Globetrotter

Title: Design Diagrams for the MSD App

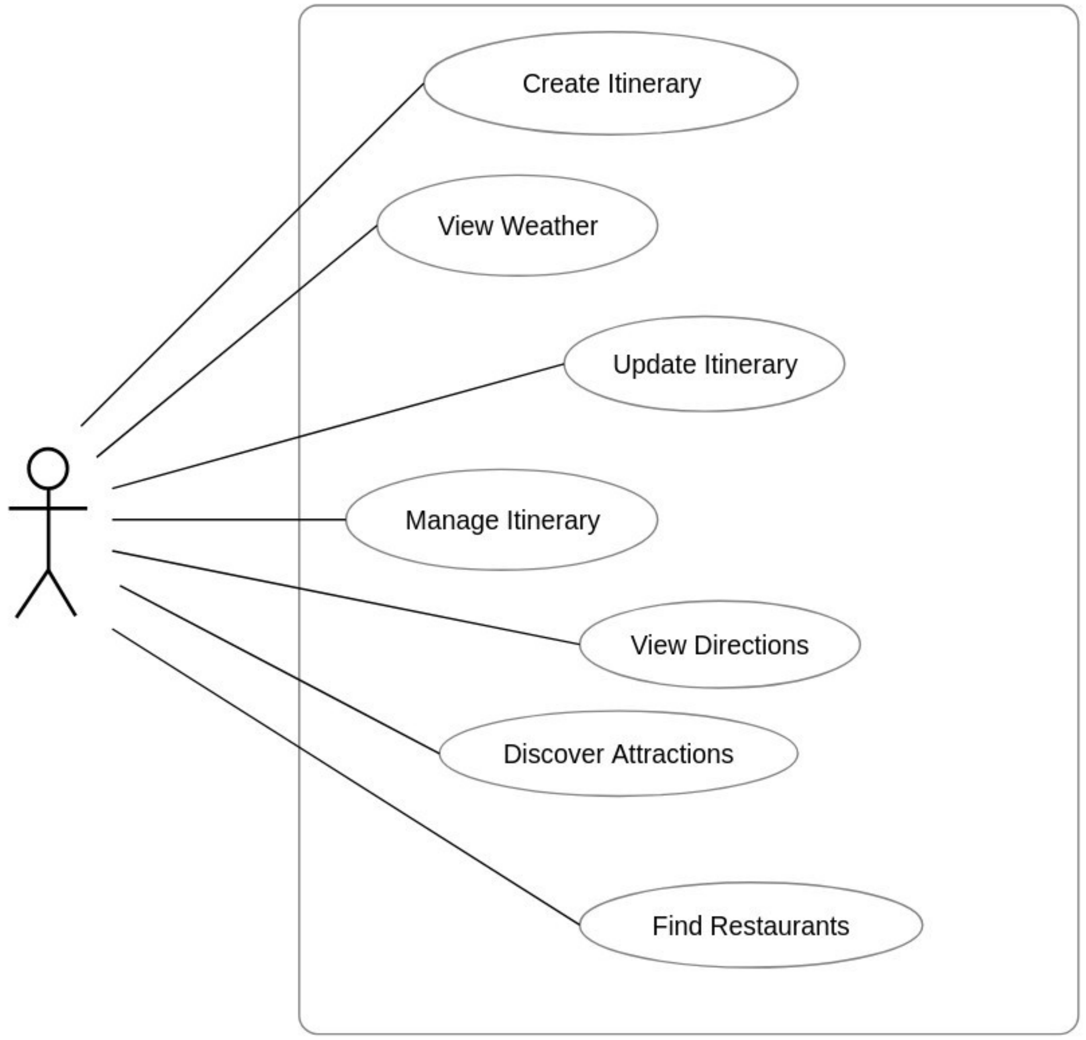
By Elena Juarros González

Mobile Software Design

1.- Purpose of the App

- Globetrotter is a travel planning app designed to make exploring new destinations easier. It enables users to create and manage itineraries, discover attractions, find restaurants, view local weather, and get directions, all in one place. With features like an interactive itinerary builder and lists of must-visit spots, Globetrotter helps users to make the most of their travel experiences. Whether you're planning a weekend getaway or a long-term adventure, Globetrotter is your ultimate travel companion.
- **Problem addressed:** provide single platform for organizing travel plans.
- **Main Features:**
    - Create, edit and manage trip plans with daily activities and notes.
    - Find local attractions, restaurants and points of interest near destinations.
    - Visualize trip routes, attractions and directions using Google Maps (embedded maps).
    - View real-time weather updates for each destination.
- **Target Users:** frequent travelers, tourists and trip planners.
- **User Benefits:** organized trip planning, centralized source of information, and optimize daily travel plans.

2.- Use-Case Diagram



2.1- Use Cases

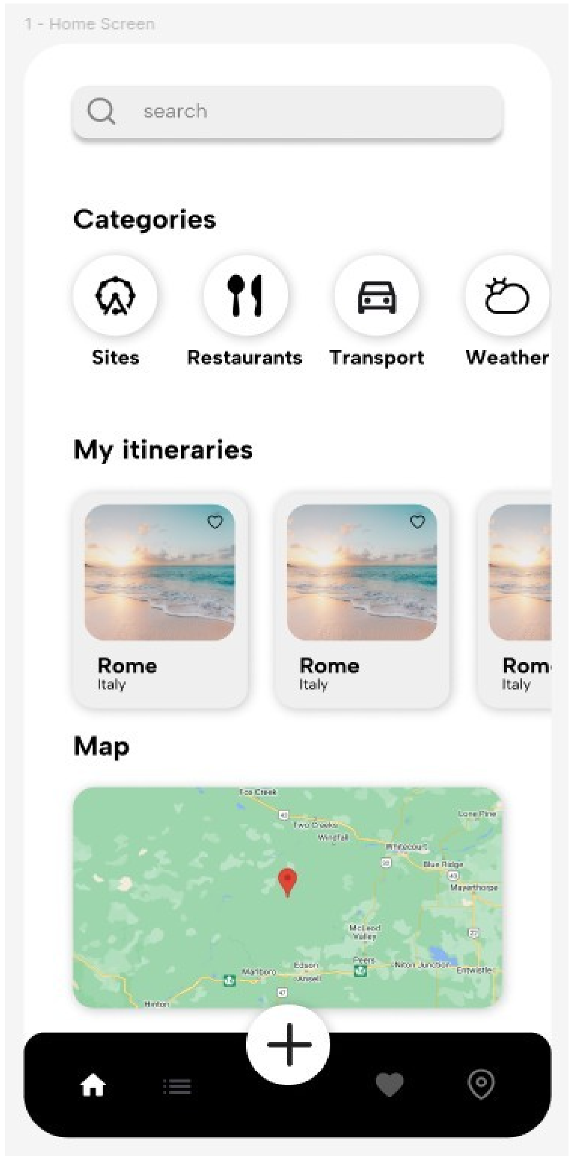
- **Create Itinerary**: allows the user to create a new travel itinerary specifying trip details (destination, dates, ...).
- **Update Itinerary**: allows the uer to modify details of an existing itinerary, adjusting dates or adding new activities.
- **Manage Itinerary**: lets the user view, organize and delete existing itineraries from their list.
- **Discover Attractions**: provides a list of attractions for a selected location.
- **Find Restaurants**: offers a list of restaurants in the selected destination.
- **View Directions**: displays navigation directions to a selected location, attraction or restaurant.
- **View Weather**: shows current weather forecast for the selected destination.

3.- Screen flow

- This is a video showing the navigation flow between screens.

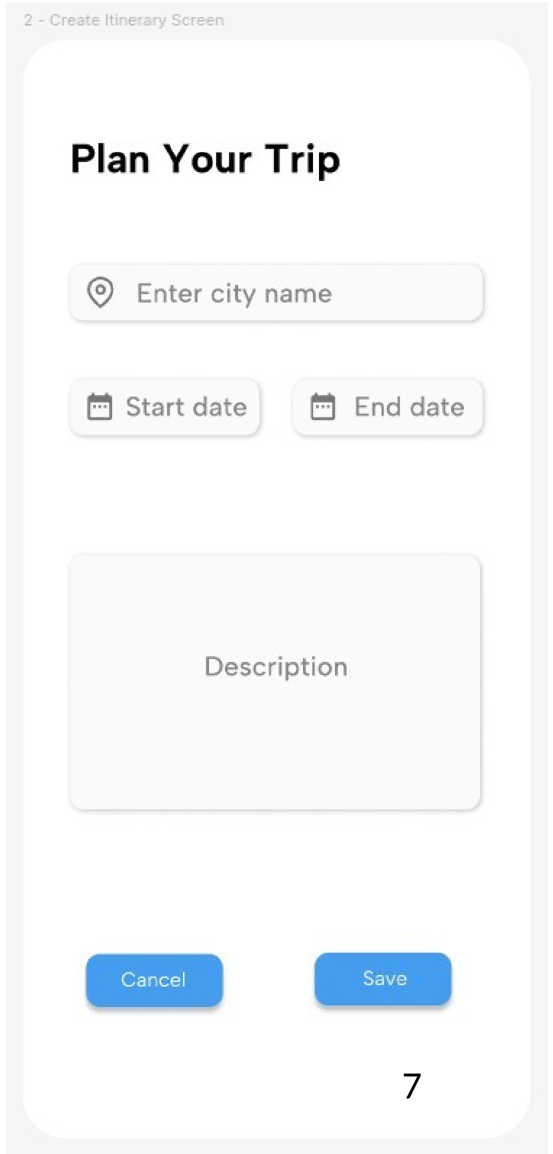
3.- Screen flow. Home Screen

- This is the main screen
- Search bar to filter itineraries
- Categories horizontal recycler view
- Itineraries horizontal recycler view
- Map with itineraries marked
- Navigation bar with buttons to go to other screens



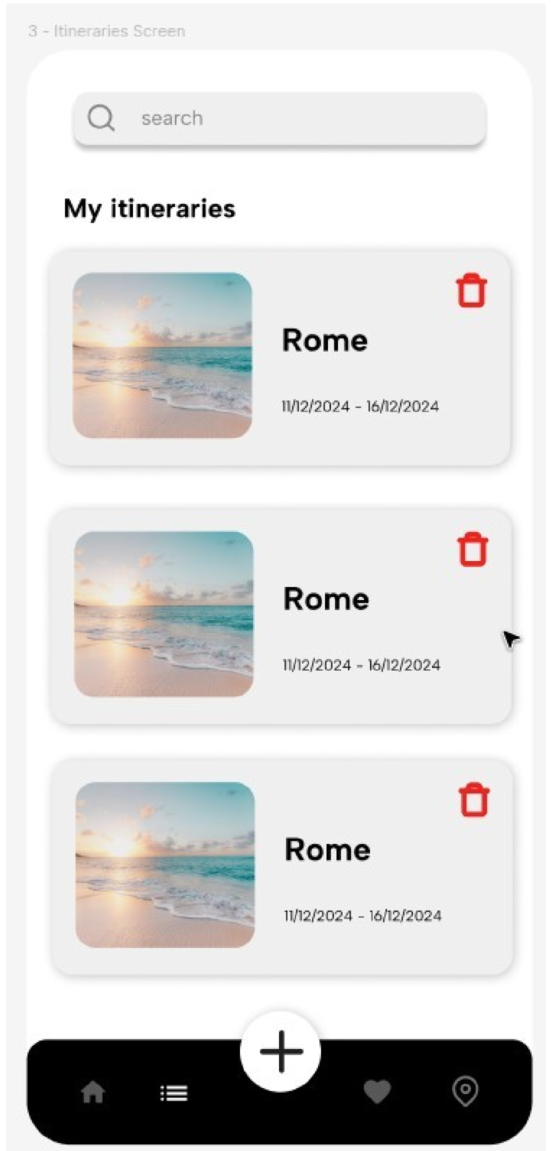
3.- Screen flow. Create Itinerary Screen

- This is the input screen
- EditText for inputting information
- Dates EditText opens a calendar to select date
- Buttons to cancel or save itinerary
- After saving an itinerary recycler views will be updated



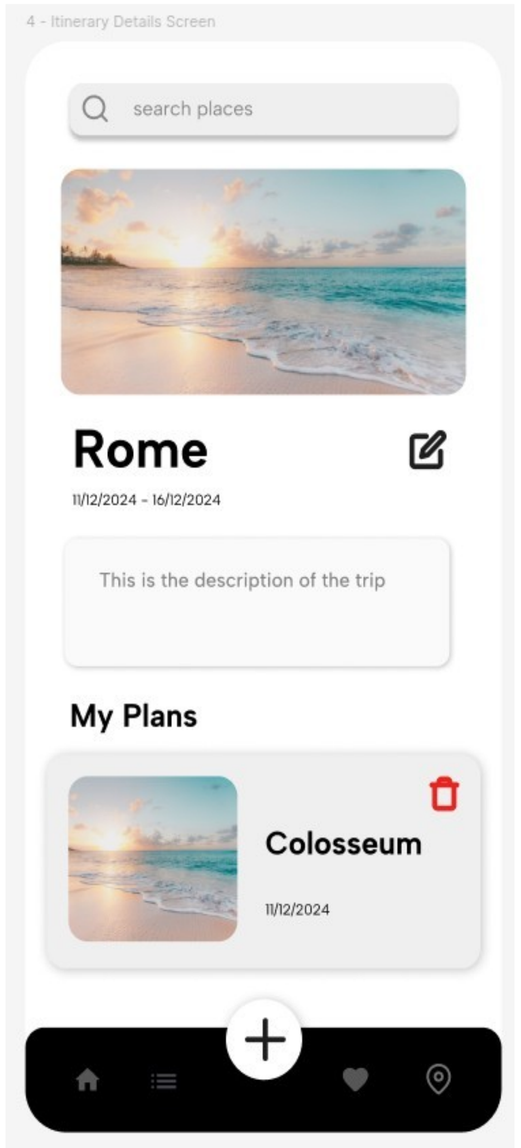
3.- Screen flow. Itineraries Screen

- This is the itineraries screen
- Search bar to filter itineraries by name
- RecyclerView to show itineraries
- Delete button to delete an itinerary
- Navigation bar with buttons to go to other screens



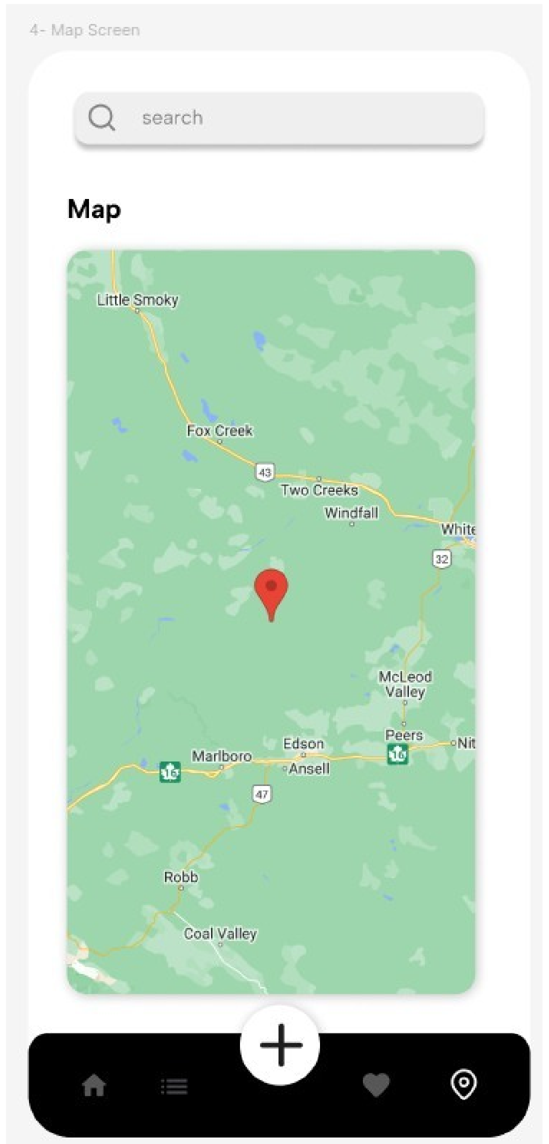
3.- Screen flow. Itinerary Details Screen

- This is the itinerary details screen
- Search bar to search for attractions or restaurants
- Itinerary information
- Edit button to modify an itinerary
- RecyclerView to show attractions or restaurants ordered by date
- Navigation bar with buttons to go to other screens



3.- Screen flow. Map Screen

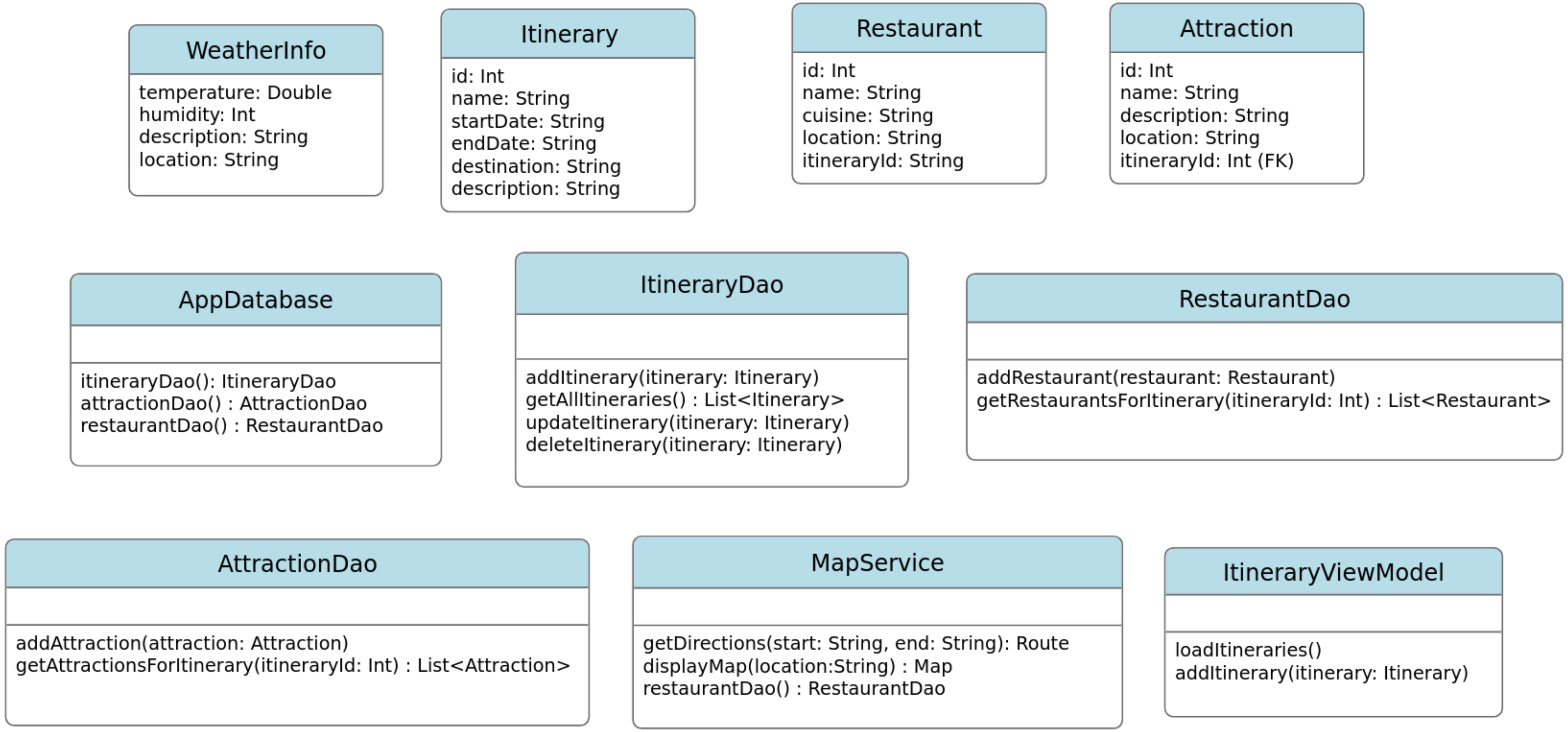
- This is the map screen
- Search bar to search for places
- Interactive map
- Navigation bar with buttons to go to other screens



3.- Screen flow. Other Screens

- **Categories Screen:**
  - Search bar to find places of such category
  - RecyclerView with items of that category
- **Attraction / Restaurant Screen:**
  - Details about the attraction
  - Weather embedded with live forecast for the day of that activity

4.- Class Diagrams


WeatherInfo
temperature: Double
humidity: Int
description: String
location: String
Itinerary
id: Int
name: String
startDate: String
endDate: String
destination: String
description: String
Restaurant
id: Int
name: String
cuisine: String
location: String
itineraryId: String
Attraction
id: Int
name: String
description: String
location: String
itineraryId: Int (FK)
AppDatabase
itineraryDao(): ItineraryDao
attractionDao() : AttractionDao
restaurantDao() : RestaurantDao
ItineraryDao
addItinerary(itinerary: Itinerary)
getAllItineraries() : List<Itinerary>
updateItinerary(itinerary: Itinerary)
deleteItinerary(itinerary: Itinerary)
RestaurantDao
addRestaurant(restaurant: Restaurant)
getRestaurantsForItinerary(itineraryId: Int) : List<Restaurant>
AttractionDao
addAttraction(attraction: Attraction)
getAttractionsForItinerary(itineraryId: Int) : List<Attraction>
MapService
getDirections(start: String, end: String): Route
displayMap(location:String) : Map
restaurantDao() : RestaurantDao
ItineraryViewModel
loadItineraries()
addItinerary(itinerary: Itinerary)

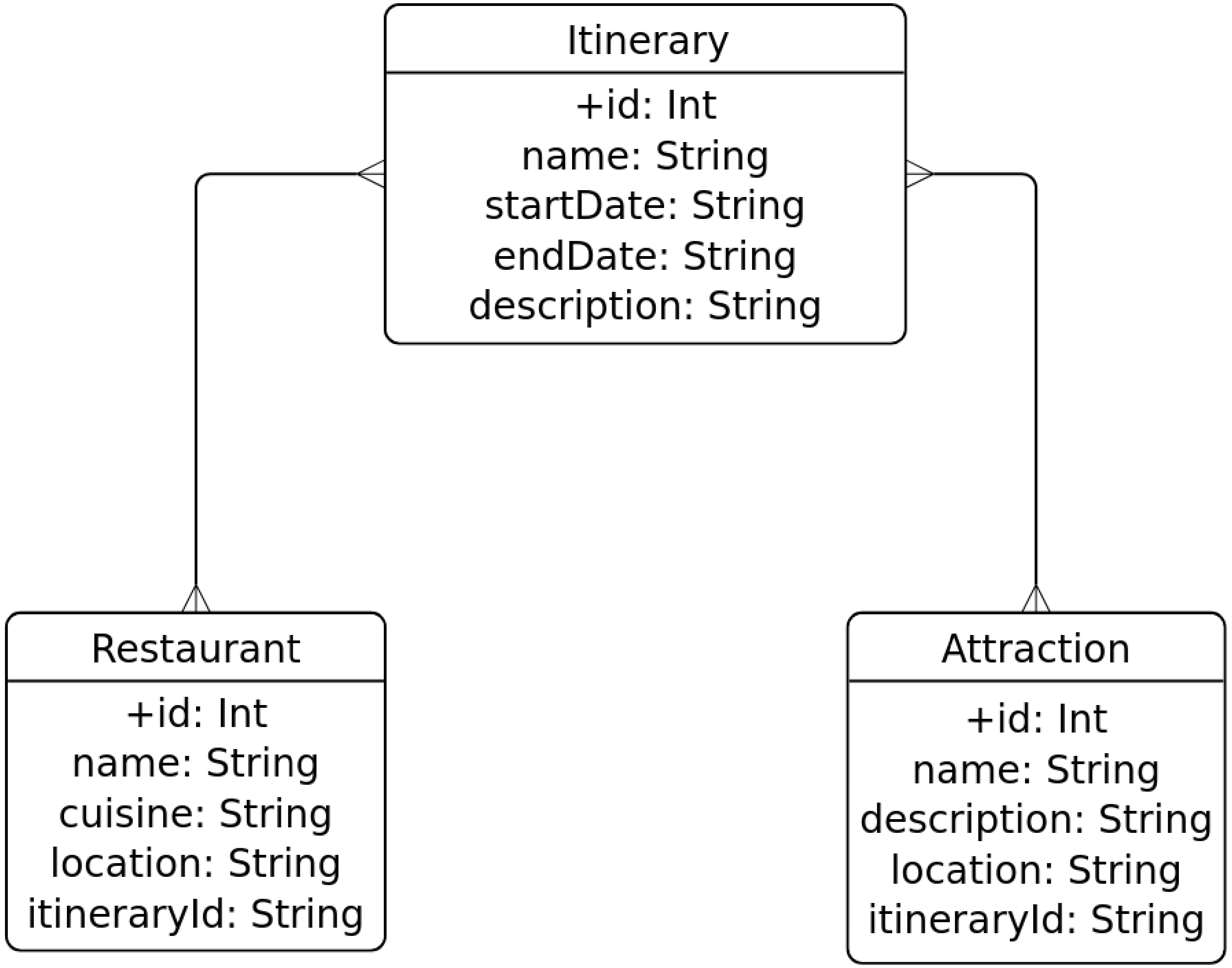
4.- Class Diagrams Description

- **Itinerary Class**: represents the main entity for storing trip details.
- **Attraction Class**: represents attractions related to an itinerary.
- **Restaurant Class**: represents restaurants for an itinerary.
- **WeatherInfo Class**: represents weather related information.
- **AppDatabase Class**: defines the Room database for the app.
- **ItineraryDao, AttractionDao and RestaurantDao Interfaces**: data access objects for interacting with database.
- **MapService**: manages map-related operations like directions and navigation.
- **ItineraryViewModel Class**: connects the UI to database and API data.

4.- Class Diagrams Relationships

- AppDatabase contains Itinerary, Restaurant and Attraction.
- DAOs define how each table in the database is accessed.
- Itinerary is linked to Attraction and Restaurant through a foreign key (itineraryId). It is accessed via the ItineraryDao.
- Restaurant belongs to Itinerary through a foreign key (itineraryId). It is accessed via the RestaurantDao.
- Attraction belongs to Itinerary through a foreign key (itineraryId). It is accessed via the AttractionDao.
- MapService gets directions and map data for an Itinerary

5.- Database Design



5.- Database Design

- **Itinerary Table:** store information for each travel plan.
- **Restaurant Table:** store information on restaurants that a user plans to go on a given itinerary.
- **Attraction Table:** store information on specific attractions or point of interest that a user plans to visit on a given itinerary.
- **Relationships:**
    - Itinerary to Restaurant: many-to-many relationship, each itinerary can contain multiple restaurant, and each restaurant can be in multiple itineraries.
    - Itinerary to Attraction: many-to-many relationship, each itinerary can contain multiple attractions, and each attraction can be in multiple itineraries.