SPOTIFY APP CASE STUDY

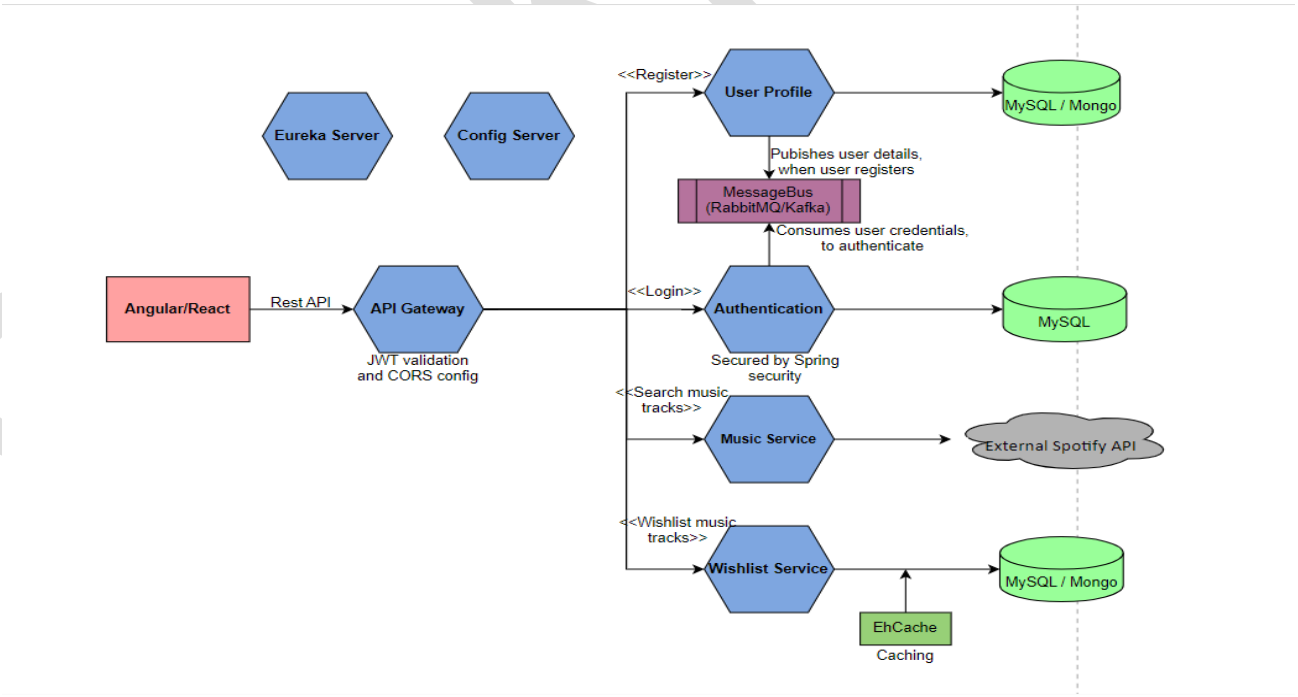
1	**Name of the Project**	**Spotify App**
2	**Objective**	Develop a Music Application that lists the sound tracks for a particular user, allows users to search for music tracks, and save tracks to favorites/wishlist. The application needs to fetch music tracks by registering with the following API and get API Key required to call the API. **https://developer.spotify.com/** **Sample API:** https://api.spotify.com/v1/albums/0sNOF9WDwhWunNAHPD3Baj" -H "Authorization: Bearer {your access token}
3	**Functional Requirements**	1) User Interface (UI) should achieve the following: a) User Registration b) User Login c) List Common Music Tracks d) View a Music Track from the list e) Search for a music track f) Add a music track to your favorite list g) View favorite tracks h) UI should be responsive which can run smoothly on various devices. i) The UI should be appealing and user friendly
4	**Non-functional Requirements**	1) The app should be able to load music tracks quickly and smoothly, even on low-end devices. 2) The app should be able to handle a large number of users without slowing down or crashing. 3) The app should be easy to use and navigate, even for users with no prior experience with music apps. 4) The app should protect user data from unauthorized access, modification, or deletion.
5	**Technical Requirements**	1) Application should be developed using Microservices in the Backend. JWT tokens to be used for securing the Backend. 2) Frontend should be developed using Angular/React. 3) Microservice patterns like API Gateway, Service Discovery, Microservice communication, Configuration Server should be used. 4) Comprehensive Unit tests and integration tests with coverage should be implemented to validate the functionality of the Application. 5) Application should be integrated with SQL databases on Cloud. 6) Application should be deployed on AWS Cloud. 7) SCM like Gitlab to be used for regularly committing the source code. 8) Implement Documentation of API using Swagger/Open API. 9) Application should be integrated with the CI/CD process on AWS.
6	**Tools and Technologies to be used**	SCM : Gitlab Middleware : Spring Boot Frontend : Angular/React Data Store : MySQL Testing : JUnit, Jest/Jasmine CodeQuality : Sonar Lint, JaCoCo CI : Gitlab/AWS/Jenkins API Documentation : Swagger

		Message Bus : RabbitMQ/Kafka Containerization : Docker, Docker Compose Cloud : AWS

User Stories

1	**As a user, I should be able to register with the application so that I can login and use the functionalities of the application.**
2	**As a user, I should be able to login with my username and password in order to access the functionalities of the application.**
3	**As a user, I should be able to search music tracks using Third Party API.**
4	**As a user, I should be able to save music tracks to a wishlist/favourite so that I can access them later.**
5	**As a user, I should be able to access music tracks saved to my wishlist/favourite.**
6	**As a user, I should be able to delete music tracks saved to my wishlist/favourite.**

High Level Architecture Diagram



The responsibilities of the microservices in the above figure are as follows:

- **User Profile Service**: This Service is responsible for storing user registration details. The Service publishes the user credentials sent as part of registration to the message bus and stores the remaining user profile information in the database.
- **Authentication Service**: This Service is responsible for consuming user credential from the message bus and storing it in the database. When a user logs in, this service validates the login credentials against the credentials stored in the database. If the credentials match, this service generates a JWT token and sends back as response, else an error message is sent.
- **Music Service**: This Service is responsible for accessing an external Spotify API to fetch music based on the search criteria coming in as a request and returning back the list of matching music tracks as response.
- **Wishlist Service**: This Service is responsible for storing music tracks bookmarked by users in the database.
- **API Gateway**: This Service acts as the entry point of the system. It intercepts all the requests and validates the JWT Token before routing it to the appropriate microservices.
- **Eureka Server**: This Service acts as a service registry where all the other microservices registers during startup for discoverability.
- **Config Server**: This Service acts as a centralized location to store the configuration of the other microservices of the system.

Recommended Steps to complete the Case Study

Step 1: Understand the Case Study
Step 2: Identify the Data Model and draw the data flow diagram
Step 3: Draw the UI Wireframes
Step 4: Create the Boilerplate
Step 5: Setup CI/CD Pipeline
Step 6: Implement and write test cases for the backend
Step 7: Implement and write test cases for the frontend
Step 8: Integrate the frontend with the backend
Step 9: Dockerize all services of the application
Step 10: Configure Docker Compose for Container Orchestration
Step 11: Deploy to Cloud

Deployment

i) The backend services and frontend services should be deployed on EC2 instances.
ii) Cloud SQL Databases to be used.