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

---

1. Introduction

1.1 Project Scope and Objectives The goal of this project is to design and implement a full-stack gambling platform, with games such as roulette, and coin flip. The project includes the development of both a backend, a frontend, and the use of DevOps practices to improve the development and deployment processes.

The project will provide an secure and intuitive user experience. Software using modern frameworks and tools will be implemented. The project will be implemented with an emphasis on using the practices of DevOps to enable continous integration and deployment.

1.2 Problem Statement and Solution Overview

Problem Statement: In this project several challenges had to be overcome, these include:

- **Scalability:** Making it possible for several different users to play simultaneously, while maintaining uninterrupted gameplay.
- **Security:** Preventing exploitation of tokens, to prevent potential threats.
- **Maintainability:** Creating a system that can be updated and improvied without any significant overhead.
- **Deployment:** Developing a deployment process with frequent updates without downtime

Solution Overview: The project addresses these challenges the following ways:

- A **React frontend** using **Vite** and **TypeScript** to create a user-friendly interface.
- A **Quarkus-based backend** written in Java to implement core functionalities such as game logic, authentication, and data management. **PostgreSQL** is used as the database for data storage.
- **DevOps practices**:
  - **Containerization** with Docker to provide environments that are consistent across development, testing, and production.
  - **Version control and automation** using GitHub and GitHub Actions for continuous integration and deployment.
- An emphasis on **secure authentication and authorization**, by using **JWT tokens** to manage user sessions effectively.

These approaches both addresses the technical challenges, and ensures that the platform is secure, and scalable.

1.3 Methodology The project follows the Agile methodology of iterative development in a manner that allows for easier addaptation to changes in require-

ments and the users needs. The process initiates with comprehensive requirements gathering and analysis, where user stories and use cases are written in detail to ensure alignment with end-user expectations. This approach promotes clarity of purpose for the project and lays a solid foundation for later phases in development.

The development phase involves the creation of both the frontend and backend, each designed with scalability, and maintainability in mind. The structure of the frontend is component-based, built with React and TypeScript, promoting reusability. This makes it easier for developers to update and extend the user interface every time something a new feature is implemented. The backend uses a layered architecture based on Quarkus, which isolates concerns into three layers for business logic, API endpoints, and data management. This makes the code cleaner and easier to scale, making it easier to integrate further features.

DevOps practices are integrated into the project to improve the development and deployment processes. Continuous integration and deployment pipelines will be implemented with the help of GitHub Actions where code builds, testing, and deployments are automated. It minimizes manual labour, reduces the possibilities of human errors, and speeds up the release cycle. Docker is used for containerization, which maintains the consistency of the environments throughout the development, testing, and production stages, enhancing reliability and simplifying the deployment process.

The deployment strategy adopted here is cloud-based, with the backend deployed to Google Cloud, and the frontend hosted on Netlify.

By integrating these methodologies, the project ensures a smooth development process, that's responsive to change.

I. Analysis

2. Domain Analysis

User Stories and Requirements

Use Cases & User Stories

The project has not fundamentally changed since the previous delivery, meaning there haven't been any changes to our requirements, use cases or user stories. We used our user stories (US) to derive the corresponding use cases (UC). A use case diagram was created for the first use case, **Game Selection & Play,** to illustrate the system interactions.

Main – UC 1.1. Diagram

```
flowchart TD
    User["fa:fa-user User"] -->|Place Bet| PlaceBet
    User -->|Play Game| PlayGame
```

```
    User -->|Select Game| SelectGame
    User -->|Show Outcome| ShowOutcome

    subgraph GamblingWebsite## Use Cases & Stories
    end
```

Use Cases describe the interactions between a user (or another system) and the system itself to achieve a specific goal. They focus on what the system should do and define the steps involved in achieving the user's objectives, including different success and failure scenarios.

User Stories are short, simple descriptions of features told from the perspective of the end-user. They typically follow the format: "As a *type of user*, I want *goal* so that *benefit*." User stories help capture user needs and provide a foundation for creating detailed use cases.

Our Use Case & User Stories

We used our user stories (US) to derive the corresponding use cases (UC). A use case diagram was created for the first use case, **Game Selection & Play,** to illustrate the system interactions.

Main – UC 1.1. Diagram

```
flowchart TD
    User["fa:fa-user User"] -->|Place Bet| PlaceBet
    User -->|Play Game| PlayGame
    User -->|Select Game| SelectGame
    User -->|Show Outcome| ShowOutcome

    subgraph GamblingWebsite
        direction LR
        PlaceBet(Place Bet)
        PlayGame(Play Game)
        SelectGame(Select Game)
        ShowOutcome(Show Outcome)
    end

    PlaceBet -->|Handle transaction| PaymentProvider["fa:fa-credit-card PaymentProvider"]
    PlayGame -->|Ensure fairness| SystemAdmin["fa:fa-user-shield SystemAdmin"]
```

The diagram shows the main interactions within the gambling website. The **User** selects games, places bets, plays, and views outcomes. The **Payment Provider** handles transactions, while the **System Admin** ensures game fairness and compliance.

US 1.1

As a **Gambler**:

- I want an easy-to-navigate website where I can quickly access and select games, place wagers securely, and enjoy a fair gaming experience.

So that:

- I can participate in gambling activities seamlessly.

The primary use case involves enabling users to play a game and place wagers on it. Please note that elements marked as "Example" may not be included in our version of the program due to scope limitations. Subsequent use cases will include brief summaries, as the main flow has yet to be finalized for the sub use cases. All these use cases (including the main one) are **WIP** and will be subject to change.

UC 1.1: Game Selection & Play

Use Case Section Description **Name:** UC 1.1: Game selection & play

Scope:

- Selection of available games
- User interaction with game features
- Placing wagers
- Ensuring a fair and compliant gaming experience
- Secure transactions

Primary actor: User (gambler)

Level: cloud / level 0

Stakeholders & Interest

- **User:** Wants an easy-to-navigate webpage, easy access to games, and the ability to place wagers.
- **System Admin:** Needs to ensure that the games function smoothly, comply with legal regulations, and maintain fairness in wagers.
- **Payment Provider:** Interested in secure & fast transactions.

Preconditions:

- User must have an active account and be signed in.
- User must have sufficient funds in their account to play (free play if applicable).
- "The platform must be legal".

Postconditions:

- If the user wins, their account balance is updated with the winnings.
- If the user loses, the wager amount is deducted from their account balance.

- Game data, including the outcome and amount wagered, is stored in the system for auditing and regulatory purposes.
- The user has the option to leave feedback on the game or report any issues.

Main Success Scenario

1. **Game Lobby:**
   - User navigates to the Landing Page (home page for games).
   - The website displays a list of available games (e.g., slots, poker, blackjack, roulette) with categories like Top Games, New Releases, Jackpot Games, etc.
   - User sees options for sorting and filtering games by type, popularity, jackpot size, etc.
2. **Game Selection:**
   - User clicks on a game thumbnail to view detailed information about the game.
   - Information includes game rules, minimum/maximum bets, potential winnings, RTP (Return to Player) percentage, and a "Play Now" button.
3. **Game Loading:**
   - The system loads the selected game in the browser, initializing game assets and connecting to the game server if needed.
   - A loading screen shows the game logo or promotional visuals while waiting.
4. **Bet Placement:**
   - The user is presented with betting options (e.g., stake size for a slot machine, bet type for blackjack).
   - The user chooses the amount they want to wager and confirms the bet.
   - If applicable, the system checks the user's available balance to ensure they have enough funds.
5. **Game Play:**
   - The game begins. For example:
     - **Slots:** User clicks "Spin" and watches the reels turn.
     - **Blackjack:** User receives virtual cards and makes decisions (hit, stand, etc.).
     - **Roulette:** User places bets on numbers or colors and watches the wheel spin.
   - The game result is calculated based on chance (RNG for slots, cards drawn, etc.).
   - If the user wins, the system calculates the winnings based on the game's rules and updates the user's balance.
6. **Game Outcome:**
   - The game outcome (win/loss) is displayed on the screen.
   - If the user wins, they are shown a breakdown of the win (e.g., wager amount, multiplier, and total win).

- If the user loses, they are notified of the loss and given the option to play again or exit.

7. **Game Exit:**
   - The user can choose to continue playing, select a new game, or exit to the main Game Lobby.
   - If the user exits, the system saves their current session (if applicable) for future retrieval.

Extensions

1. **Insufficient Funds:**
   - If the user doesn't have enough funds to place a bet, they are notified and redirected to the deposit page.
   - They are offered an option to play in free/demo mode if available.
2. **Game Connection Loss:**
   - If the user loses connection during gameplay, the system saves the game state and restores it when they reconnect.
   - If the game outcome is already decided (e.g., a slot spin completes server-side), the user will see the result upon reconnecting.
3. **Bonus Play:**
   - If the user has an active bonus (e.g., free spins or match bonus), they are notified of the bonus during game selection.
   - The system tracks bonus progress and winnings separately from the user's main balance.

Frequency of Occurrence: Every time the user wants to play a game.

US 1.2

As a gambler:

- I want my winnings to be credited to my account and losses debited immediately after each game so I can track my balance.

UC 1.2: Game Outcome Processing

- Winnings are credited to the user's account, and losses are debited.
- Storing game data for auditing, fairness checks, and regulatory compliance.

Standalone Use Cases

Use cases without a corresponding user story.

UC 1.3: Betting Mechanics

- Allowing users to place bets, select stakes, and confirm wagers.
- Handling different game types (e.g., slots, poker, blackjack, roulette).

UC 1.4: Backend Game Hosting

- Hosting and managing games to ensure real-time gameplay.
- Dynamically scaling resources to handle user traffic.
- Processing game outcomes securely.
- Storing game data for compliance and auditing.
- Ensuring high availability, security, and fairness across all hosted games.

System Architecture Overview A lot of thought has gone into designing the system architecture for the program. The architecture has been designed to ensure scalability, reliablity and to rely on DevOps principles. Out approach enables efficient development, maintenance, and provides flexibility for future updates. A detailed breakdown of the different architectural layers is provided below:

Architectural Layers:

1. **Presentation Layer (Frontend):**

The frontend consistes of the user interface part of the platform, with a focus on giving the user an intuitive experience. The tools that have been used to achieve this include:

- The frontend has been developed with **React**, a modern JavaScript library used to build dynamic and interactive web interface.
- The integration of **TypeScript** provides the program with type safety, reducing development errors and enhancing maintainability.
- Navigation is handled with **React Router**, enabling a seamless transition between pages.
- For the Styling of the program **Tailwind CSS** is used. Tailwind is a utility-first CSS framework that ensures responsive and consistent design across devices.

2. **Business Logic Layer (Backend):**

The backend is built using **Quarkus**, a cloud-native framework optimized for building lightweight and high-performing Java applications. It The most important business logic, including:

- Secure authentication using **JWT** (JSON Web Tokens) for access and refresh tokens.
- Modular services handling game logic, user balance management, and transaction workflows.
- APIs for frontend interaction, designed with REST principles and using Hypermedia formats like **Siren** for enhanced discoverability.

3. **Data Access Layer (Database):**

Data management is another important part of the program, critical for the platform's integrity, handled by:

- **PostgreSQL**, a powerful relational database for transactional data such as user balances, game outcomes, and transaction records.
- Integration with **Hibernate**, simplifying database operations and maintaining a clean abstraction between domain models and database schemas.
- Future enhancements include the implementation of repository patterns for improved data handling and scalability.

Technical Stack Selection The technical stack for the project was chosen from the previous experiences of the group members, but also to meet the requirements of a modern, scalable web application. Each component was picked to fit the experience of the group, the performance, reliablity and,alignment with project goals.

Frontend:

- **React:** A library used for building interactive and reuseable UI componenets. React supports dynamic updates without requiring a page reload, which makes for a better user experience.
- **TypeScript:** Provides static type checking, making the codebase more robust and reducing runtime errors.
- **Vite:** A build tool that offers fast build times and hot-reloading, making the work of the developer easier.
- **Tailwind CSS:** Simplifies responsive design implementation and ensures a consistent UI experience.

Backend:

- **Quarkus:** A Java framework tailored for microservices and cloud-native applications. Its fast startup time and low memory footprint make it ideal for the project.
- **PostgreSQL:** A reliable and scalable database for handling structured data, ensuring high availability and performance.
- **Hibernate:** An ORM tool that abstracts database interactions, reducing boilerplate code and enhancing maintainability.

DevOps and Deployment:

- **Docker:** Provides the project with consistent development and production environments through containerization.
- **GitHub Actions:** Used for automating the CI/CD pipeline, ensuring code quality and fast deployment.
- **Google Cloud & Netlify:** Supports scalable hosting for the backend and seamless deployment of the frontend.

Justification: This stack has been designed to best fit the project's objectives, balancing modern features with stability. The stack supports easy development,

ensures system reliability, and provides room for future enhancements, such as the integration of additional games.

Security Requirements In this project, **JWT (JSON Web Tokens)** are used to restrict the amount of time a user can spent before they have to loggin again, this is done to manage user authentication and maintain session control. This approach is essential to ensure security on the site.

Use of JWT Tokens

- **Session Duration:** Each JWT has an expiration time, which decides when a user has to log in again.
- **Stateless Authentication:** This way the backend doesn't have to store any information, since the JWT contains all the necessary information to authenticate a user.

Advantages of Using JWT Tokens

1. **Efficiency:**
   - JWT tokens are stateless, this means that the server doesn't need to maintain session data. This reduces the servers load.
   - The tokens carries all the required claims (e.g., username, user ID) within the token.
2. **Security:**
   - The use of expiration times limits the duration of exposure if a token is compromised.
   - Tokens are signed using algorithms like **HMAC SHA-256**, making it impossible to tamper with the token, unless the signing key is exposed.
3. **Flexibility:**
   - JWTs can be used across several different services and platforms, enabling easy integration in distributed systems.
   - Refresh tokens can be implemented to allow users to obtain new access tokens without the need to log in again.

Disadvantages of Using JWT Tokens

1. **Token Revocation Complexity:**
   - Invalidating a token before it expires (in case of user logout), can be challenging, since the server does not store session data.
   - Implementation of token blacklists can add additional complexity and reduce the "stateless" advantage.
2. **Token Size:**
   - JWT tokens can be large due to their payload, which may include claims and metadata. This can slightly increase bandwidth usage.
3. **Security Risks:**

- Attacks can forge valid tokens if the signing key is stolen.
- Storing tokens on the client side can expose them to cross-site scripting (XSS) attacks

JWT tokens provide and efficient approach to session management, but must be implemented carefully to avoid the associated risks. ### 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