Revisions			
Version	**Description**	**Date**	**Person**
1.0	The document was created.	02.12.2023	Elif Beril Sayli Özde Uysal Annie Yang
1.1	The document was updated for Iteration 3.	16.02.2023	Elif Beril Sayli Özde Uysal Annie Yang

Samwise Service App
Design

This template describes how the design can be organized to be understood from multiple perspectives. It also provides suggestions for how patterns and descriptions of small, reusable interactions can be used to minimize redundancy.

It is important not to think of design as "a document." Design information that is worth keeping for some duration must have a long-lived form. But that form might be as a repository in a visual modeling tool, or as subdirectories of whiteboard diagrams captured with a digital camera, or as an actual document that provides structure for images taken from a myriad of sources.

This template describes the information that should be conveyed. Typically, it works best to convey the information graphically (either with UML or another unambiguous notation), or at least in words, at an abstract level. You can enhance this with code examples, but best not to render the design solely at the code level.

The structure of the design is suggested in this template.

Design structure

The architectural design is composed of three layers: UI Layer, Domain Layer and Technical Services Layer. In the UI Layer, HTML framework is included. In the Domain Layer, entities representing use cases such as User, Service, Profile, Calendar, Payment, Filtering and Review are included. In the technical services layer, Express.js, MySQL and Google Calendar are included.

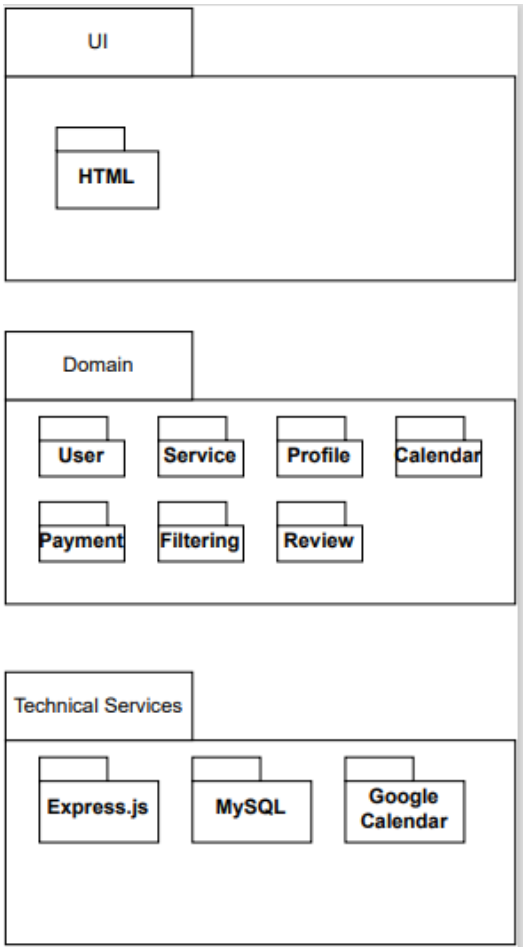


Figure 1. Layered Architecture

Subsystems

There are no subsystems within the system.

2. Patterns

2.1 [Layered Pattern]

Overview

The Layered Architecture pattern organizes the SamWise App into distinct layers, each responsible for specific functionalities. This pattern promotes separation of concerns and modularity, facilitating maintainability and scalability. The layered structure provides better organization to implement code and promotes a clear hierarchy of

responsibilities of main parts. The intention behind this decision is to structure the application into logical layers, promoting a modular and scalable architecture. The motivation is that the Layered Architecture pattern addresses the need for a structured approach to handling data, user interactions, and business logic independently, allowing for easier development and adaptability while maintaining a clear separation of responsibilities.This pattern is applicable to complex applications like SamWise, where a clear separation of data management, user interface, and business logic is essential for robust development and future enhancements.

Structure

Data Layer:

Responsibility: Data storage and retrieval.

Components: Database systems, data models, data access logic.

Presentation Layer:

Responsibility: User interface and interaction.

Components: UI components, views, controllers, user interface logic.

Business Logic Layer:

Responsibility: Core application logic and business rules.

Components: Service classes, use cases, application-specific logic.

Behavior

Data Flow:

Presentation Layer interacts with the Business Logic Layer to request and display data.

Business Logic Layer communicates with the Data Layer to fetch or store data.

User Interaction:

Presentation Layer handles user input and communicates with the Business Logic Layer to execute corresponding actions.

Business Logic Layer processes user requests, applying business rules as needed.

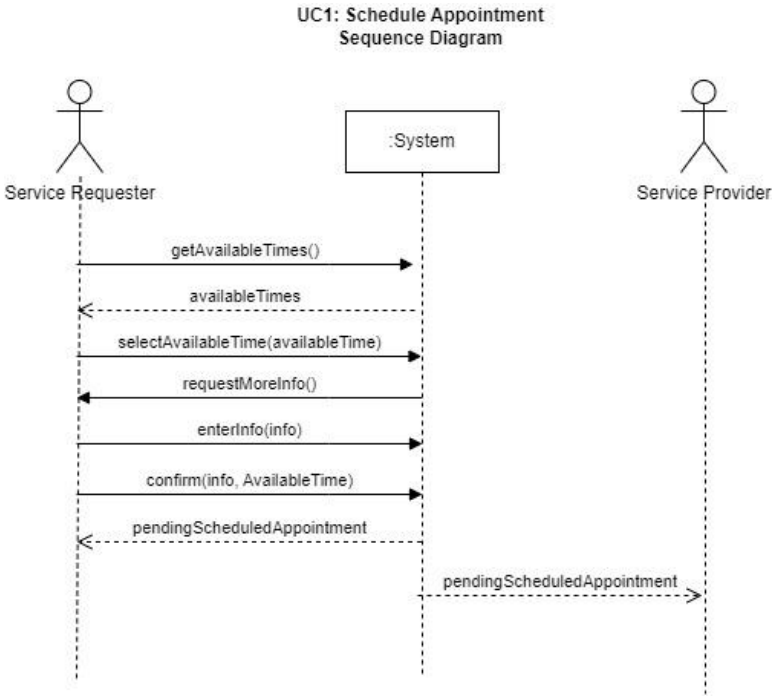
Example

For example, a scenario where a user views a service through the SamWise App. The Presentation Layer collects the request details, communicates with the Business Logic Layer to validate and process the request, and finally, the Business Logic Layer interacts with the Data Layer to retrieve the data from the database.

3. Use-case realizations

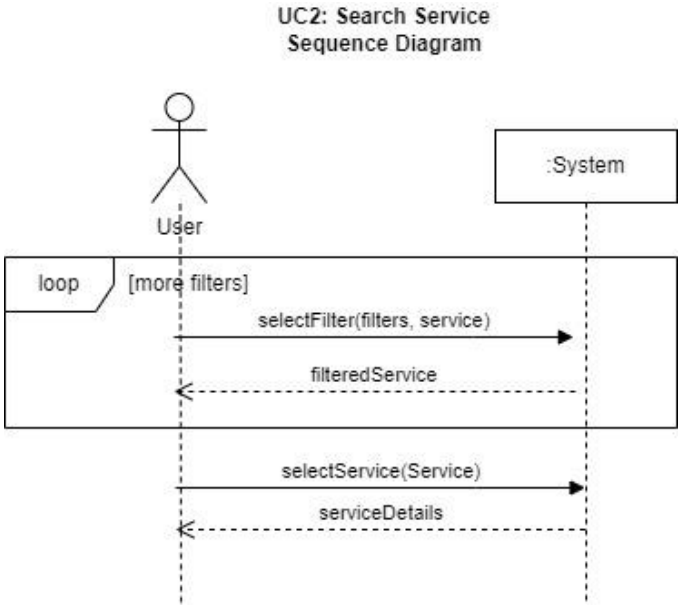
3.1 [Realization of Use-case 1]

Basic Scenario



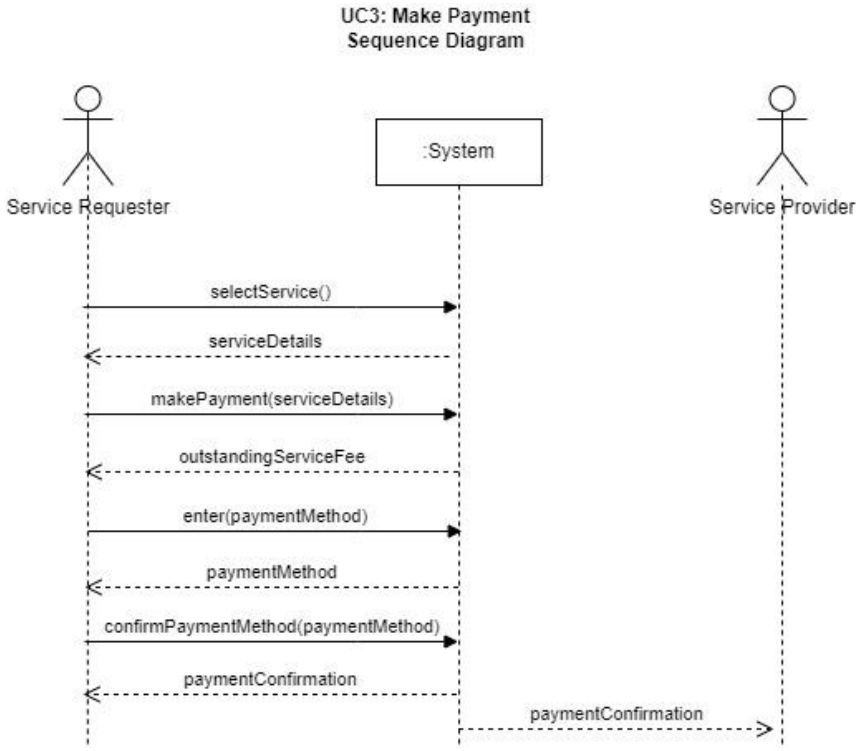
3.2[Realization of Use-case 2]

Basic Scenario



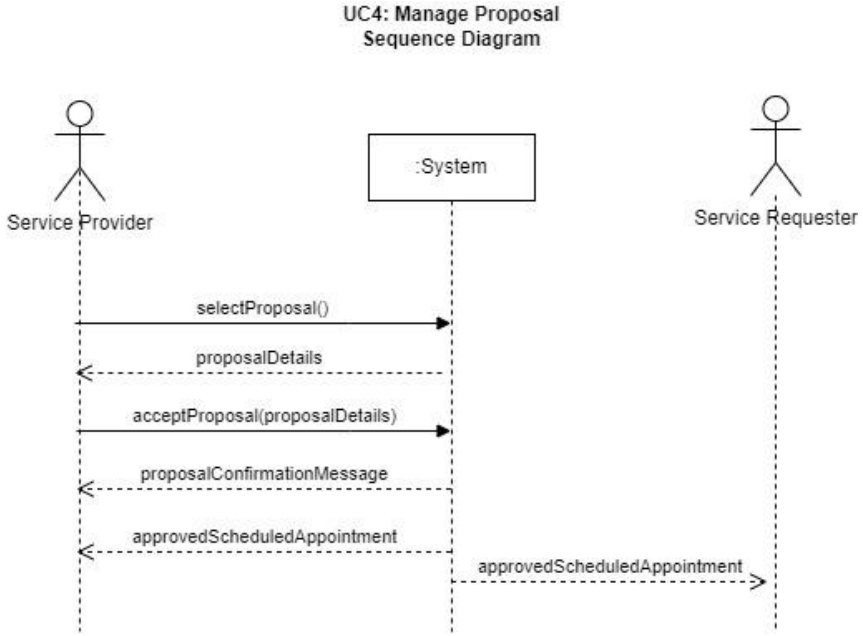
3.3[Realization of Use-case 3]

Basic Scenario



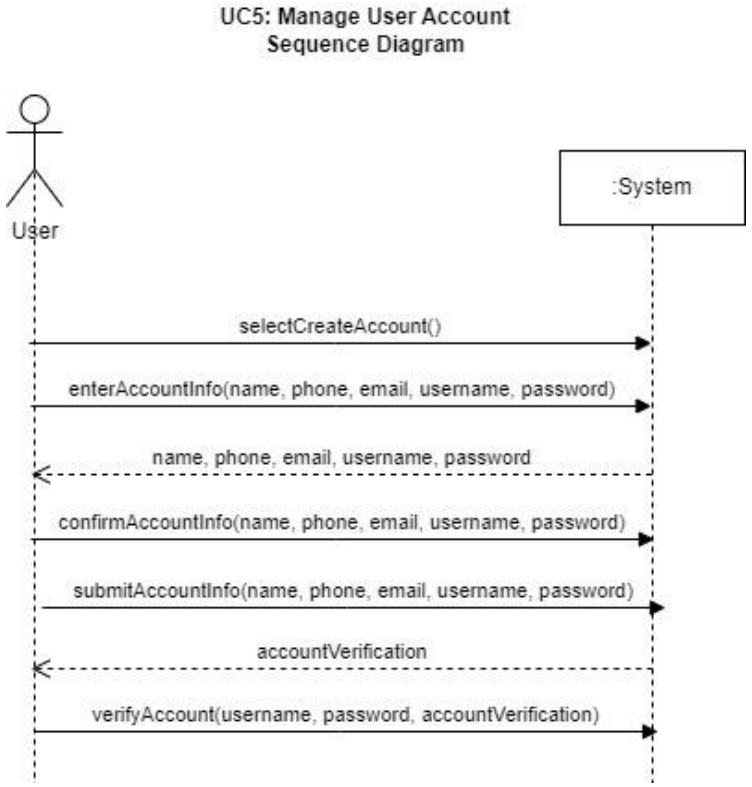
3.4[Realization of Use-case 4]

Basic Scenario



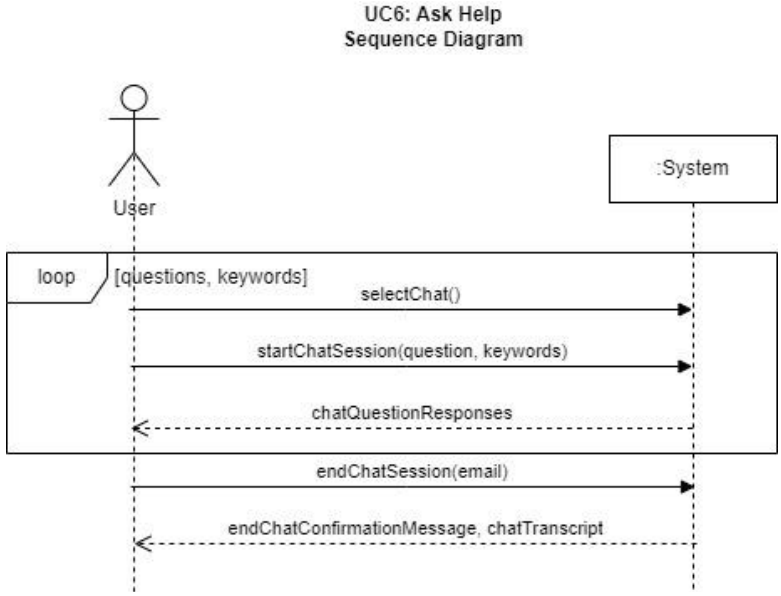
3.5[Realization of Use-case 5]

Basic Scenario



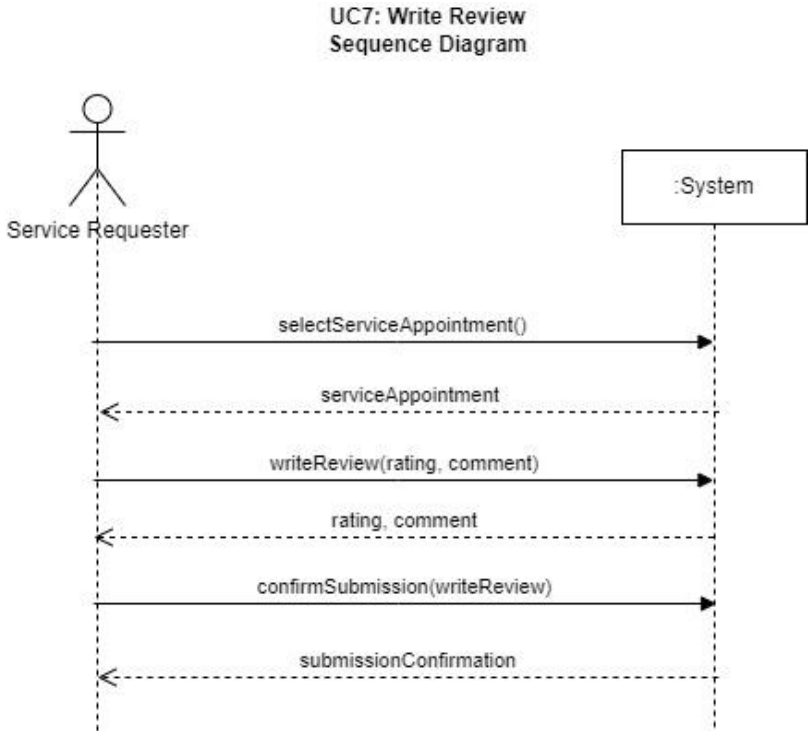
3.6[Realization of Use-case 6]

Basic Scenario



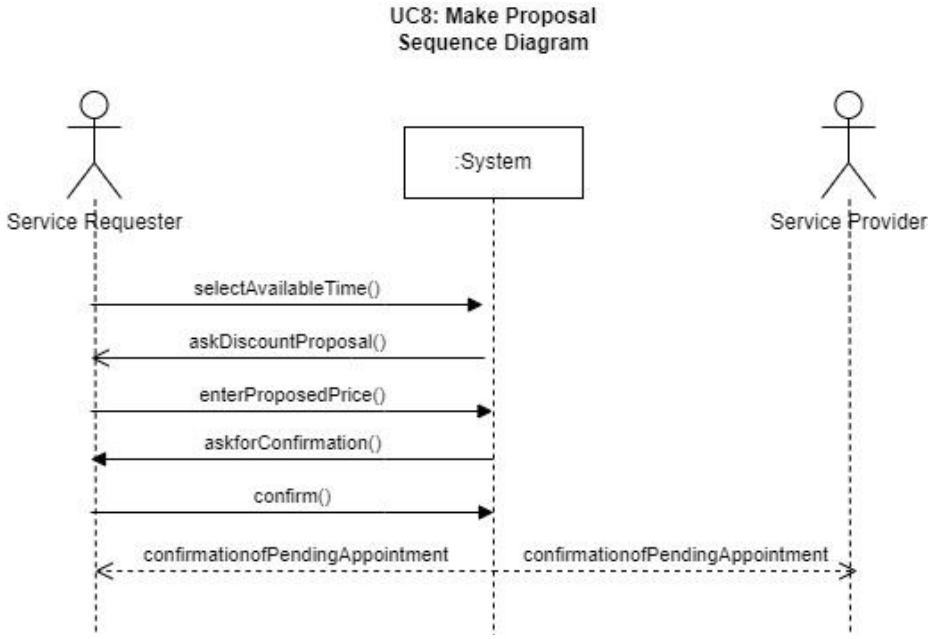
3.7[Realization of Use-case 7]

Basic Scenario



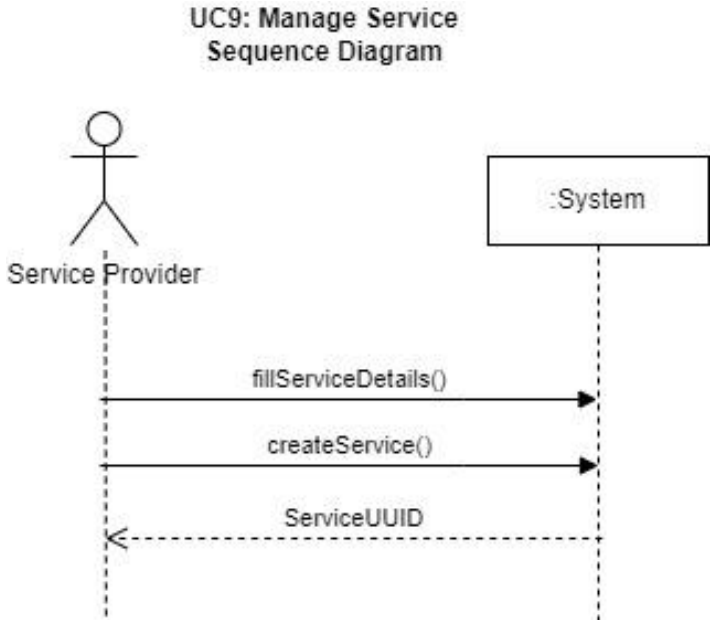
3.8[Realization of Use-case 8]

Basic Scenario



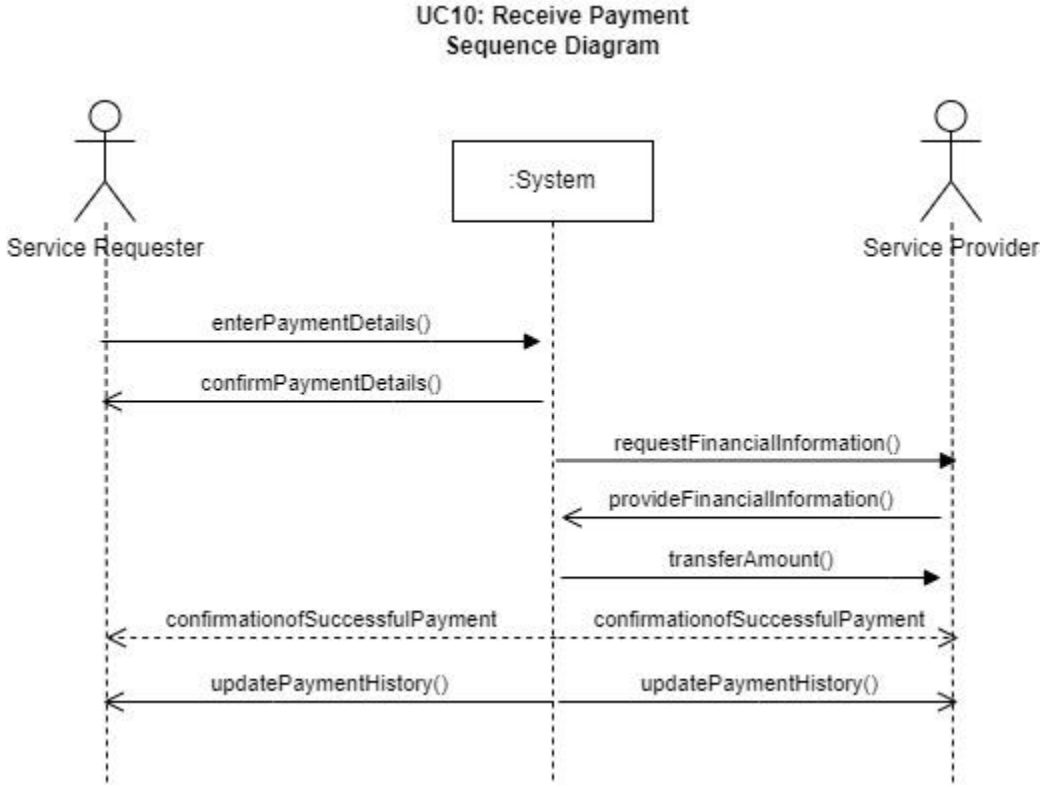
3.9[Realization of Use-case 9]

Basic Scenario



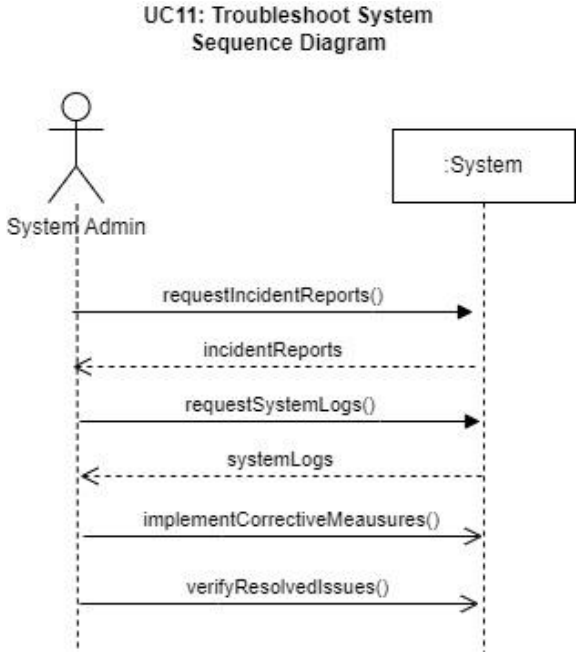
3.10[Realization of Use-case 10]

Basic Scenario



3.11[Realization of Use-case 11]

Basic Scenario



3.12[Realization of Use-case 12]

Basic Scenario