***Our Project Visual Diagram***

1. **User Interaction Flow :-**

**Welcome Dashboard**

**(LogIn/SignUp)**

**Product Form Page**

**(Add,Update)**

**Registeration User**

**LogIn user**

**Dashboard Open**

**(User/Admin)**

**Product List Show**

**Product Delete**

**LogOut After Main Dashboard Page**

**2.Authantication Flow :-**

**Tocken Validation**

**LogIn Request**

**3.** **Front-end and Back-end Interaction :-**

**AJAX/Fetch Request**

**Error Handling**

**User Registration & Authentication Flow**

**Product Management (CRUD Operations)**

**Front-End Interaction (Routing, Views)**

**Validation and Error Handling**

**Spring Security Config (WebSecurityConfigurer and JWT Filter)**

**JWT Token Generation(POST /auth/login)**

**1. Authentication and Authorization**

1. **User Entity and Repository**
   * **Entity:** User
     + Fields: id, username, password, role
   * **Repository:** UserRepository
     + Methods for CRUD operations
2. **User Service**
   * **Service:** The UserServiceImpl class implements the UserService interface and provides methods for managing user-related operations:
   * **User Registration:** Saves new user details to the database.
   * **User Retrieval:** Retrieves user information based on username.
3. **Spring Security Configuration**
   * + **SecurityConfiguration:** Configure Spring Security to handle authentication and authorization, including setting up JWT-based security.
     + JWTAuthanticationFilter: A custom filter that intercepts HTTP requests to authenticate JWT tokens.
     + IlleagalAccess: To handle cases where access to a resource is forbidden, such as when a user does not have the required permissions.
     + UserReposiotry: Interface for accessing user data from the database.
     + JWTAuthanticationEntryPoint: Entry point used by Spring Security to handle unauthorized access attempts.
4. **JWT Token Generation**
   * **Endpoint:** POST /auth/login
     + Generate JWT upon successful authentication
5. **Role-Based Access Control**
   * **APIs:** Secure endpoints using role-based access
     + Example: Admin routes (POST /products, PUT /products/{id}, DELETE /products/{id})

**2. Product Management**

1. **Product Entity and Repository**
   * **Entity:** Product
     + Fields: productId, name, description, price
   * **Repository:** ProductRepository
     + Methods for CRUD operations
2. **Product Service**
   * **Service:** ProductService
     + Business logic for handling products
3. **Product Controller**
   * **Controller:** ProductController
     + **APIs:**
       - GET /allproducts - Fetch all products
       - POST /products - Add a new product (admin)
       - PUT /products/{id} - Update a product (admin)
       - DELETE /products/{id} - Delete a product (admin)

**3. Front-end Implementation**

1. **Routing and Views**
   * **Pages:**
     + **Register Page:** Register Users
     + **Login Page:** Authenticate users
     + **Dashboard:** User and Admin dashboards based on roles
     + **Product List Page:** Display products
     + **Product Form Page:** Add and update products
     + **Admin Dashboard:** Manage products
2. **AJAX/Fetch for API Calls**
   * **Examples:**
     + Fetch product list
     + Submit product form
     + Handle delete requests
3. **Token Management**
   * **Storage:** JWT tokens stored in local storage
   * **Headers:** Include JWT in request headers for protected APIs

**4. Validation and Error Handling**

1. **Form Validation**
   * **Client-Side:** Validate forms using JavaScript or front-end libraries
   * **Server-Side:** Validate input data in the controllers and services
2. **Error Handling**
   * **Client-Side:**
     + Display error messages for invalid inputs and API errors
   * **Server-Side:**
     + Handle exceptions and return appropriate HTTP status codes and messages

* **Our Project Url given below:-**

1. [**http://localhost:8081/dashboard-page**](http://localhost:8081/dashboard-page)
2. [**http://localhost:808q/login-page**](http://localhost:808q/login-page)
3. [**http://localhost:8081/register-page**](http://localhost:8081/register-page)
4. [**http://localhost:8081/dashboard?Authorization=ZXlKaGJHY2lPaUpJVXpVeE1pSjkuZXlKemRXSWlPaUp5YjJocGRFQm5iV0ZwYkM1amIyMGlMQ0pwWVhRaU9qRTNNalF6TVRjeU56VXNJbVY0Y0NJNk1UY3lORE15TURnM05YMC44aDRuSzF5MUd4YzhnMlJSazJxQmZ3NHk2bzJHR3V0LWRlZUZId3VUVXVFSkI2aTQyN25jaTJNa0FkYVpaMHd3RE84Ui1BZ0NnT2tWZnNQTUhKWVJwQQ%3D%3**](http://localhost:8081/dashboard?Authorization=ZXlKaGJHY2lPaUpJVXpVeE1pSjkuZXlKemRXSWlPaUp5YjJocGRFQm5iV0ZwYkM1amIyMGlMQ0pwWVhRaU9qRTNNalF6TVRjeU56VXNJbVY0Y0NJNk1UY3lORE15TURnM05YMC44aDRuSzF5MUd4YzhnMlJSazJxQmZ3NHk2bzJHR3V0LWRlZUZId3VUVXVFSkI2aTQyN25jaTJNa0FkYVpaMHd3RE84Ui1BZ0NnT2tWZnNQTUhKWVJwQQ%3D%253)**D**

* **explanation application.properties file:**

1. **spring.application.name=RBASpringSecurityProject**
   * **Purpose: Sets the name of the Spring Boot application. This name can be used in logs, metrics, and monitoring tools to identify the application.**
2. **spring.datasource.url=jdbc:mysql://localhost:3306/management\_system**
   * **Purpose: Specifies the JDBC URL for connecting to the MySQL database. It includes the database host (localhost), port (3306), and the database name (management\_system).**
3. **spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver**
   * **Purpose: Defines the JDBC driver class name for MySQL. This driver is used to connect to the MySQL database.**
4. **spring.datasource.username=root**
   * **Purpose: Provides the username for accessing the MySQL database. In this case, the username is root.**
5. **spring.datasource.password=root**
   * **Purpose: Provides the password for accessing the MySQL database. In this case, the password is root.**
6. **spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect**
   * **Purpose: Configures the JPA (Java Persistence API) to use MySQL 8 dialect. This ensures Hibernate uses the appropriate SQL syntax for MySQL 8.**
7. **spring.jpa.hibernate.ddl-auto=update**
   * **Purpose: Configures Hibernate to automatically update the database schema to match the entity models. The update setting means Hibernate will apply changes to the database schema but won't drop existing tables.**
8. **spring.jpa.show-sql=true**
   * **Purpose: Enables SQL logging, so SQL statements executed by Hibernate will be printed to the console. This is useful for debugging and monitoring SQL queries.**
9. **spring.mvc.view.prefix=/WEB-INF/views/**
   * **Purpose: Specifies the prefix path for view resolution. This tells Spring MVC where to find JSP files. All view names will be prefixed with this path.**
10. **spring.mvc.view.suffix=.jsp**
    * **Purpose: Specifies the suffix for view resolution. This tells Spring MVC to look for files with the .jsp extension as view templates.**
11. **logging.level.org.springframework.security=DEBUG**
    * **Purpose: Sets the logging level for Spring Security to DEBUG. This will output detailed logs related to Spring Security, which is helpful for troubleshooting security-related issues.**
12. **spring.main.allow-circular-references=true**
    * **Purpose: Allows circular references in Spring Beans. This can be useful in some cases where beans depend on each other in a circular manner. By default, Spring Boot does not allow circular dependencies to avoid potential issues.**