

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	<i>s205449</i>
Kasper Falch Skov	<i>s205429</i>
Johan Søgaard Jørgensen	<i>s224324</i>
Henrik Lynggaard Skindhøj	<i>s205464</i>
Kevin Wang Højgaard	<i>s195166</i>
Sebastian Halfdan Lauridsen	<i>s215769</i>

Link to GitHub Project

Table of Contents

1. Introduction
 - Project Scope and Objectives
 - Problem Statement and Solution Overview
 - Methodology
- I. Analysis
 2. Domain Analysis
 - User Stories and Requirements
 - System Architecture Overview
 - Technical Stack Selection
 - Security Requirements
 3. Technical Foundation
 - Framework Selection Rationale
 - Development Environment Setup
 - Project Structure
- II. Implementation and DevOps Practices

- 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

1. Introduction

Project Scope and Objectives

Problem Statement and Solution Overview

Methodology

I. Analysis

2. Domain Analysis

User Stories and Requirements

System Architecture Overview

Technical Stack Selection

Security Requirements

3. Technical Foundation

Framework Selection Rationale

Development Environment Setup

Project Structure

II. Implementation and DevOps Practices

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 The project features a React application for the frontend using Typescript. This is designed to provide an engaging and interactive experience for the users. It has a landing page with multiple sections, and navigation to each implemented game. Furthermore it is developed in a modular way, ensuring that scaling the project up and down can be done easily.

Key sections and features The landing page is the first page of the application and where the user will start when they enter. It is built up by using several components that form the landing page when put together. The header is at the top of the landing page, and serves as a navigation bar. It has the logo of the application and then links to different sections of the application: Home, About, Features and Contact. Below that is the hero section, where you would typically have a banner, welcoming the user onto the page and inviting them to navigate around the application to explore the different features. Below that is the games section where the different implemented games are listed along with buttons that navigate to said games. Below that there is a “Start” section, which is a component that invites the user to sign up in order to play the games. Lastly a footer is implemented, which is a generic footer containing copyright information and links to resources such as terms of service and privacy policy. The application also has separate game pages, which when navigated to, shows each of the games.

Navigation and Routing

For routing in the application react-router-dom is used to enable navigation between different pages. The routes are defined in the index.tsx file, and currently contains routes for blackjack, coinflip and the landing page. This setup makes it easy if new games are added, because they can just be added to the index file.

TypeScript Integration The project is built with TypeScript, which makes it possible to do static type checking. This is super helpful in order to catch errors during development instead of having to catch them at runtime. TypeScript and the static type checking is best used when defining interfaces and types for components props and states. This limits the data passed between components such that it has to be structured properly for the component to accept it.

Vite Build Tool Implementation for building the application this project uses Vite. This is a popular build tool with some advantages mainly aimed at ease of use for the developers. The advantages include a really fast development

server providing instant updates when changes are made to the code. It supports TypeScript out of the box, which minimized the amount of work developers have to focus on configuring build pipelines.

State Management In this project, Reacts built in state management hooks are used. UseNavigation is used from react-router-dom is used to handle the navigation state and allowing the user to navigate to different pages. Local states are also being used within the components with useState. useState can be used to handle specific UI states, for example when users are interacting with the application, or when dynamic content rendering is used.

Component Architecture

Security Features

Token Security Implementation

Package Management

6. DevOps Implementation

Version Control Practices

Git Workflow Git is an open-source version control system that can be used between developers to help manage and track changes in their repositories. It has become the standard tool that developers use to manage software development projects ranging from small personal projects to huge projects in big organizations. The tool helps track changes to files over time, allows several developers to collaborate on the same project while ensuring that they don't accidentally overwrite each other's code, and helps provide a history of changes which developers can also use to revert to previous iterations of the same file if problems arise. Git is also being used in this project. Currently the project consists of three repositories. One for the frontend, backend and the report. In each of these repositories each group member has their own branch in which they can make changes without interfering with the other group members work. When a group member has then fully developed or written something, it can be merged into the dev branch.

GitHub Integration Github is an extension to the git version control system, that also serves as a hosting platform for git repositories. It comes with a lot of features that can aid the collaboration between developers to a great extent. For this project, a github organization has been made. That way the group can have their three repositories in a shared space. To each repository the group also has a work board, where tasks can be created such that each group member can see what needs to be done. This helps create an overview of the project

status. When group members have developed or written something, they can create a pull request, “asking” for their branch to be merged into a different branch. It is essentially an invitation for the rest of the group to review what has been committed to the repository before merging it into the development branch. This is a way for developers to collaborate and catch errors before they reach a shared branch. Pull requests can also be used to setup continuous integration and continuous deployment.

Continuous Integration/Continuous Deployment

GitHub Actions Configuration

Build Server Setup

Testing Pipeline For the backend of the project, we’re using a GitHub Actions workflow as a testing pipeline. The workflow is triggered on push events as well as pull request events into the dev and main branch. The **build** part of the workflow has steps included to set up the JDK, generating fake JWT keys and building the project using Maven. Following that there is a **test** job which depends on the build job. This means that the test job will not run if the build job fails. If the build job has succeeded, It will download the build artifacts, set the JWT key environment variables, and run the tests for the backend using Maven. This pipeline runs, as said previously, every time a pull request is made, making it super useful, as it prevents any pull requests that do not either build or pass all the tests from being merged into the dev or main branch.

Containerization

Docker Implementation

Container Registry

Cloud Deployment

Google Cloud Setup

Netlify Frontend Deployment

Monitoring and Maintenance

7. Conclusion

Project Outcomes

Future Improvements

Lessons Learned