
- Example: The Coordinate struct in taxiAPI.go only contains the necessary properties for representing a coordinate.

```go
// taxiAPI.go
type Coordinate struct {
    Latitude  float64 `json:"latitude"`
    Longitude float64 `json:"longitude"`
}
```

SOLID PRINCIPLES

Dependency Inversion Principle

- Our code depends on abstractions (interfaces) rather than concrete implementations, making it more flexible and easier to modify.
- Example: The getRecommendedLocations function in recommendation.go depends on the getNearbyTaxiLocations function, which is an abstraction.

```go
// recommendation.go
func getRecommendedLocations(lat float64, lon float64, numOfRandomCoordinates in
    var locations [][]float64
    taxiLocations := getNearbyTaxiLocations(lat, lon)
    // ...
}
```

- The `getRecommendedLocations` function depends on the `getNearbyTaxiLocations` function, which is an abstraction. This allows for flexibility in the implementation of `getNearbyTaxiLocations`.

Observation Pattern

- Our code uses the observation pattern to trigger the query of taxi locations every time the current location changes. This allows loose coupling where functions that change currentLocation do not have to query taxi locations as well, but just have to notify the change in currentLocation.

```
105   useEffect(() => {
106     axios
107       .get(
108         `${server}taxi?latitude=${location.latitude}&longitude=${location.longitude}`
109       )
110       .then((res) => {
111         // setTaxiLocations(res.data);
112
113         if (res.data) {
114           setTaxiLocations(res.data);
115         }
116
117         setCanContactServer(true);
118       })
119       .catch((e: AxiosError) => {
120         console.log(e.request);
121         checkErr(e);
122       });
123   }, [location]); // reset taxi location every time current location is updated.
```

Good SWE Practices

Modular and Organized Code Structure

- Codebase divided into separate files based on functionality, such as locationAPI.go, mapsAPI.go, recommendation.go, server.go, and taxiAPI.go

Error Handling and Logging

- Our code includes proper error handling and logging, improving the reliability and maintainability of the application.

```go
if err != nil {
    log.Printf("Error in handleStreetNameToCoordinates for streetName %s: %v", str
    c.JSON(http.StatusInternalServerError, gin.H{
        "error": "Internal Server Error",
    })
    return
}
```

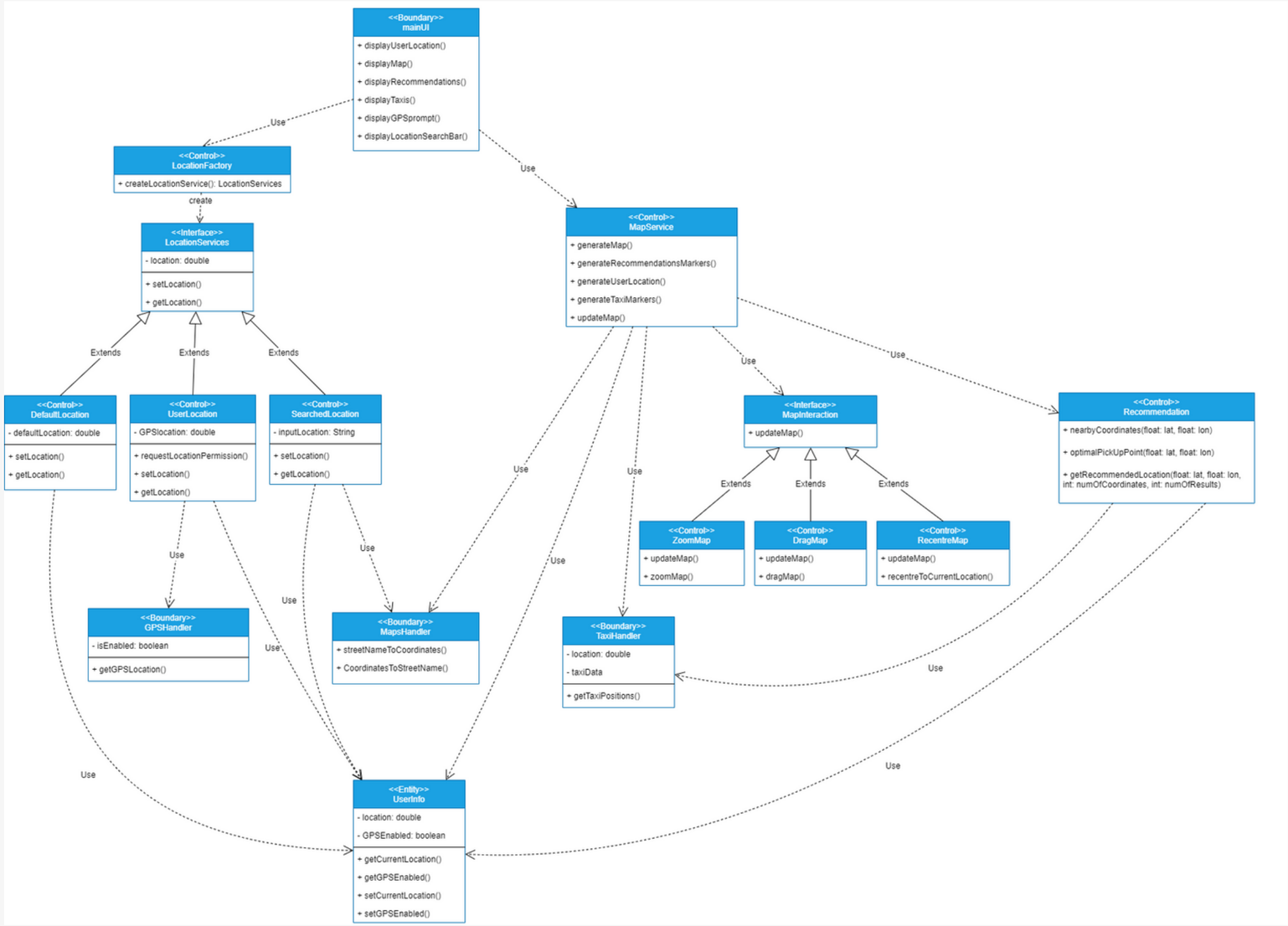
Concurrency and Synchronization

- In locationAPI.go, we have used a mutex (sync.Mutex) to synchronize access to shared variables latestLat, latestLng, and locations.
- Using a mutex ensures that only one goroutine can access the shared data at a time, preventing race conditions and maintaining data integrity.
- Proper synchronization is crucial in concurrent systems to avoid data corruption and unexpected behavior.

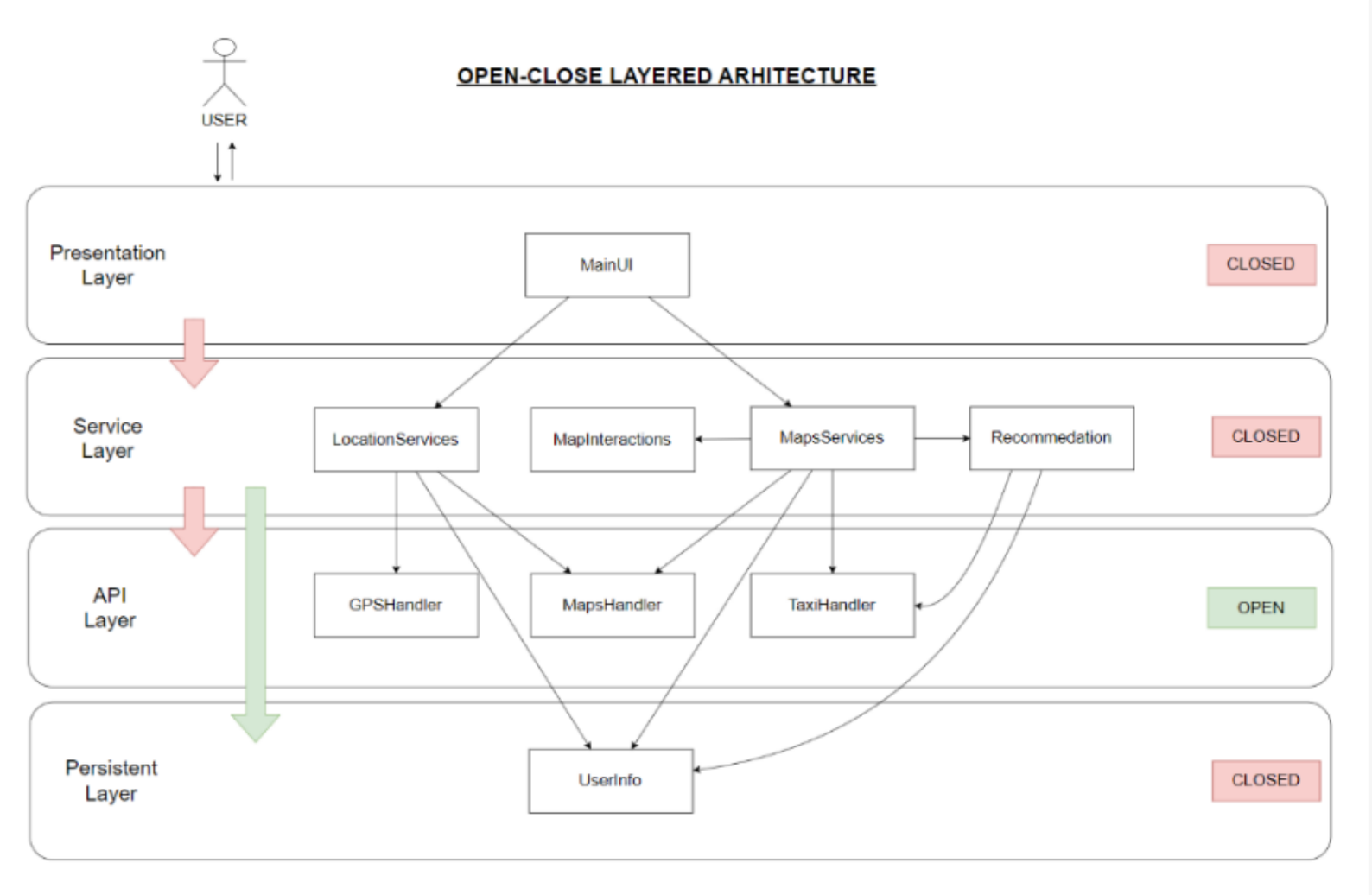
SYSTEM DESIGN

1. SYSTEM ARCHITECTURE
2. CLASS DIAGRAM
3. OVERVIEW DIAGRAMS
4. SEQUENCE DIAGRAMS

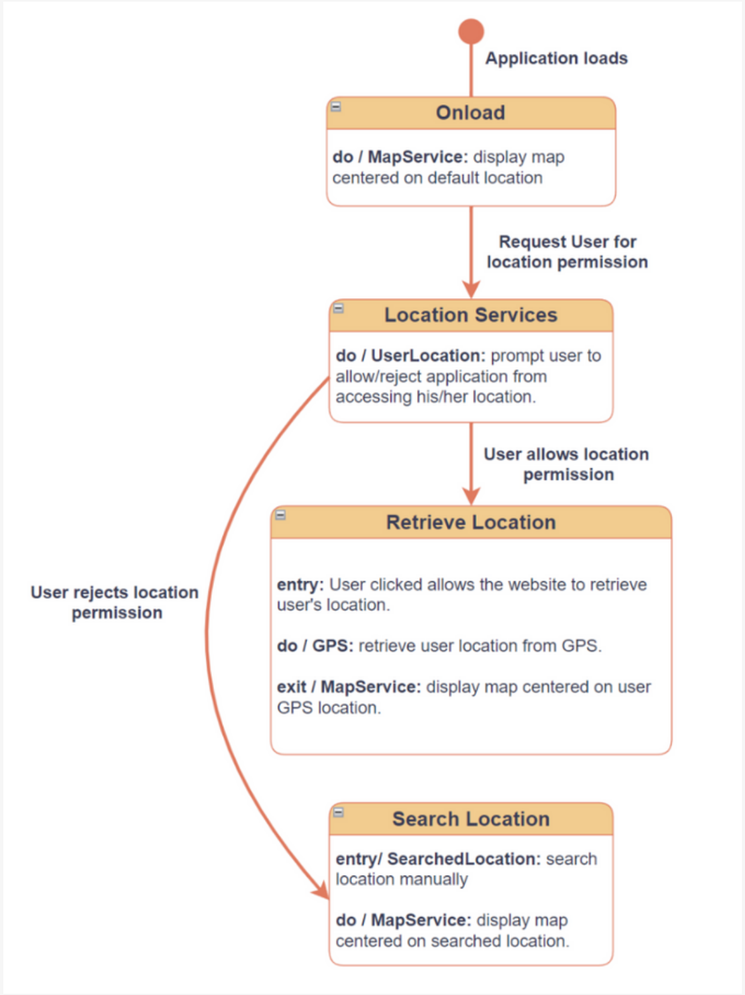
CLASS DIAGRAM

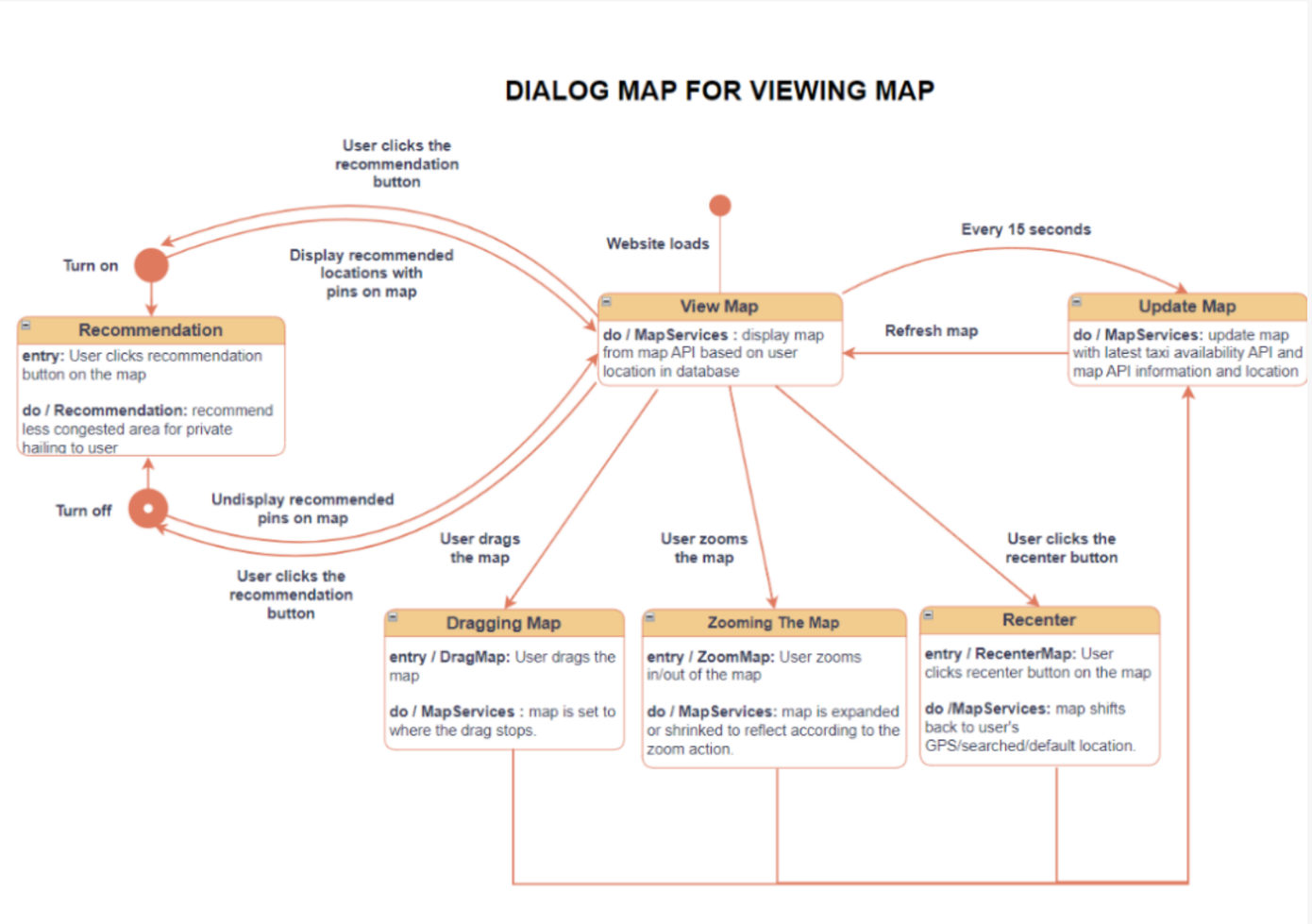


SYSTEM ARCHITECTURE

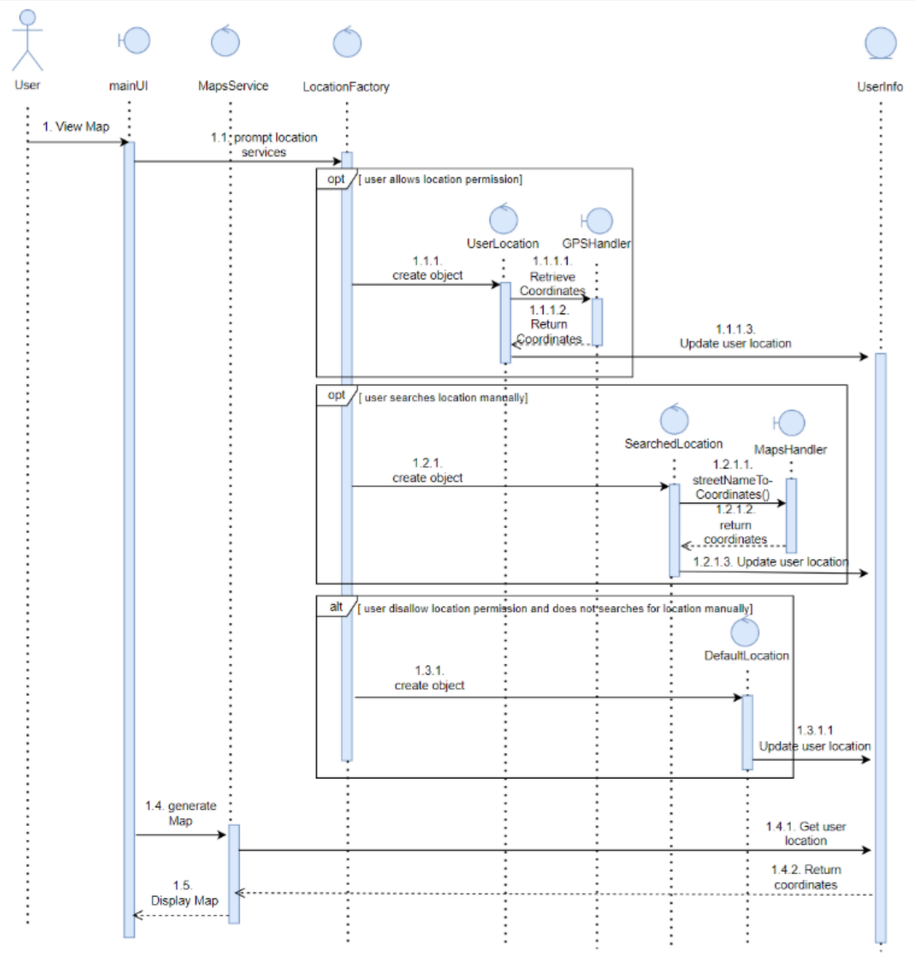


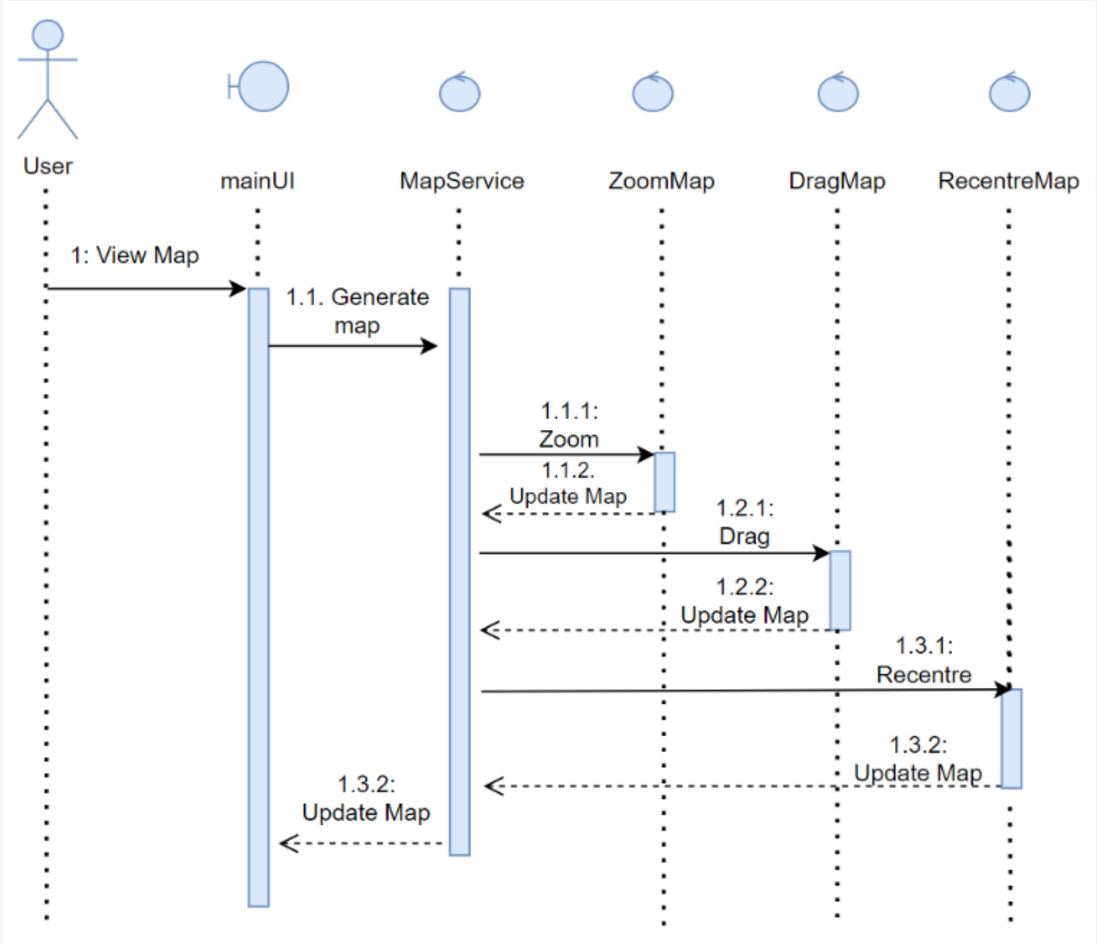
OVERVIEW DIAGRAM



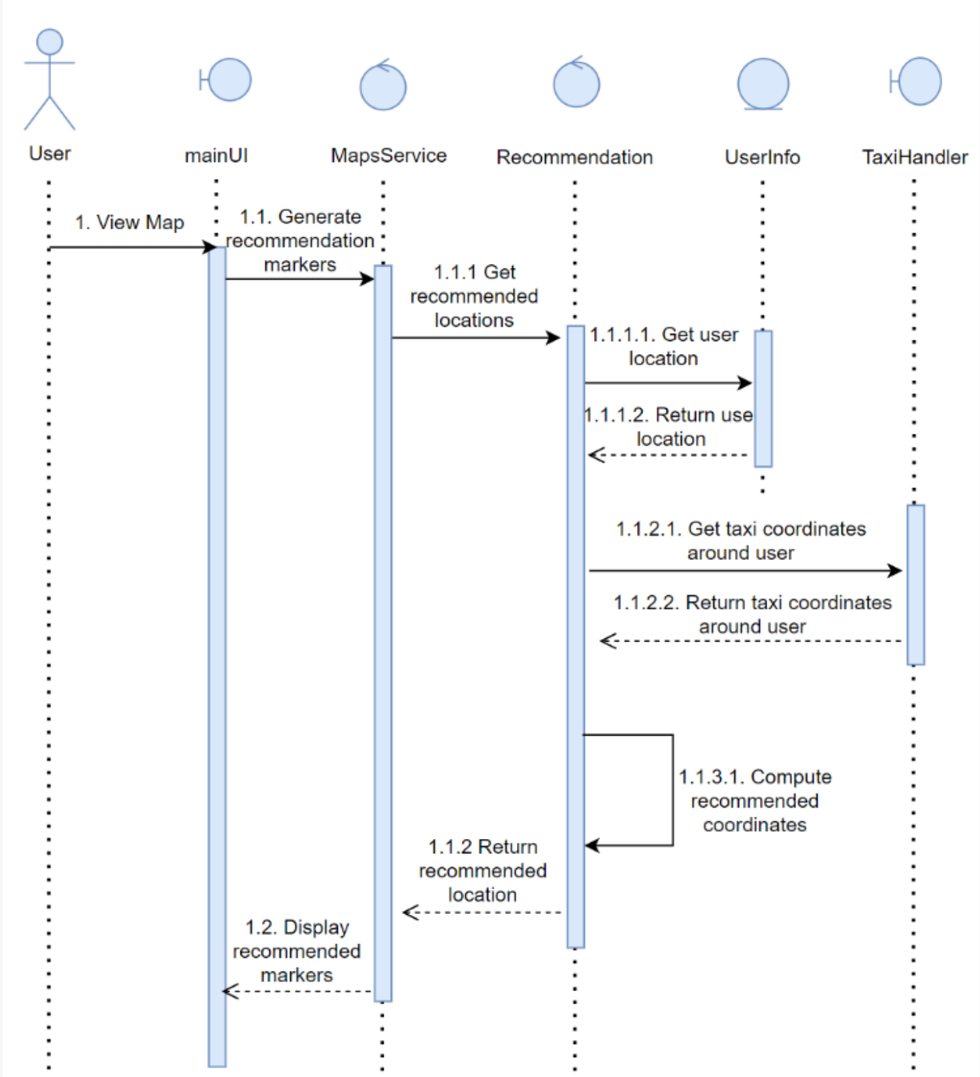


SEQUENCE DIAGRAM





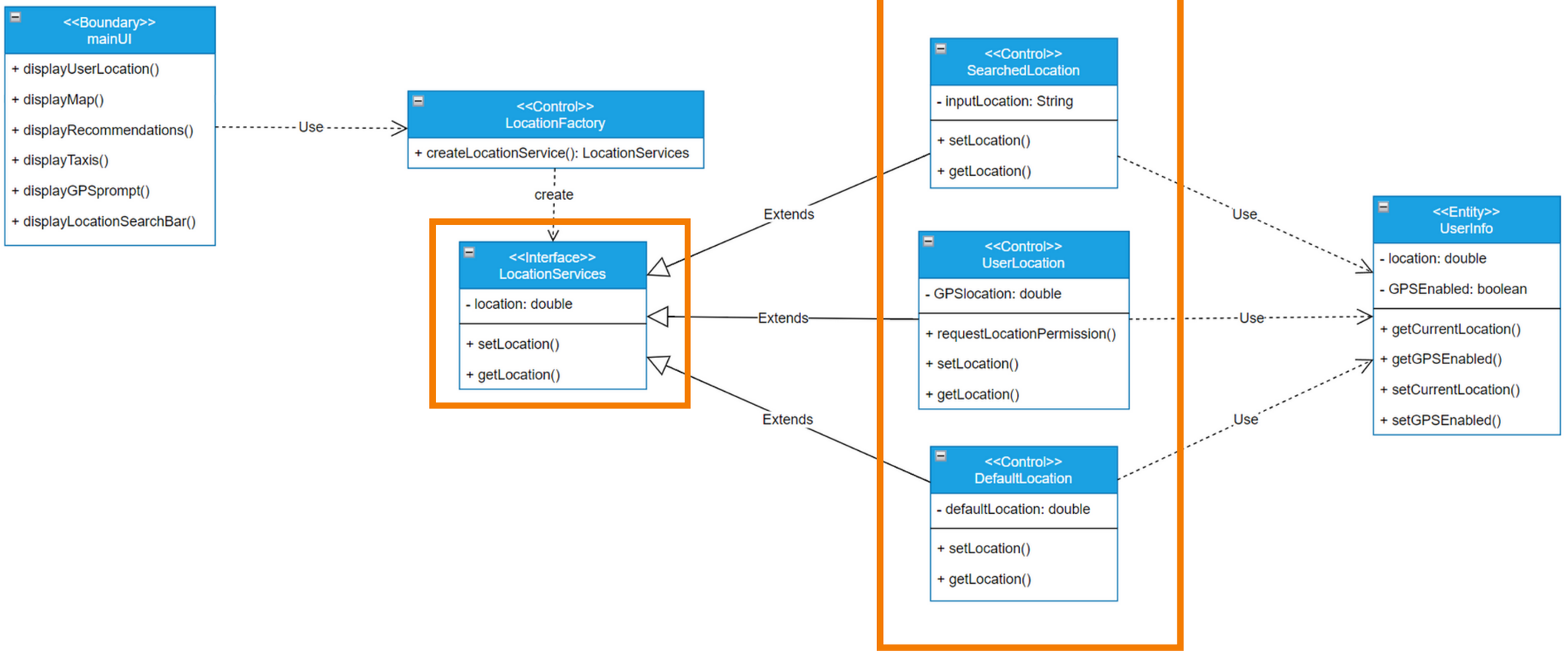
SEQUENCE DIAGRAM



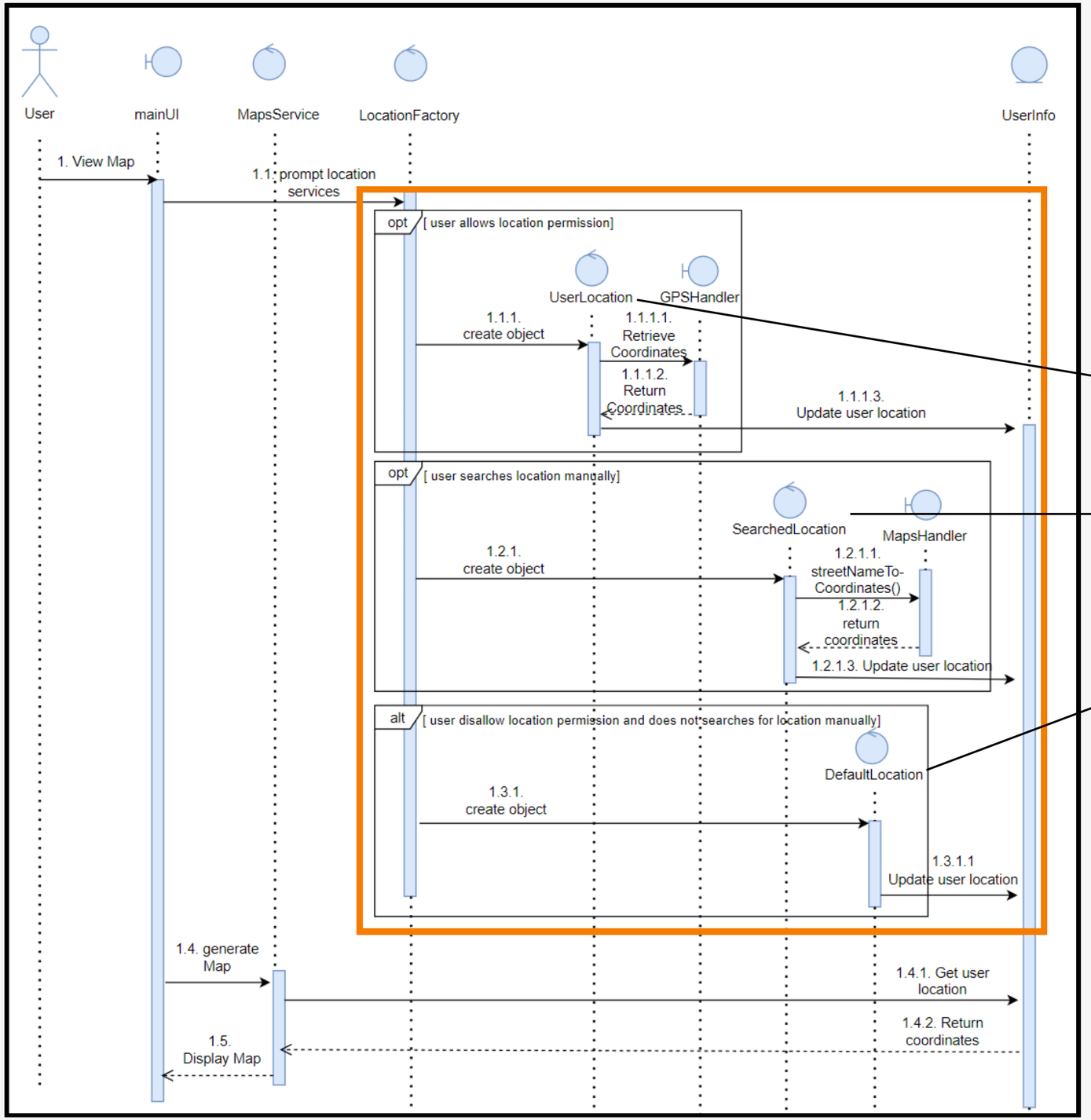
Highlighted Use Cases

Setting Location on Map

Factory Design Pattern

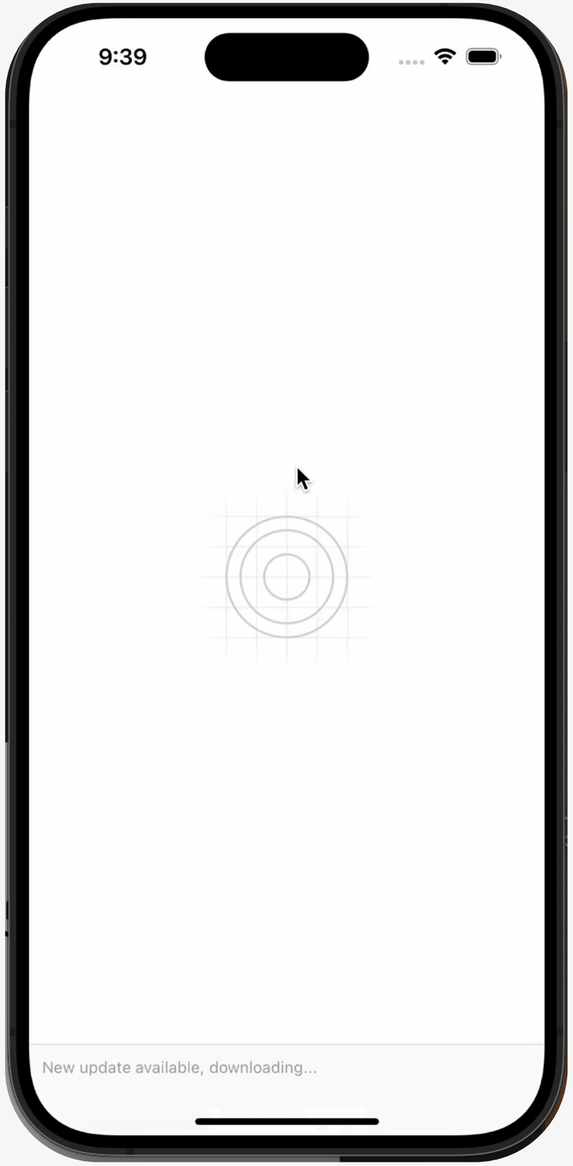


Factory Design Pattern

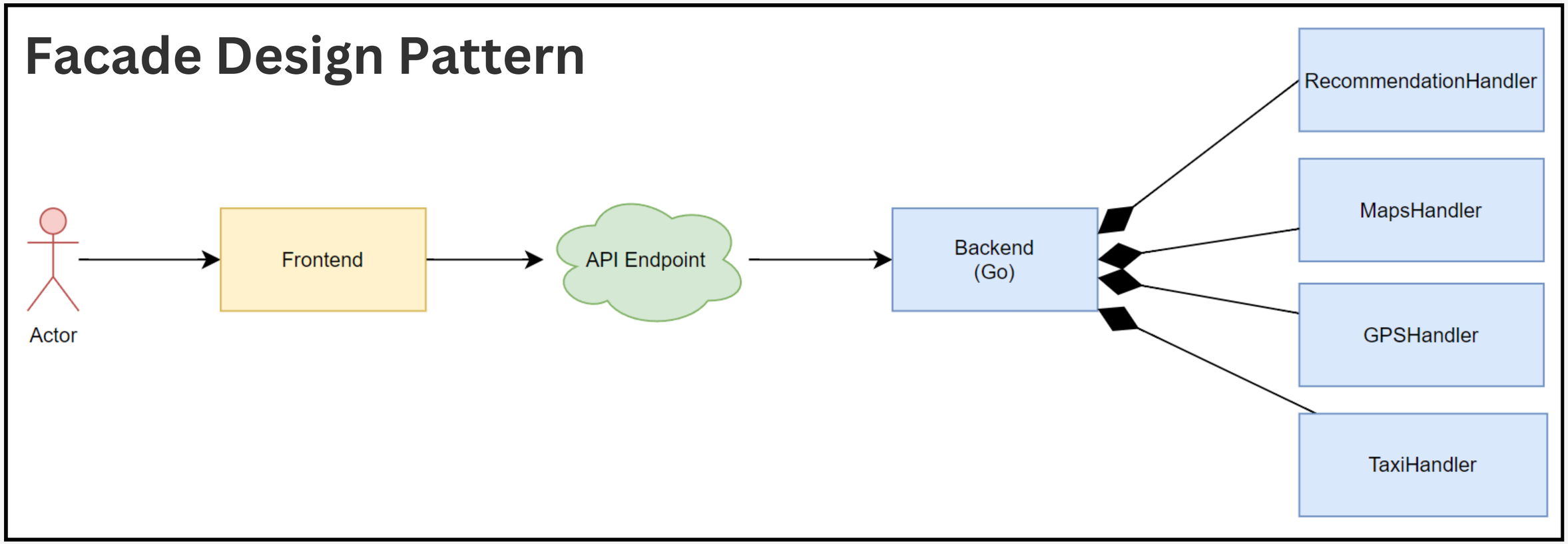


Each object performs a different purpose depending on what action was taken by the user

When GPS Location Service Is Disallowed



Backend Integration



Traceability

```go
// @title Backend APIs
// @version 1.0
// @description APIs for frontend use.
// @BasePath /
func main() {
    // Initialize Gin router
    r := gin.Default()

    // Define routes
    r.GET("/taxi", handleTaxiCoordinates)
    r.GET("/recommendations", getRecommendedLocationsHandler)

    locationGroup := r.Group("/location")
    {
        locationGroup.Handle("GET", "/", handleLocation)
        locationGroup.Handle("PUT", "/", handleLocation)
    }

    mapsGroup := r.Group("/maps")
    {
        mapsGroup.GET("sntocoords", handleStreetNameToCoordinates)
        mapsGroup.GET("coordstosn", handleCoordinateToStreetName)
    }

    r.GET("/docs/*any", ginSwagger.WrapHandler(swaggerFiles.Handler))
    r.GET("/example", handleExample)
    // Start the HTTP server using Gin
    log.Fatal(r.Run("0.0.0.0:80"))
}
```

GO taxiAPI.go

GO recommendation.go

GO locationAPI.go

GO mapsAPI.go

Backend APIs 1.0

[Base URL: /]

doc.json

APIs for frontend use.

Location API

GET /location Get latest location

PUT /location Update location

Maps API

GET /maps/coordstosn Convert coordinates to street name

GET /maps/sntocoords Convert street name to coordinates

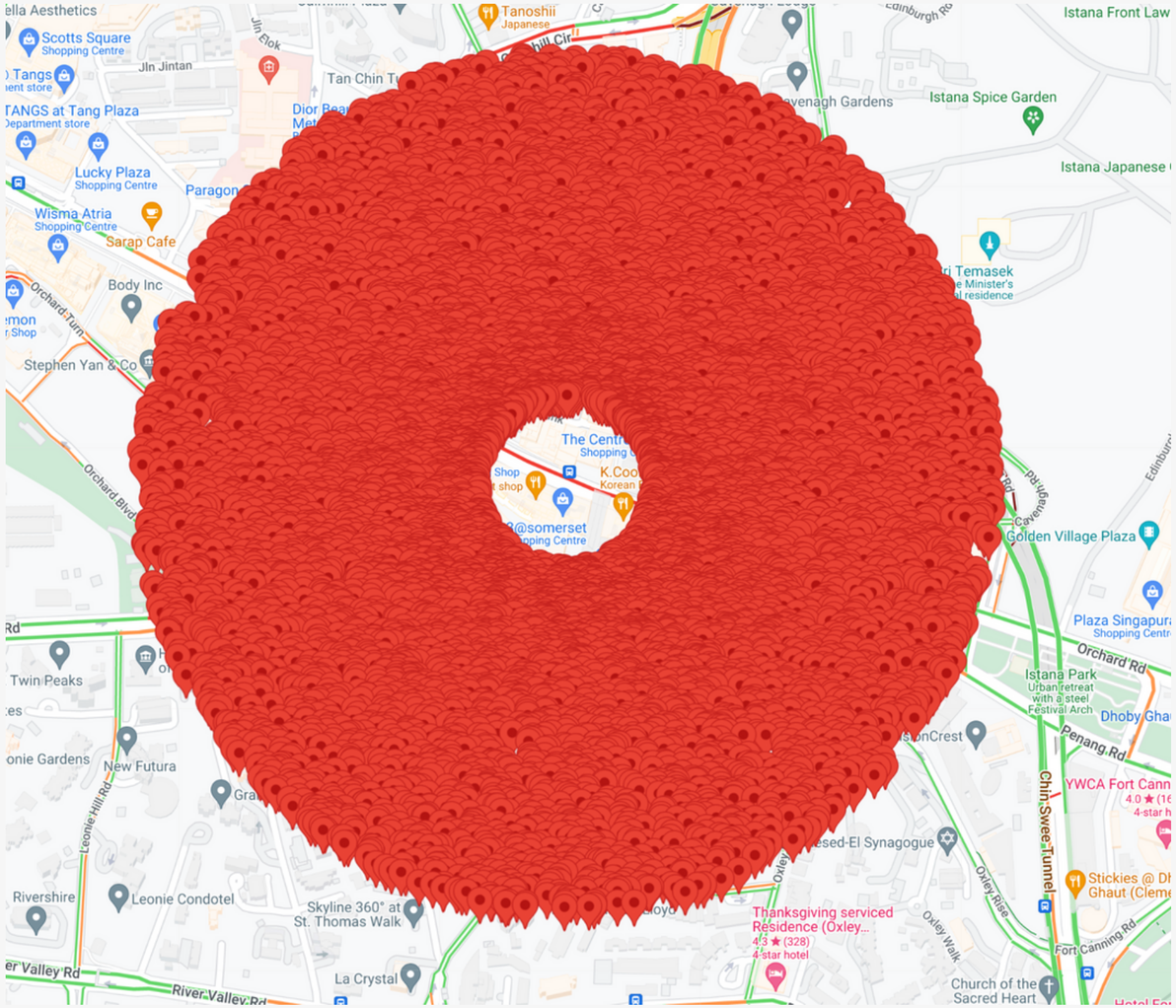
Recommendation API

GET /recommendations Get Recommended Locations

Taxi API

GET /taxi Get taxi coordinates within 5km of a given coordinate

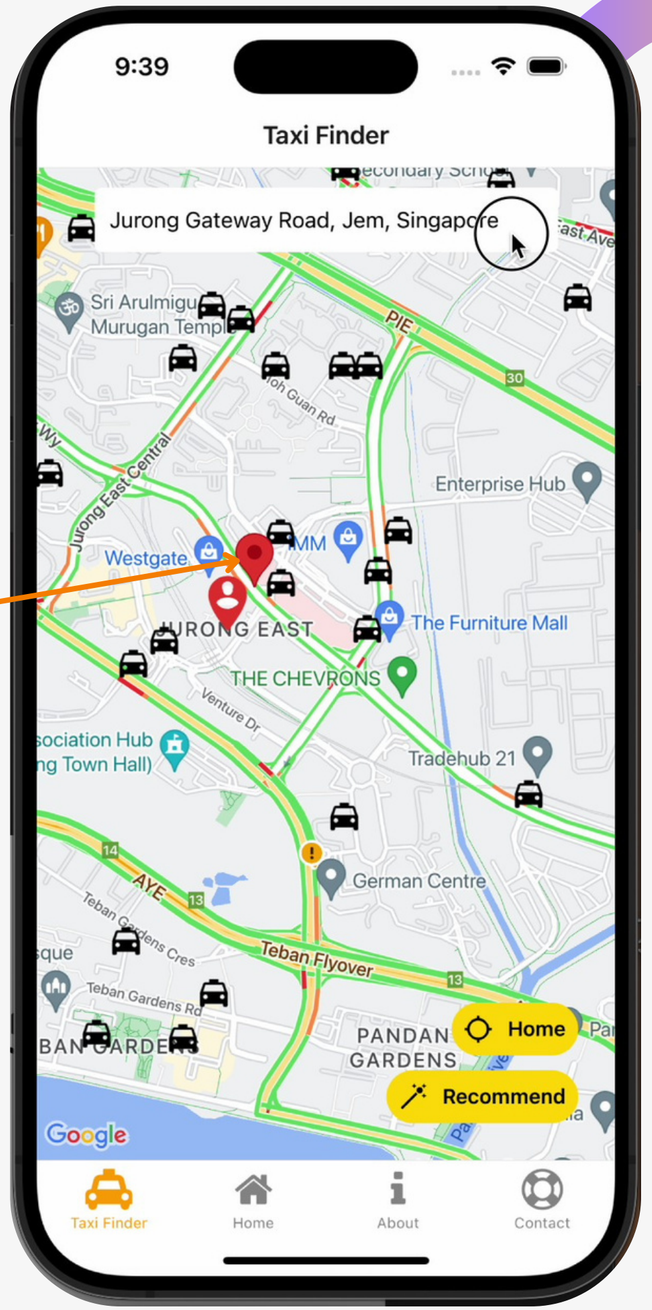
Recommendation Feature

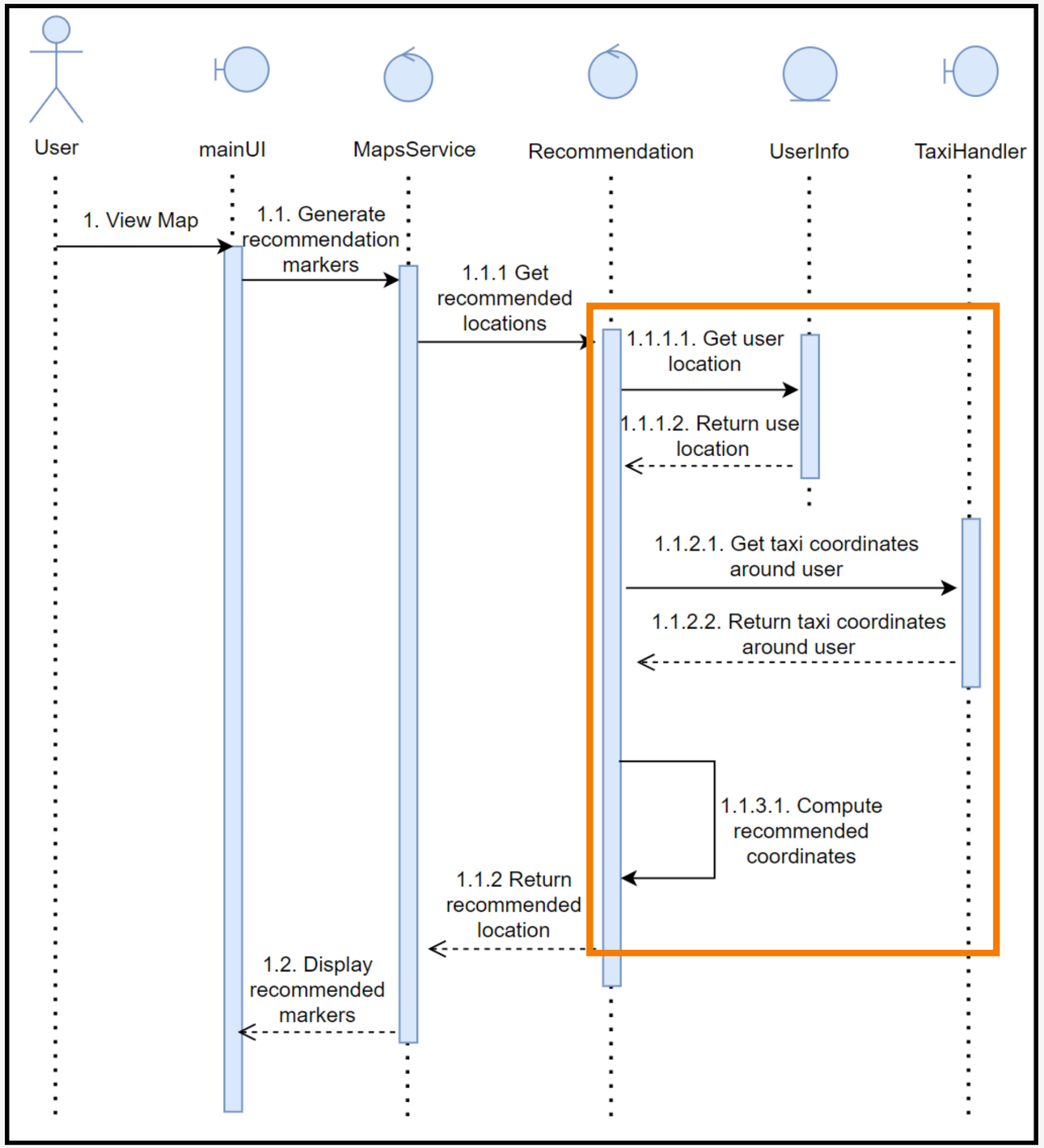


- We generate **10,000** random coordinates around user.
- Compute sum of distance of each coordinate to taxis within **5km** proximity.
- Sort out the top 25 coordinates.

- Filter top 25 coordinates and get the **travel duration** to a centralised location in Singapore.

- Filter **top 5 optimal coordinates** and recommend them; coordinates that are close will be merged together.




User
mainUI
MapsService
Recommendation
UserInfo
TaxiHandler
1. View Map
1.1. Generate recommendation markers
1.1.1 Get recommended locations
1.1.1.1. Get user location
1.1.1.2. Return use location
1.1.2.1. Get taxi coordinates around user
1.1.2.2. Return taxi coordinates around user
1.1.3.1. Compute recommended coordinates
1.1.2 Return recommended location
1.2. Display recommended markers

Thank You