**Project Plan**

**TICKETWAVE**

|  |
| --- |
| **Date : 9th September 2024** |
| **Version : 1.2** |
| **State : Draft** |
| **Author : Anh Huynh** |

#### Version History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 0.0 | 9/9/2024 | Anh Huynh | Making project plan | Draft |
| 1.0 | 18/9/2024 | Anh Huynh | Making some adjustments based on the teacher’s feedback | On-going |
| 1.1 | 23/9/2024 | Anh Huynh | Adjust activities and time phases based on URS changes | On-going |
| 1.2 | 11/10/2024 | Anh Huynh | Number the WBS products | On-going |

Contents

[**1. Project assignment 4**](#_Toc42673512)

[1.1 Context 4](#_Toc42673513)

[1.2 Goal of the project 4](#_Toc42673514)

[1.3 Scope and preconditions 4](#_Toc42673515)

[1.4 Strategy 5](#_Toc42673516)

[1.5 End products 6](#_Toc42673517)

[**2. Project organisation 8**](#_Toc42673519)

[2.1 Stakeholders and team members 8](#_Toc42673520)

[2.2 Communication 8](#_Toc42673521)

[**3. Activities and time plan 9**](#_Toc42673525)

[3.1 Phases of the project 9](#_Toc42673526)

[3.2 Time plan and milestones 10](#_Toc42673527)

[**4. Testing strategy and configuration management 1**](#_Toc42673525)**3**

[4.1 Testing strategy 1](#_Toc42673526)3

[4.2 Test environment and required resources 1](#_Toc42673527)4

[4.3 Configuration management 1](#_Toc42673528)4

[**5. Risk 1**](#_Toc42673529)**5**

[5.1 Risk and mitigation](#_Toc42673530) 15

# Project assignment

## Context

This project is an individual assignment as part of a software engineering course, aimed at creating a ticket management system. The platform is designed to support a wide range of events, including concerts, workshops, sports, and theatre performances, and will cater to different user roles such as regular users, sales managers, and administrators. The reason for this project is to address the lack of integrated platforms that cater to different types of events, providing a unified solution that enhances user experience and operational efficiency. The system will be developed using modern technologies: React for the frontend, Java Spring Boot for the backend, and MySQL for the database, aligning with industry standards and course requirements.

## Goal of the project

The primary goal of this project is to develop a ticket management system that simplifies the process of discovering, booking, and managing event tickets for both users and organisers. The system will centralise ticket management, allowing users to easily search for events, book tickets, and manage their bookings while providing organizers with tools to effectively promote events and manage sales. Key benefits of the system include improved event visibility, secure and efficient ticket handling, and enhanced user interaction through features such as real-time chat support. Overall, the project aims to create a seamless and accessible platform that improves both user and organizer experiences.

## Scope and preconditions

|  |  |
| --- | --- |
| **Inside scope:** | **Outside scope:** |
| 1. **User-Friendly Event Management Platform:** A responsive platform where users can discover events, book tickets, and manage accounts. | 1. **Payment Processing**: The system will not handle financial transactions or integrate with payment gateways. |
| 1. **Access Control Based on User Roles:** The system will feature role-based access, ensuring regular users, sales managers, and administrators have appropriate permissions for their respective functions. | 1. **Advanced Reporting and Analytics**: Detailed performance metrics, advanced reporting, and data visualizations will not be included. |
| 1. **Event Search and Filters:** A robust search and filtering feature that allows users to find events based on multiple criteria, such as name, location, and event type. |  |
| 1. **Ticket Management**: The function allows users to book tickets and view their purchases. |  |
| 1. **Real-Time Notifications and User Support**: The system will provide notifications related to event updates, bookings, and reminders, along with real-time chat or support capabilities. |  |

**Preconditions:**

* The project must utilize the specified tech stack: React for the frontend, Java Spring Boot for the backend, JPA/Hibernate for ORM, and MySQL for the database. These choices align with the course guidelines and industry standards.
* The project will be developed using Agile methodologies with a series of six sprints, each with specific deliverables and milestones.
* Security measures such as JWT-based role-based authorization must be implemented to protect user data and ensure secure access control.
* The project is expected to adhere to software development best practices, including CI/CD pipelines for continuous integration and testing.

## Strategy

The project will be developed using an Agile approach, specifically the Scrum framework, which allows for iterative and incremental development through a series of sprints. Each sprint will last three weeks and focus on delivering specific features or improvements, allowing for continuous feedback, adaptation, and refinement of the system.

**Justification for Agile Approach:**

* **Flexibility and Adaptability:** Agile allows for changes to be made easily based on feedback or new insights, which is essential for a project of this nature where user experience and system functionality must evolve.
* **Incremental Delivery:** By breaking the project into manageable sprints, each phase can focus on delivering functional components, enabling early testing, feedback collection, and continuous improvement.
* **Enhanced Collaboration and Feedback:** Agile encourages regular communication and collaboration with stakeholders, including feedback from instructors, which ensures that the project aligns with expectations and addresses any issues promptly.
* **Risk Mitigation:** The iterative nature of Agile helps in identifying and addressing potential risks early in the development process, reducing the likelihood of major setbacks later on.

This approach is ideal for developing a dynamic, user-focused application like a ticket management system, as it allows the project to adapt quickly to changing requirements and ensures that the end product is closely aligned with user needs and project goals.

## A diagram with text on it Description automatically generated with medium confidenceEnd products

The Product Breakdown Structure (PBS) outlines the key deliverables of the project, which include both the software application and the necessary documentation to ensure a comprehensive and maintainable system.

**End Product of Project:** A complete ticket management system with supporting documentation and testing.

**A: Key Product - Frontend Application**

**A1: Product – User Interface**

**A1.1: Subproduct – User Dashboard:** Provides users with an overview of their events, tickets, and account settings.

**A1.2: Subproduct – Event Search and Booking Page:** Allows users to search for events by name, location, and other filters and to book tickets.

**A2: Product – Authentication and User Management Interface:** Handles user login, registration, and role-based access control.

**B: Key Product – Backend System**

**B1: Product – Core Service:** Provided services for authentication, ticket management, and event handling.

**B2: Product – Data Management:** Manages data storage and retrieval operations ensuring secure and efficient access.

**C: Key Product – Supporting Documents**

**C1: Product – Design and Architecture Documents:** Detailing the system architecture, data flow, and component interactions to guide the development process and ensure maintainability.

**C2: Product – Test Reports and Documentation:** Includes comprehensive testing results such as unit, integration, and system testing reports to validate the application’s performance and functionality.

**C3: Product – Project Plan Document:** Outlines the project scope, goals, deliverables, and the strategic approach taken during development.

**C4: Product – User Feedback Reports:** Documentation of user feedback collected during the project, along with details of the improvements made in response to this feedback.

**C5: Product – Final Reflection and Evaluation Document:** Summarizes the project outcomes, challenges faced, and lessons learned, providing a critical evaluation of the project’s success and areas for future improvement.

# Project organisation

## Stakeholders and team members

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Abbreviation** | **Role and functions** | **Availability** |
| Raja Gorentla | - | Semester coach | Monday morning and Tuesday afternoon |
| Tim Kurvers | - | Technical teacher | Monday afternoon and Tuesday morning |
| Marcel Boelaars | - | Technical teacher | Wednesday |
| Anh Huynh | - | Developer | Full-time available for the project duration |

## 

## Communication

* **Regular Feedback with Teachers (Feedpulse)**
* **Goal:** To provide ongoing support, address small questions, and offer feedback as needed.
* **Manner:** on-demand communication during class hours or via Microsoft Teams/emails.
* **Frequency:** As needed, ask for feedback anytime during class hours or via Teams/emails when necessary, at least twice per sprint.
* **Attendees:** Developer (student) and teachers.
* **End-of-Sprint Feedback Session**
* **Goal:** To discuss the progress made during the sprint, review completed tasks, and gather feedback on deliverables.
* **Manner:** In-person meeting at school.
* **Timing:** At the end of each sprint.
* **Attendees:** Developer (student) and technical teachers.
* **Project Management Tool (Jira)**
* **Goal:** To track tasks, manage the backlog, and monitor project progress.

# Activities and Time Plan

## Phases of the project

The project will be delivered in three main versions. Each version may span multiple sprints depending on the complexity of tasks and will address specific user stories (URS). "Could have" features will only be implemented if time permits, ensuring priority is given to "must have" and "should have" requirements.

* **Version 1 (Core Functionality - MVP)**:  
  This version focuses on delivering the **must-have** features across Sprints 1, 2, and 3. The core backend and frontend setup, user authentication, ticket management, and event management are covered.

**User Stories in Version 1**:

* US1: Book tickets
* US2: View booked tickets
* US3: Search for events
* US4: Register an account
* US5: Log in
* US11**:** Admin manages user comments
* US12: Admin manages users
* US14**:** Sales manager creates event listings
* US15**:** Sales manager edits event listings
* **Version 2 (Expanded Functionality)**:  
  This version covers **should-have** features, including password resets, admin reporting, and real-time notifications or chat. Sprints 4 and 5 will focus on these features.

**User Stories in Version 2**:

* US6: Reset password
* US7: Real-time notification
* US8: Real-time chat support
* US13: Admin generates reports on ticket sales and user activity
* **Version 3 (Final Product with Full Features)**:  
  This version focuses on **could-have** features such as event recommendations and seat selection. Sprint 6 will complete the system, including these optional features if time permits.

**User Stories in Version 3**:

* US9: Event recommendations
* US10: Seat selection using a floor plan

## Time plan and milestones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **Effort** | **Start date** | **Finish date** | **Milestone** |
| **Sprint 1** | 3 weeks | Sep 2, 2024 | Sep 20, 2024 | **- Backend REST API setup:** 3 working endpoints (GET, POST, DELETE), CI/CD initialized. Layering, dependency inversion, controller, business, and persistence layers.  **- Project Plan Document:** Finalized plan and initial backlog in Jira.  **- Initial Tests:** Unit tests integrated with CI/CD pipeline. |
| **Sprint 2** | 3 weeks | Sep 23, 2024 | Oct 11, 2024 | **- Frontend Setup:** Initial React app setup with one backend service consumption.  **- User Authentication:** Register (US4), Log in (US5), User Account Management (US12).  **- CORS Configuration in Controllers.**  **- Documentation:** Design Document V1 with C4 model (Levels 1-3), SOLID principles. |
| **Sprint 3** | 4 weeks | Oct 14, 2024 | Nov 8, 2024 | **- Complete MVP:** Implement US1, US2 (Ticket Booking and Viewing), Admin Manages Comments (US11). Finalize all Must-Have features for the MVP.  **- Initial Backend to Database Integration:** Connect backend to real MySQL database, replacing mock repositories.  **- Testing:** Unit tests for business layer, mock persistence layer with Mockito. |
| **Version 1 Complete** | V1 delivers **MVP** with must-have features. | | | |
| **Sprint 4** | 3 weeks | Nov 11, 2024 | Nov 29, 2024 | **- Admin Reporting:** Implement US13 (Admin generates reports on ticket sales and user activity).  **- Password Reset:** Implement US6 (Password Reset).  **- SonarQube Integration:** Static code analysis and SonarQube in CI pipeline, with screenshots for deliverable. |
| **Sprint 5** | 3 weeks | Dec 2, 2024 | Dec 20, 2024 | **- Real-time Notifications/Chat:** Implement US7 and US8.  **- CI Setup Documentation:** Include a diagram of the CI pipeline setup (GitLab, SonarQube, etc.). |
| **Version 2 Complete** | V2 adds **Should-Have** features like notifications, chat, and reporting. | | | |
| **Sprint 6** | 4 weeks | Dec 20, 2024 | Jan 17, 2025 | **- Event Recommendations and Seat Selection:** Implement US9 and US10 if time permits.  **- Security Report:** Evaluate how the application deals with OWASP top 10 security risks.  **- Final Testing:** Full integration and end-to-end tests, including performance testing. |
| **Version 3 Complete** | V3 delivers the final product with **Could-Have** optional features (if time permits). | | | |

# Testing strategy and configuration management

## Testing strategy

The testing strategy will cover different levels to ensure the system works as expected. The main testing types include Unit Testing, Integration Testing, End-to-End (E2E) Testing, and Acceptance Testing.

1. **Unit Testing:**
   * **Purpose:** Test individual pieces of code like functions and methods.
   * **Tools:** JUnit for backend (Spring Boot) and Jest for frontend (React).
   * **Scope:** Focus on small, specific parts of the application.
   * **Automation:** Fully automated and runs automatically during development.
   * **Goals:** Aim for 70-80% coverage on key components.
   * **Why:** Quickly identify and fix errors in specific code parts.
2. **Integration Testing:**
   * **Purpose:** Test how different parts of the application work together.
   * **Tools:** Spring Boot Test, Postman, and Mockito for mocking dependencies.
   * **Scope:** Covers things like database interactions and API calls.
   * **Automation:** Mainly automated; complex cases may need some manual checks.
   * **Goals:** Ensure that components communicate properly.
   * **Why:** Catch issues that occur when different parts of the system interact.
3. **End-to-End (E2E) Testing:**
   * **Purpose:** Test the entire application flow, simulating real user actions.
   * **Tools:** Cypress or Selenium for automated browser testing.
   * **Scope:** Includes complete user paths like logging in, searching events, and booking tickets.
   * **Automation:** Mostly automated for consistency.
   * **Goals:** Confirm the application behaves correctly from start to finish.
   * **Why:** Validate the overall user experience and ensure the full system works together.
4. **Acceptance Testing:**
   * **Purpose:** Verify that the system meets the requirements defined by stakeholders.
   * **Tools:** Manual testing and feedback sessions with stakeholders.
   * **Scope:** Focus on key features that must work correctly.
   * **Automation:** Mostly manual, based on acceptance criteria.
   * **Goals:** Ensure the system is ready for use.
   * **Why:** Final check to make sure the system meets user needs.
5. **Static Code Analysis and Quality Checks:**
   * **Purpose:** Assess code quality and detect issues automatically.
   * **Tools:** SonarQube.
   * **Automation:** Runs with each code commit.
   * **Why:** Maintain consistent code quality and identify potential problems.

## Test environment and required resources

The project will use a structured setup to support testing and deployment:

1. **Development Environment:**
   * **Purpose:** Local setup for coding and initial testing.
   * **Resources:** IDEs (IntelliJ, VS Code), Gradle, and Docker.
2. **Testing Environment:**
   * **Purpose:** Dedicated environment for automated tests.
   * **Resources:** Cloud-based virtual machines or containers.
3. **Acceptance Environment:**
   * **Purpose:** Staging environment for final checks before going live.
   * **Resources:** Cloud infrastructure similar to the production setup.
4. **Production Environment:**
   * **Purpose:** The live environment is accessible to users.
   * **Resources:** Secure, monitored cloud servers.

## Configuration management

1. **Version Control System:**
   * **Tool:** Git on GitHub or GitLab.
   * **Branching Strategy:** GitFlow with main, development, and feature branches.
2. **Promotion and Release Strategy:**
   * **Process:** Code progresses from feature branches to development and then to main for release.
   * **Releases:** Tagged for easy tracking.
3. **Change Management:**
   * **Tool:** Jira for tracking tasks and issues.
   * **Process:** Changes are reviewed, tested, and merged following a structured workflow.

# Risk

## Risk and mitigation

|  |  |  |
| --- | --- | --- |
| **Risk** | **Prevention activities** | **Mitigation activities** |
| 1.Technical Challenges or Bugs | - Regularly integrate code and run automated tests to catch issues early. | - Use debugging tools and consult teachers for guidance.  - Allocate extra time in sprints for bug fixes. |
| Scope Creep | - Define the project scope clearly at the start.  - Regularly review the backlog to keep focus on what’s important. | - Defer less critical features to future sprints if needed.  - Re-prioritize features if new requests come up, focusing on essential tasks. |
| Lack of Feedback Availability | - Schedule feedback sessions ahead of time and communicate frequently.  - Be proactive in seeking feedback during class hours or via other available channels. | - Use Teams or email for feedback if in-person is not possible. |
| Missed Deadlines | - Set realistic goals for each sprint and monitor progress regularly.  - Break tasks into smaller, manageable pieces to prevent bottlenecks. | - Re-prioritize tasks to focus on the most critical features.  - Adjust deadlines slightly to accommodate delays where possible. |
| Performance Problems | - Optimize code during development and conduct basic performance tests. | - Simplify or optimize features if performance issues arise. |

* **Risk Matrix**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Risk | |  | | --- | | **Likelihood (1-5)** |  |  | | --- | |  | | |  | | --- | | **Impact (1-5)** | | Risk Priority Number (RPN) |
| 1.Technical Challenges or Bugs | 3 | 4 | 12 |
| Scope Creep | 4 | 3 | 12 |
| Lack of Feedback Availability | 2 | 3 | 6 |
| Missed Deadlines | 3 | 5 | 15 |
| Performance Problems | 2 | 3 | 6 |