Wanderers
Design Report on Software Maintainability

Version 2.0

By: Hazim Khoiruddin, Isaac Chun Jun Heng, J'sen Ong Jia Xuan, Kim Seo Jin, Leong Mininn Miko, Raghav Rajendran Nair, Yu Zi Hao Albert

Document Change Record

Revision	Description of Change	Approved by	Date
0.1	Initial Template	Albert Yu	20/03/2025
1.0	Revised Edition	Isaac Chun	24/03/2025
1.1	Fixed formatting issues, typos and added missing content	Kim Seo Jin	25/03/2025
2.0	Added diagrams and refine content for grammatical flow	Albert Yu	26/03/2025

Contents

1. Design Strategies

1.1 The Planning Phase Before Development

Our project team brings together members with relevant past experiences, creating a foundation for informed decision-making. We systematically analyse similar past projects as reference points to anticipate improvements and identify potential pitfalls. By leveraging our team's diverse expertise in various areas such as full stack development, project management and DevOps, we develop comprehensive plans that address both immediate needs and future requirements.

Recognizing that ***Wanderers*** will likely experience increased traffic as adoption grows, we have proactively identified scalability as a critical factor for future exploration and development. The application's heavy dependence on user experience led us to adopt the **Client-Server architectural model**. This infrastructure framework particularly benefits our Graphic User Interface design by enabling function modifications and additions without disrupting other system components, thereby enhancing maintainability.

1.1. The Process of Developing

Our team implements Agile development practices to enhance software maintainability throughout the development lifecycle. We operate in short, focused iterations known as sprints that deliver incremental value while maintaining high quality standards. This approach allows us to regularly reassess priorities and make necessary adjustments based on stakeholder feedback and evolving requirements. By breaking development into manageable sprints, we create natural inspection points for code quality, architecture integrity, and adherence to maintainability standards.

1.2. Testing Framework

Our testing strategy integrates seamlessly with our Agile workflow, emphasizing continuous validation rather than end-stage verification. Each user story / issue includes an acceptance criterion that serve as the foundation for tests, ensuring features meet both functional requirements and maintainability standards. Test automation runs with each integration when a commit is pushed into a branch (via GitHub Workflows), providing immediate tests feedback on code quality and potential maintenance issues. This continuous testing approach identifies problems early when they are less costly to address and prevents the accumulation of technical debt that would complicate future maintenance efforts.

1.3. Sprint Retrospectives for Maintenance Improvement

We conduct regular structured retrospectives at the end of each sprint (2 weeks) to identify challenges and implement process improvements. These sessions provide a formal mechanism for the team to reflect on code quality, documentation standards, and technical practices that impact long-term maintainability. Insights gained during retrospectives translate into actionable

items for subsequent sprints, creating a continuous improvement cycle for our maintenance practices. This regular reflection ensures that maintainability remains a priority and evolves alongside the application's functionality.

1.4. Agile Artifacts: Documentation

Following our Agile methodology, documentation evolves continuously alongside the codebase rather than being created as a byproduct. We maintain living documentation that captures architectural decisions, design patterns, and component interactions, updating these resources as the system evolves. Our definition of "done" for user stories includes updating relevant documentation, ensuring it remains accurate and valuable for future maintenance activities. Through this approach, documentation serves as an effective knowledge transfer mechanism that supports sustainable long-term maintenance.

2. Architectural Design Patterns

Wanderers implements a three-tier architecture that provides clear separation of concerns between the user interface, application logic, and data management components. This architectural approach enables independent management, updating, and scaling of each tier without affecting the others, enhancing overall system maintainability and flexibility.

The **presentation** tier utilizes Next.js to deliver optimized HTML, CSS, and JavaScript content to users. This modern front-end framework efficiently renders the user interface in a minified and visually appealing format. The presentation tier communicates exclusively with the application tier through well-defined APIs, maintaining proper separation and preventing direct access to the database.

Express.js JavaScript Web Framework powers the **application** tier, serving as the central component for processing business logic and handling user inputs. This middleware layer manages all communication between the presentation and database tiers, ensuring proper data validation and transformation. The application tier executes database queries to retrieve itinerary details and user profile information, applying appropriate business rules before returning results to the presentation layer.

The **database** tier houses the application's data management system, implemented using Supabase. This platform facilitates efficient storage and retrieval of both relational data and complex objects such as user-uploaded photographs. By isolating data storage functions in a dedicated tier, the architecture enables specialized optimization of database performance and security without affecting other system components.

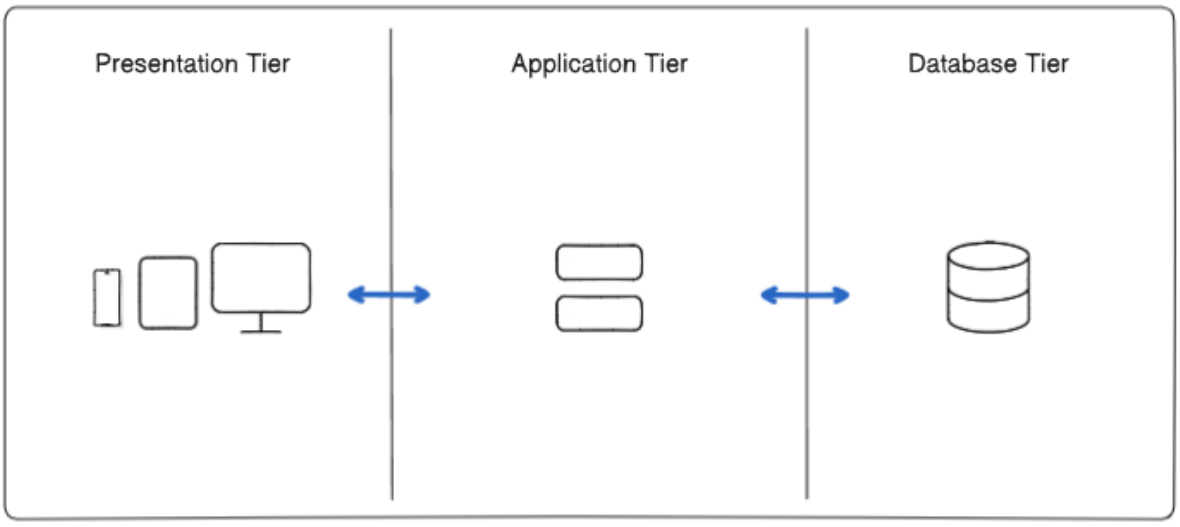


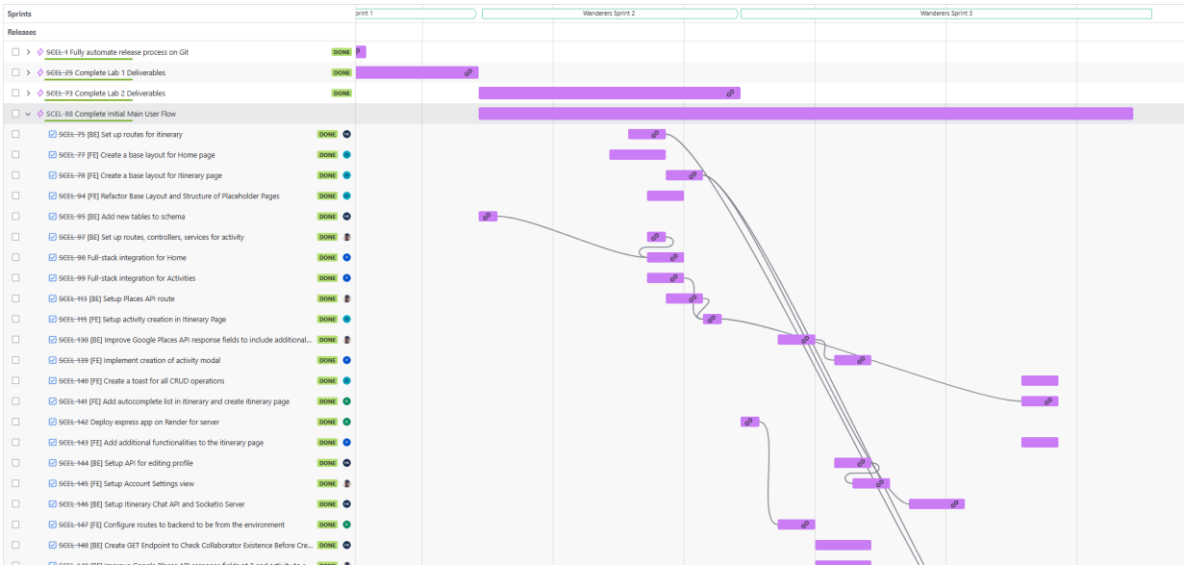
Figure 1: Wanderer layered architecture diagram

3. Software Configuration Management Tools

This is where we will discuss version control management and tracking on who made what changes and when.

3.1. Jira

Jira serves as our primary project management and issue tracking system. It provides comprehensive tracking of who made what changes and when. The system seamlessly integrates with GitHub, creating direct links between commits, pull requests, and their corresponding issues, which establishes clear traceability from initial requirements to final implementation. Jira's time tracking capabilities allow us to measure and analyze development effort across tasks and projects, providing valuable insights for resource allocation and future planning. Using Jira also allows for timeline planning and prioritization in sprint using timelines.



3.2. GitHub

GitHub is a source code hosting platform using the distributed version control and source code management Git. GitHub is chosen for its familiarity and support provided by various IDE applications. GitHub also supports issue tracking similar to a ticketing system. Whether it's a software bug, code enhancement or documentation, users can open an issue, label them appropriately and assign them for other team members to resolve. All users involved will receive timely updates on the progress of the issue. It features branch protection rules to enforce code reviews, detailed history tracking of all code changes (track difference between commits). In our iteration, we enforce the usage of PRs such that technical reviews must be conducted on each feature branch, improving code quality.

3.3. OneDrive

Microsoft OneDrive is used as a file storage and for the backup of documents

initially created. This service allows users to share and store files within the group easily. Furthermore, OneDrive allows users to edit documents concurrently and supports version contro