Complex Systems and DevOps: Deliverable 2

Date: November 25, 2024

Course Details

Course Name: Complex Systems and DevOps **Course Code:** 62582 **Semester:** Fall 2024

Team Members

Name	Student Number
Christoffer Fink	*s205449*
Kasper Falch Skov	*s205429*
Johan Søgaard Jørgensen	*s224324*
Henrik Lynggaard Skindhøj	*s205464*
Kevin Wang Højgaard	*s195166*
Sebastian Halfdan Lauridsen	*s215769*

Link to GitHub Project

Table of Contents

4. Backend Development
    - Quarkus Framework Implementation
    - REST API Design with Siren Hypermedia
    - Database Integration
    - Business Logic Implementation
    - Security Implementation
        - JWT Authentication
        - Backend Security Measures
    - Testing Strategy
        - JUnit Implementation
        - REST-assured Testing
    - OpenAPI Documentation
5. Frontend Development
    - React Application Structure
    - TypeScript Integration
    - Vite Build Tool Implementation
    - State Management
    - Component Architecture
    - Security Features
        - Token Security Implementation
    - Package Management
6. DevOps Implementation
    - Version Control Practices
        - Git Workflow
        - GitHub Integration
    - Continuous Integration/Continuous Deployment
        - GitHub Actions Configuration
        - Build Server Setup
        - Testing Pipeline
    - Containerization
        - Docker Implementation
        - Container Registry
    - Cloud Deployment
        - Google Cloud Setup
        - Netlify Frontend Deployment
    - Monitoring and Maintenance
7. Conclusion
    - Project Outcomes
    - Future Improvements
    - Lessons Learned

+– csdg8 +– auth | +– AuthController.java | +– AuthResource.java | +– AuthService.java | +– dto | | +– CreateTokenRequest.java | | +– CreateTokenResponse.java | | +– GetAuthResponse.java | | +– RefreshAccessTokenRequest.java | | +– RefreshAccessTokenResponse.java | +– TokenService.java +– user | +– dto | | +– CreateUserRequest.java | | +– GetCollectionUser-

Response.java | | +– GetUserResponse.java | +– UserController.java | +– User.java | +– UserResource.java | +– UserService.java | … “‘

Snippet of the backend file structure showing its organization by the domains “auth” and “user”.

To further simplify new features or changes our general class hierarchy is as follows

1. Resource
2. Controller
3. Service
4. Business Logic

The resource classes only have direct interaction with a single controller in the same domain or subdomain. A controller only has direct interaction with resources and service classes. The service classes may interact with multiple controllers and services and finally the business logic, in our case the game logic, may only interact with other business logic classes or preferrably a single service class.

Another feature of this model is our separation of DTOs from implementation classes. None of the API responses return a class that is directly used in a service or in business logic. Instead, they are mapped to a DTO to both shield the actual business logic from the web application framework and vice versa but also to limit accidentally exposing sensitive information, like passwords for returned user objects.

A further improvement to this structure would be, in the same vein as our DTOs, shielding the service and business logic from our database. This would allow a, not-so, “hot-swap” of frameworks or database fully isolating the API from the business logic and service, and the database.

Security Implementation *quarkus security with rolesallowed, 401 vs 403.*

JWT Authentication *jwt tokens, upn, claims etc. shortlived/longlived tokens what and why etc.*

Backend Security Measures *probably covered above*

Testing Strategy

JUnit Implementation

REST-assured Testing

OpenAPI Documentation