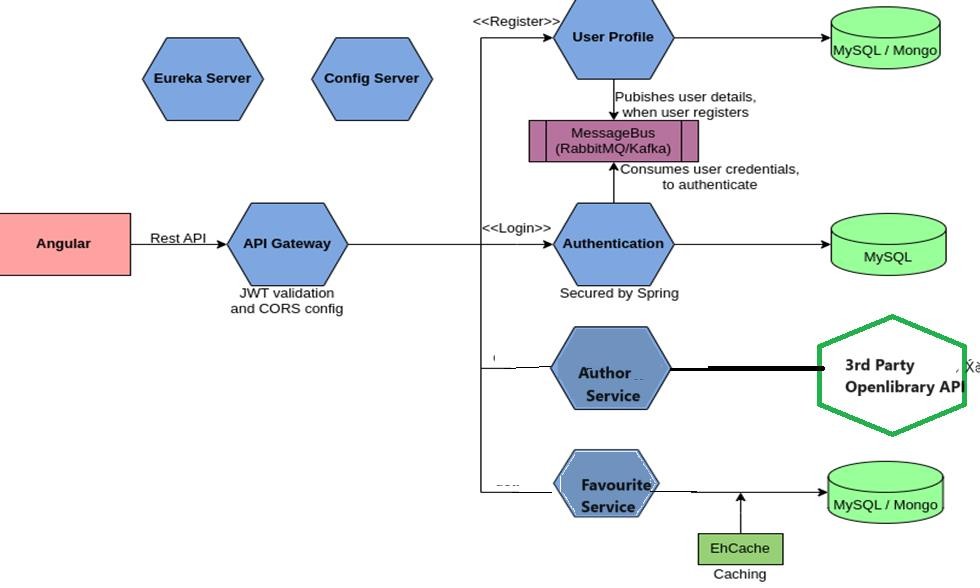
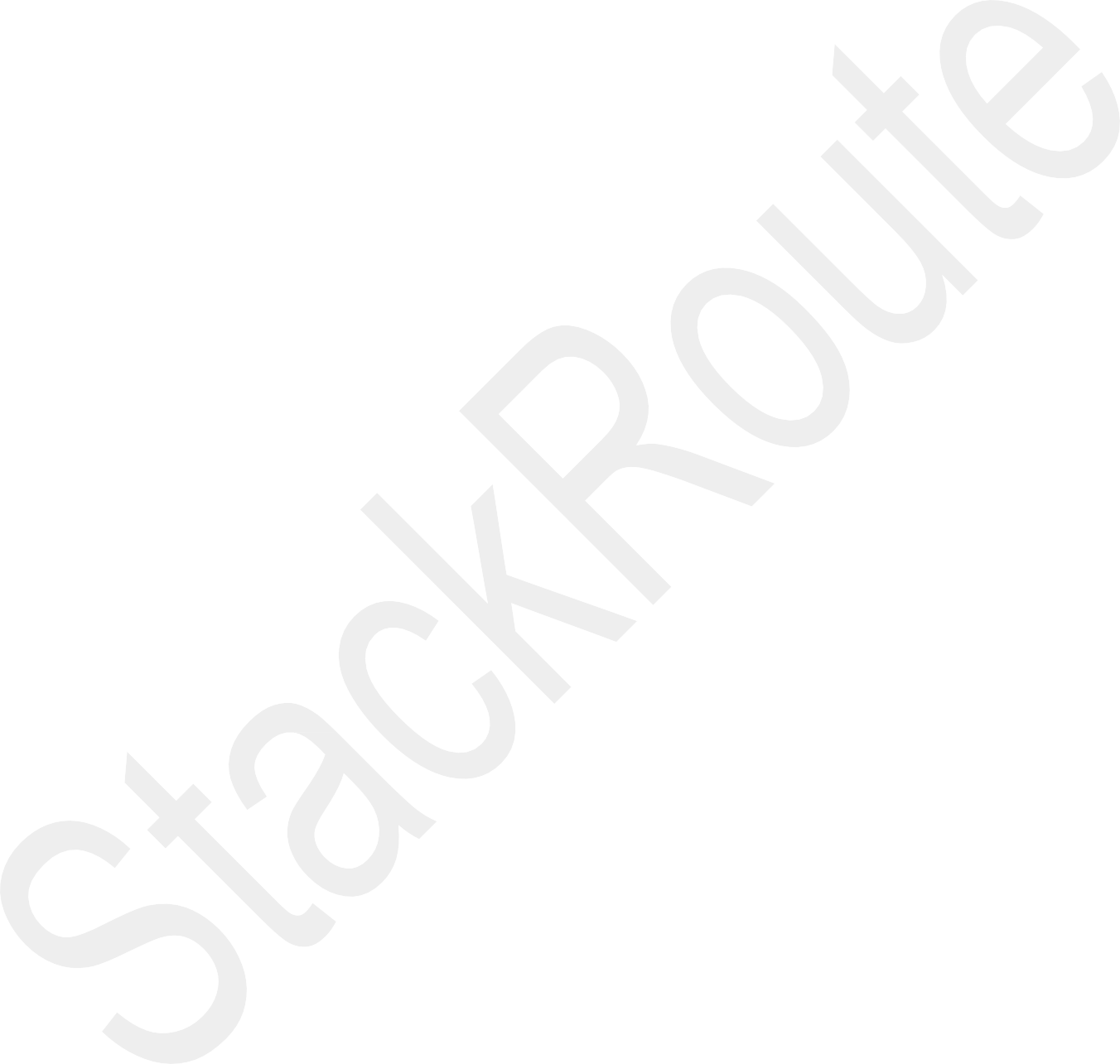
AuthorFinder App CASE STUDY

|  |  |  |
| --- | --- | --- |
| **1** | **Name of the Project** | **Authorfinder App** |
| **2** | **Objective** | Develop an application that allows users to view the authors and their journals and add their favourite authors details  The application needs to fetch details by registering with the following API  https://openlibrary.org/dev/docs/api/authors  **Sample API:** https://openlibrary.org/search/authors.json?q=twain |
| **3** | **Functional Requirements** | 1. User Interface (UI) should achieve the following:    1. User Registration    2. User Login    3. Search and view the author details    4. Add the author data into favorite list    5. View favorite.    6. UI should be responsive which can run smoothly on various devices.    7. The UI should be appealing and user friendly |
| **4** | **Non-functional Requirements** | 1. The app should be able to load stock details quickly and smoothly, even on low-end devices. 2. The app should be able to handle many users without slowing down or crashing. 3. The app should be easy to use and navigate, even for users with no prior experience with stock 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 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 6. SCM like Gitlab to be used for regularly committing the source code. 7. Implement Documentation of API using Swagger/Open API. |
| **6** | **Tools and Technologies to be used** | SCM : Gitlab  Middleware : Spring Boot  Frontend : Angular  Data Store : MySQL  Testing : JUnit, Jest/Jasmine CodeQuality : Sonar Lint, JaCoCo CI : Gitlab/AWS/Jenkins API Documentation : Swagger  Message Bus : Kafka |

# 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 to access the application's functionalities.** |
| **3** | **As a user, I should be able to view all stocks based on country name using Third Party**  **API.** |
| **4** | **As a user, I should be able to save stock details to a wishlist/favourite so that I can access them later.** |
| **5** | **As a user, I should be able to access stock details saved to my wishlist/favourite.** |
| **6** | **As a user, I should be able to delete stock details 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 matches, this service generates a JWT token and sends back as response, else an error message is sent.
* **Author Service**: This Service is responsible for accessing an external openlibrary api to fetch author details based on any specific criteria as a request and returning back the details of author as response.
* **Favorite Service**: This Service is responsible for storing author details 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.

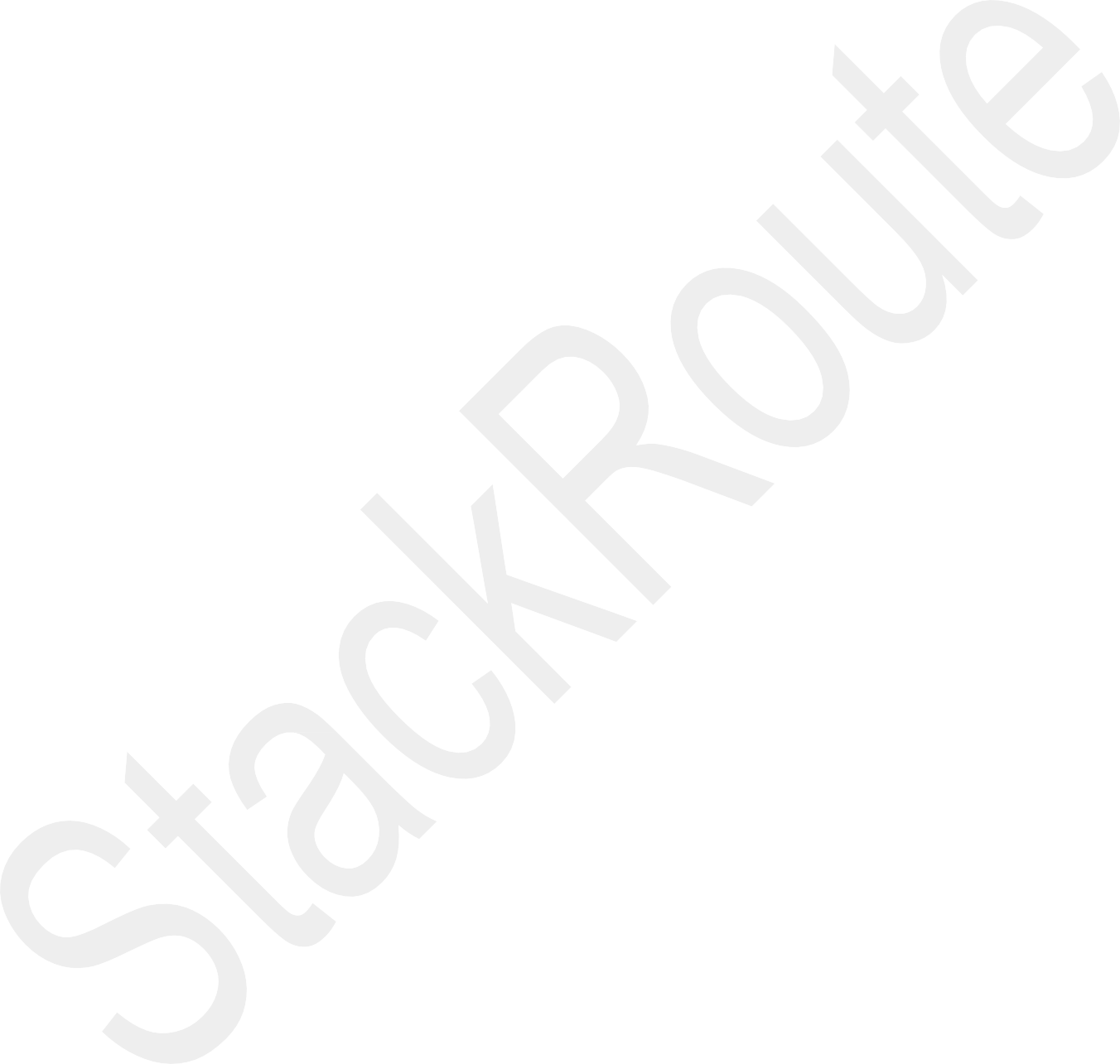
# Recommended Steps to complete the Case Study

**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:** Implement and write test cases for the backend **Step 6:** Implement and write test cases for the frontend **Step 7:** Integrate the frontend with the backend