Case Study: User Registration and Login Process

Our company aims to develop a web application where users can register and log in. This application will require users to register and then activate their accounts. Additionally, users should be able to update their profile information and deactivate their accounts. Here are the details of our requirements:

User Registration

• Users can create new accounts by filling out a registration form.

• Users must provide information such as username, email, and password.

• After the registration process is completed, a confirmation email should be sent to the user.

• In the confirmation email, the user must click on an activation link.

• Clicking the link should activate the user's account and make it active.

• Inactive users should be in a deactivated status.

Login Process

• Registered users can log in using their username and password.

• Upon successful login, the user should be provided with an access token.

• The access token should be used to authenticate the user in other API requests.

• The access token should be securely stored and checked with each request.

Profile Management

• Users should be able to update their account information (username, email, password, etc.).

• Users should be able to update their passwords.

• Users should be able to deactivate their accounts.

• Deactivated accounts should transition to a deactivated status.

System Architecture

• Our system should utilize a microservices architecture.

• Operations should follow the Command Query Responsibility Segregation (CQRS) pattern.

• Our application should be developed using the Test-Driven Development (TDD) approach.

• All actions should be recorded using an event sourcing architecture.

• We should employ the ELK (Elasticsearch, Logstash, Kibana) stack to collect, analyze, and visualize application logs. Setting up the ELK stack aims to facilitate faster issue detection and improvements by logging and analyzing our application.

• Our application will interact with external services. It should handle error scenarios and be prepared for service interruptions. The project needs to support various error handling strategies such as error management, fault tolerance, and rollback policies.

• To support an event-driven architecture in our application, we should use a service bus like Masstransit. Using a service bus, we can establish communication between appropriate events and commands for user registration, login, profile updates, and other operations.

Mock Mail Provider

For the purpose of this case study, we will mock the functionality of the mail provider. Since the focus of the case study is on the user registration and login process, simulating the actual email sending process may not be necessary. Instead, we can implement a mock mail provider that mimics the behavior of a real mail provider.

The mock mail provider can have the following features:

1. Registration Confirmation: When a user registers, instead of sending an actual email, the mock mail provider can generate a confirmation link that would normally be included in the confirmation email. This link can be logged or displayed in the console for testing purposes.
2. Account Activation: The mock mail provider can simulate the account activation process by providing a mechanism to validate the activation link. It can verify that the link is valid and mark the user account as active.
3. Logging: The mock mail provider can log the actions it performs, such as generating confirmation links, validating activation links, and marking accounts as active.