Individual Project
Project Plan Document

Date: 19.03.2024

Version: Version 1.3

Version History

Version	Date	Author	Changes	State
1.0	23.02.2024	Claudiu Badea	Initial plan with objectives and risk assessment.	Complete
1.1	26.02.2024	Claudiu Badea	Updates on testing strategies	Complete
1.2	05.03.2024	Claudiu Badea	Updates on configuration management	Complete
1.3	19.03.2024	Claudiu Badea	Added roles	Complete

Contents

1 Project Definition

1.1 Project Background

QWEST is a web application designed to simplify travel planning. It combines a React frontend and Java Spring Boot backend to offer users customized travel itineraries. The platform uses user preferences and real-time travel data to create personalized travel plans, enhancing the planning experience with interactive features.

1.2 Project Goal

The primary objective of the QWEST is to simplify and personalize the travel planning process. By offering an adaptive and user-friendly platform, it aims to cater to the unique needs and preferences of travellers worldwide, making travel planning an enjoyable and hassle-free experience.

1.3 Constraints

Development will use React for a responsive interface and Java Spring Boot for backend services. MySQL and JPA/Hibernate will manage data, with JWT for secure access control.

1.4 Current Situation

Travelers currently face the challenge of using multiple platforms to plan trips. QWEST aims to unify these aspects into a single, streamlined service, enhancing the user experience by offering a platform that adapts to their personal travel preferences and simplifies the entire planning process.

2 Risk Assessment

Risk Category	Risk Description	Mitigation Strategy	Probability	Impact	Priority
Technical	Misinterpretation of project requirements	Regular communication with advisors and iterative feedback cycles	Unlikely	Harmful	High
Technical	Integration challenges with travel APIs	Early testing with APIs and contingency planning for alternative data sources	Likely	Moderate	High
Technical	Database issues (e.g., corruption, loss)	Implement regular backups and database redundancy plans	Rare	Critical	High
Technical	Inadequate system performance	Conduct performance testing and optimization regularly	Likely	Moderate	Medium
Managerial	Misalignment with user expectations	User testing and feedback loops in early stages	Likely	Harmful	High
Managerial	Poor time management	Adopt Agile methodology with defined sprints and milestones	Likely	Harmful	High
Managerial	Scope creep impacting timelines	Enforce a strict change control process with stakeholder agreement	Likely	Moderate	Medium
Managerial	Resource allocation challenges	Detailed project planning and resource management strategies	Unlikely	Moderate	Medium
External	Changes in travel regulations affecting features	Stay updated with travel regulations and adapt the platform accordingly	Rare	Moderate	Low
External	Third-party API limitations or failures	Establish agreements with API providers and have backup solutions	Likely	Moderate	Medium
External	Cybersecurity threats to user data	Implement state-of-the-art security measures and conduct regular security audits	Likely	Critical	High

3 Deliverables

3.1 Functionality

1. Website

- **Home Page**: Interactive interface for inputting travel preferences, dynamic itinerary suggestions.
- **Itinerary Customization**: User interface for modifying and booking itineraries.
- **User Profile**: Registration, login, profile management, and historical itineraries.
- **Feedback System**: Interface for rating, reviewing, and sharing itineraries.
- **Search and Filter**: Advanced filtering options for itineraries based on destinations, budget, etc.
- **Privacy Policy & Terms of Use**: Legal pages detailing user rights and obligations.
- **Support and FAQs**: Help sections for user guidance.
- **404 and Error Handling**: Custom error pages for better user experience.

2. Admin Dashboard

- **Login System**: Secure access for admin roles.
- **Content Management**: CRUD operations for itineraries, destinations, and user feedback.
- **User Management**: Admin functionalities to manage traveler and optional content creator profiles.
- **Analytics Dashboard**: Insights on user behavior, popular itineraries, and feedback.
- **Notification Management**: System for creating and managing alerts and advisories for travelers.
- **Settings**: Configuration settings for application parameters, API integrations, etc.

3.2 Non-Deliverables

- **Real-time Multi-user Itinerary Planning**: Collaborative planning features will not be included in this phase.
- **Mobile Application**: A dedicated mobile app is not planned for this project phase.

3.3 Documentation

- **Comprehensive Project Documentation**: This will encompass the project plan, technical specifications, and user documentation, elaborating on QWEST's architecture, design choices, and features. It aims to provide a thorough understanding of the project's framework and functionalities.
- **Test Reports and Quality Assurance Documentation**: Detailed records of testing strategies, methodologies employed, test outcomes, and quality assurance actions undertaken during the development cycle. This documentation is critical for ensuring the platform's reliability and optimal performance.

3.3 Roles

1. **Traveler (Regular User):**
   - Registration and profile management
   - Customize itineraries and book services directly, with real-time pricing and availability.
   - Rate and review itineraries and bookings, contributing to a community-driven recommendation system.
2. **Travel Consultant (Admin):**
   - Manage user-generated content for quality and appropriateness.
   - Manages notifications and partnerships with service providers.
3. **Content Creator (Contributor) - Optional:**
   - Submits travel guides, tips, and itineraries for platform publication, enriching the content and user experience.

4 Phasing

QWEST will implement an Agile Scrum methodology with a six-sprint structure. This approach facilitates iterative development, continuous feedback integration, and promotes adaptability to ensure optimal project outcomes.

Sprint 1: Project Setup and Initial Backend (**Weeks 1-3**)

- **Week 1**: Research technologies, finalize the project plan, and begin backend setup with Spring Boot; focus on user authentication using JWT.
- **Week 2-3**: Continue backend setup, establish basic REST endpoints, and start database design discussions. Begin drafting documentation including URS.

Sprint 2: Frontend Development and User Flow (**Weeks 4-6**)

- **Week 4**: Set up the React development environment and start working on the homepage and user registration/login pages.
- **Week 5**: Develop the itinerary questionnaire interface and profile management functionalities.
- **Week 6**: Review and iterate based on initial feedback. Finalize the static pages and ensure responsive design principles are applied.

Sprint 3: Database Design and Data Management (**Weeks 7-9**)

- **Week 7**: Design the MySQL database schema. Integrate JPA/Hibernate for ORM. Start implementing the logic for dynamic itinerary creation.
- **Week 8-9**: Develop database connectivity and ensure data flows correctly between the frontend and backend. Begin simple data retrieval operations.

Sprint 4: Advanced Backend Features-Security Measures (Weeks 10-12)

- Week 10: Develop complex itinerary generation logic. Start integrating travel APIs for real-time data.
- Week 11: Implement role-based access control and enhance security measures. Continue API integration and testing.
- Week 12: Focus on performance testing and optimization. Begin preliminary testing of itinerary customization features.

Sprint 5: Integration, Testing, and Feedback (Weeks 13-15)

- Week 13: Integrate frontend and backend components thoroughly. Focus on itinerary customization and booking functionalities.
- Week 14: Conduct user acceptance testing, gather feedback, and iterate on the product.
- Week 15: Enhance the user feedback system and refine the UI/UX based on user testing insights.

Sprint 6: Final Adjustments (Weeks 16-18)

- Week 16: Implement real-time functionalities using Websockets. Prepare for deployment by setting up Docker.
- Week 17: Finalize documentation, including the test plan and project plan revisions. Conduct final testing of all features.
- Week 18: Review, finalize, and submit the project. Ensure all deliverables are prepared and that the application is ready for launch.

For QWEST, the adoption of three-week sprints aims to strike a balance between swift development and sufficient time for in-depth feature creation, testing, and evaluation. This period facilitates ongoing feedback incorporation and cyclic enhancements without sacrificing quality or project breadth.

Sprint operations include:

- Sprint Planning: Initiating each sprint by pinpointing tasks, setting project focus, and establishing sprint objectives.
- Sprint Demo: Concluding each sprint with a demonstration to present completed features to stakeholders, enabling feedback collection and work validation.
- Sprint Review: Following the demo, this meeting allows stakeholders and the team to reflect on sprint accomplishments, examine the progress, and solicit feedback for subsequent sprints.
- Sprint Retrospective: Conducted post-review to assess sprint performance, pinpoint improvement areas, tweak future sprint processes, and discuss successes.

5 Testing Strategy and Configuration Management

5.1 Testing Strategy

- **Unit Testing**: Individual components tested using JUnit for Java backend and Jest for React components.
- **Integration Testing**: Ensures modules work together, using Spring's testing framework and React Testing Library.
- **System Testing**: Validates the entire system's functionality, utilizing tools like Selenium.
- **User Acceptance Testing (UAT)**: Final phase where users ensure the application meets their needs.

5.2 Configuration Management

- **Version Control**: Git for branching and merging.
- **Dependency Management**: Gradle for Java and npm for JavaScript, ensuring consistent environments.

5.3 Branching Strategy

- Implement a **Git Flow model**, detailing feature, develop, release, and hotfix branches to manage development effectively and maintain production stability.

5.4 Quality Assurance

- Integrate continuous integration/continuous deployment (**CI/CD**) practices for ongoing quality checks.
- Enforce coding standards and perform code reviews to maintain high code quality.