Assignment 2 : Designing and Architecting APIs for Uber App

As a software engineer, you have been tasked with designing and architecting APIs for the Uber app. The app is used by riders to request rides and by drivers to accept and complete rides. Your task is to design and architect APIs that will allow riders to request rides, drivers to accept and complete rides, and for the app to track the progress of the ride. Use RESTful principles to design the APIs.

Instructions:

1. Identify the main resources and actions: Identify the main resources and actions that need to be performed for the Uber app. Consider resources such as rides, riders, drivers, and locations, and actions such as requesting a ride, accepting a ride, and completing a ride.
2. Define the endpoints: Based on the resources and actions identified in step 1, define the endpoints for the APIs. Use RESTful principles to define the endpoints. Consider the HTTP methods (GET, POST, PUT, DELETE) and the parameters needed for each endpoint.
3. Design the data models: Design the data models needed for the APIs. Consider the data needed for each resource and action, and the relationships between the resources.
4. Create the API documentation: Use Swagger to create the API documentation. The documentation should include information on each endpoint, the parameters, the responses, and any error codes.
5. Test the APIs: Test the APIs using a tool such as Postman. Make sure the APIs are working as expected and that error handling is working correctly.

Deliverables:

1. A document outlining the main resources and actions needed for the Uber app.
2. A document outlining the endpoints for the APIs, using RESTful principles.
3. Data models for each resource and action.
4. Swagger API documentation for the APIs.
5. Postman tests for the APIs.

Note: Consider security and authentication when designing the APIs. Make sure only authorized users can access sensitive resources.

Introduction:

In this assignment, you will be designing and architecting APIs for the Uber app. The Uber app is a ride-hailing application that connects passengers with drivers through a mobile application. The API will allow users to request a ride, track the ride, and make payments.

Requirements:

1. User Authentication: Users must be able to authenticate themselves before they can access the API. Authentication can be done using OAuth 2.0, which is a widely used standard for user authentication in APIs.
2. Request a ride: Users should be able to request a ride by specifying their current location and their desired destination. The API should return the details of the driver who has accepted the ride request, including their name, phone number, and vehicle information.
3. Track the ride: Users should be able to track the ride in real-time once a driver has accepted the ride request. The API should provide the user with the driver's location and the estimated time of arrival.
4. Payment: Once the ride is complete, users should be able to make payments through the API. The API should provide details of the fare, including the distance traveled and the time taken. The user should be able to pay using their credit card or other payment methods.
5. Feedback and ratings: Users should be able to provide feedback and ratings for the driver after the ride is complete. The API should provide a rating system and a feedback mechanism that allows users to rate the driver based on their experience.

Architecture:

1. API Gateway: The API Gateway will act as a single entry point for all API requests. It will handle authentication, routing, and load balancing. The API Gateway will also handle caching, rate limiting - VIP Customer - Give them best deal, and request throttling.
2. Microservices: The Uber app will be designed using a microservices architecture. Each service will be responsible for a specific functionality, such as ride **request, ride tracking, and payment.**
3. Containerization: Each microservice will be containerized using Docker. This will make it easier to manage and deploy the services.
4. **Service Registry**: The service registry will be used to keep track of the location of each microservice. It will be used by the API Gateway to route requests to the appropriate service.
5. **Data Store**: The data store will be used to store user information, ride details, and payment information. It will be implemented using a NoSQL database such as MongoDB or Cassandra.

The following are some possible API endpoints and corresponding actions for the Uber app:

1. Authentication API:

* Endpoint: /auth
* Actions: POST (login), DELETE (logout), PUT (change password), GET (retrieve user profile)

1. Ride Request API:

* Endpoint: /rides
* Actions: POST (create a new ride request), GET (retrieve ride details), PUT (update ride details), DELETE (cancel ride request)

1. Driver Availability API:

* Endpoint: /drivers
* Actions: POST (set driver availability), GET (retrieve available drivers), PUT (update driver availability)

1. Payment API:

* Endpoint: /payments
* Actions: POST (make payment for a ride), GET (retrieve payment details)

1. User Profile API:

* Endpoint: /users
* Actions: POST (create a new user profile), GET (retrieve user profile), PUT (update user profile), DELETE (delete user profile)

1. Navigation API:

* Endpoint: /navigation
* Actions: POST (get navigation directions for a ride), GET (retrieve current navigation status)

1. Rating API:

* Endpoint: /ratings
* Actions: POST (submit ride rating), GET (retrieve ride ratings)

1. Support API:

* Endpoint: /support
* Actions: POST (create a new support ticket), GET (retrieve support ticket status), PUT (update support ticket status)