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
1.2	Only use cases realized in Iterations 1-3 are displayed (UC #5, 9, 2). MSS/Basic Flow with User, Behavior, Attribute, Relationship subsections were added to the UC Realizations.	23.12.2023	Elif Beril Sayli Annie Yang
1.3	Guidelines were removed and introduction was added. Diagrams were updated.Page navigation and source code organization were added.	01.01.2024	Annie Yang Elif Beril Sayli

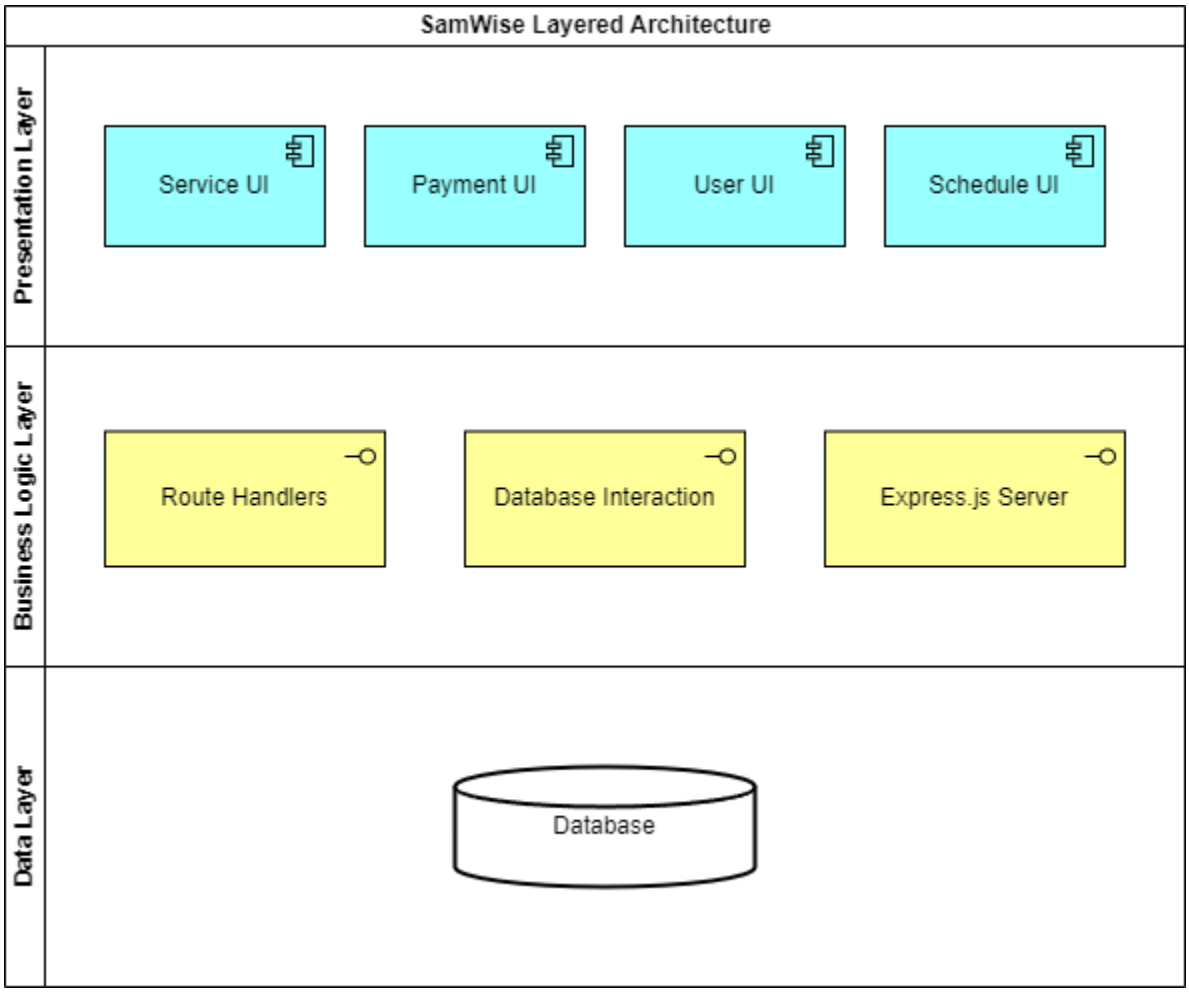
Samwise Service App
Design

This document provides a thorough understanding of the design of the Samwise Service App. It describes the system from multiple perspectives to elaborate on relevant patterns, interactions, and behaviors that constitute the system. The Design Document is organized into Design Structure, Patterns, Use Case Realizations, Page Navigation, and Source Code Organization sections so that the realization of the system is understandable.

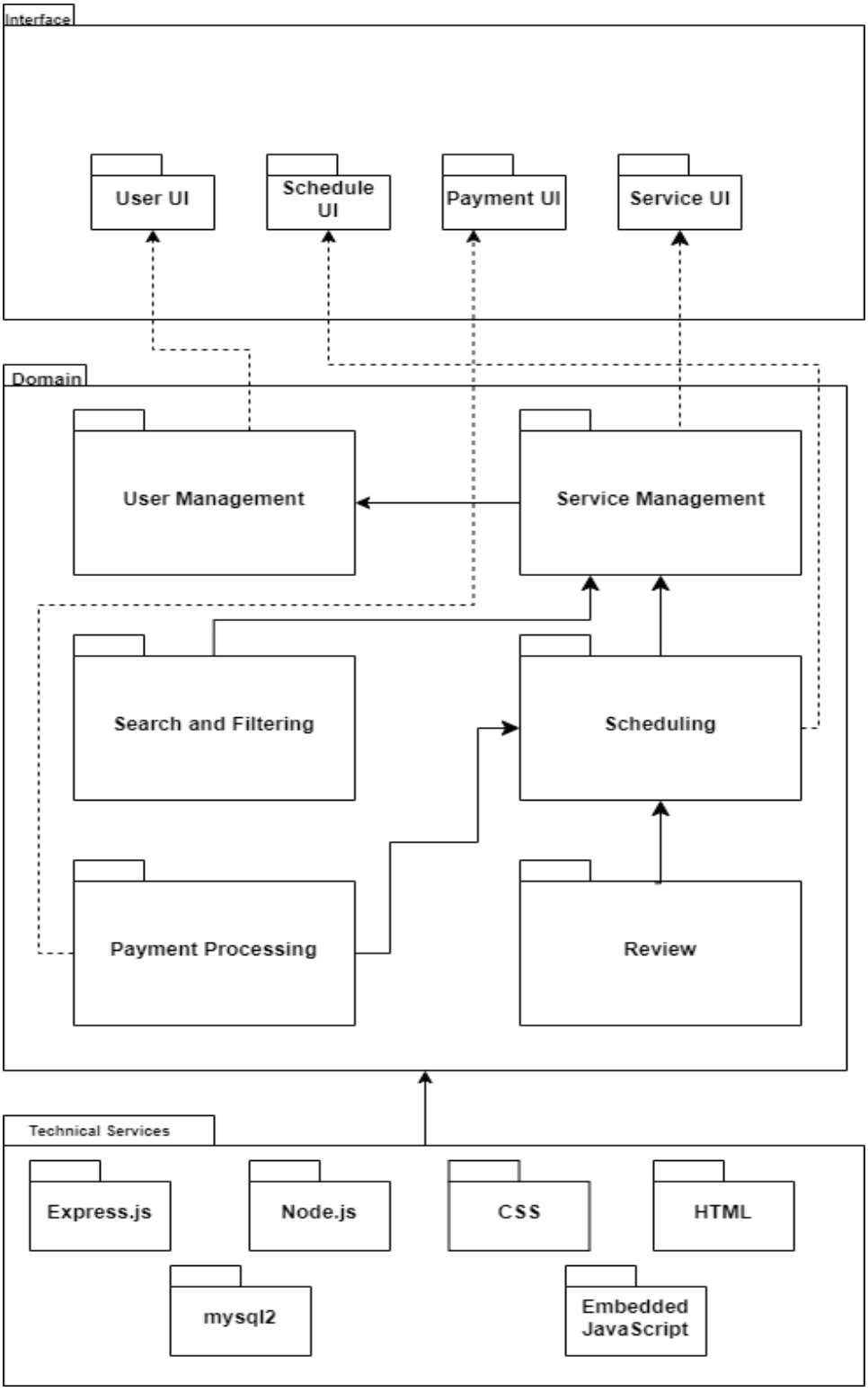
1. Design Structure

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 are included.

1.1 Layered Architecture



1.1 Logical Architecture



1.3 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: Schedule Appointment

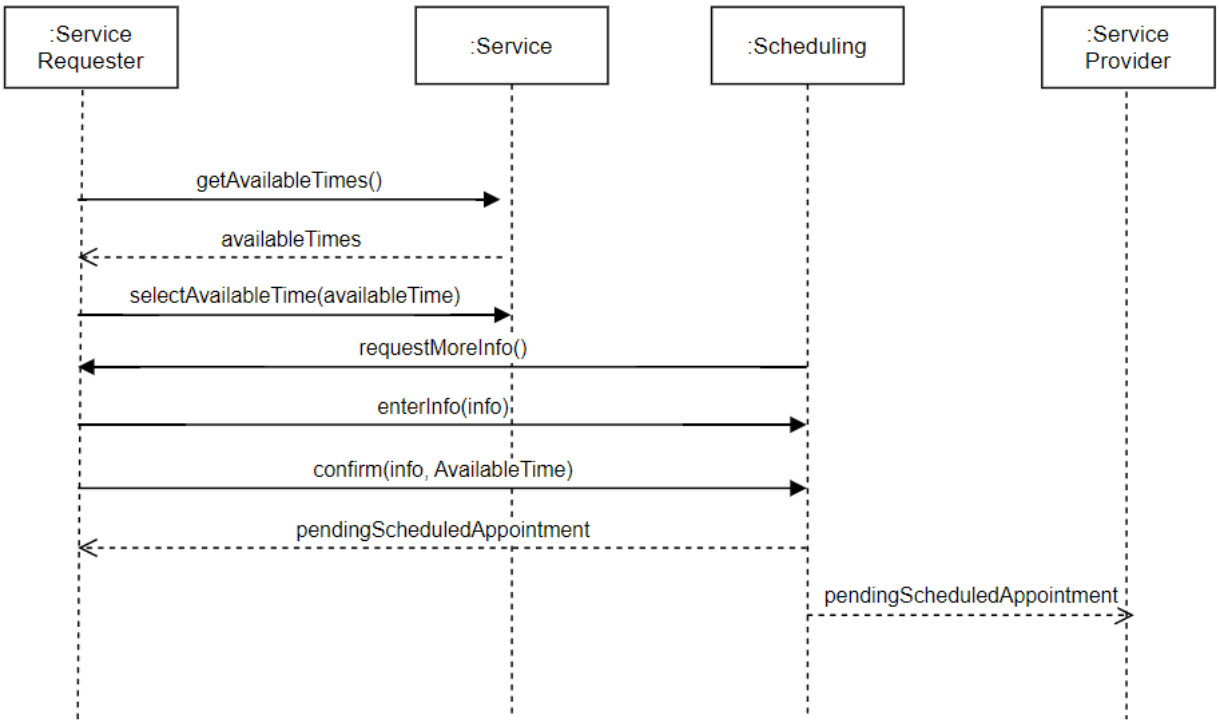
Users

- The participants consist of the Service Requester who wants to schedule a service offered by a Service Provider.
- Behavior: The Service Requester takes action on a service they want to schedule.
- Relationship: The user initiates the request of a service; inputs personal information; and interacts with the user interface.
- Attributes:
    - User Attributes: user_id, username, password, email, phone_number, status, account_type
    - Service Attributes: service_id, service_name, type, available_times, description, price, provider_id
    - Scheduling Attributes: scheduling_id, requester_id, provider_id, reservation_date, scheduled, approved

Basic Scenario

1. The Service Requester wants to schedule an appointment for a service they want to receive after they perform Use Case 2: Search Service.
2. The Service Requester clicks the available time for the service.
3. The System shows the available time for the service.
4. The Service Requester selects the desired available time.
5. The System asks the Service Requester for additional information.
6. The Service Requester provides additional information.
7. The System asks the Service Requester to confirm the additional information and available time selection.
8. The Service Requester confirms the additional information and available time selection.
9. The System sends the Service Requester a confirmation of the pending scheduled appointment.
10. The System sends the Service Provider a confirmation of the pending scheduled appointment.
11. The use case ends.

UC1: Schedule Appointment
Sequence Diagram



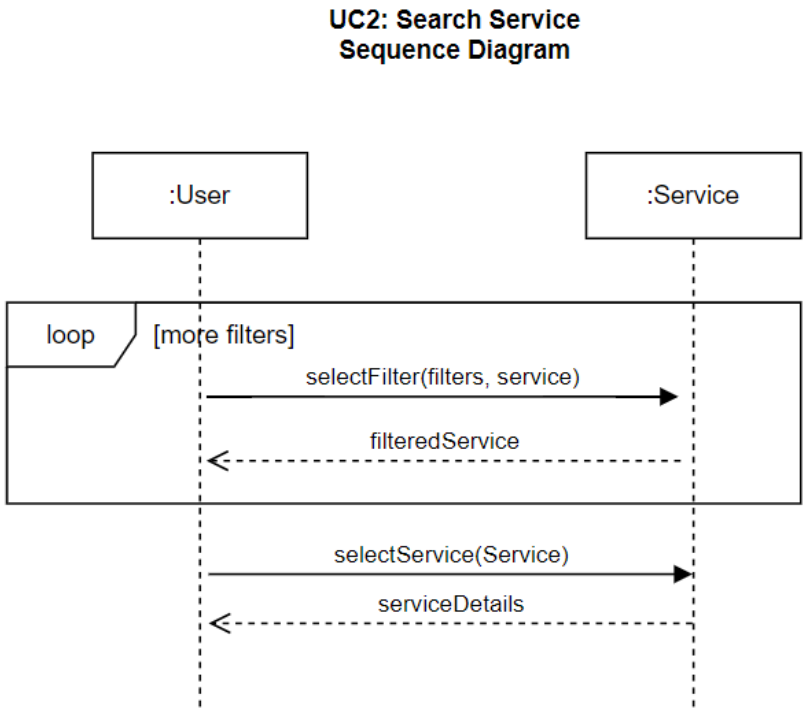
3.2 Realization of Use Case 2: Search Service

Users

- The participants consist of any individual (registered or unregistered) who uses the Samwise Service App to search for home maintenance and care services.
- Behavior: The user searches for an existing service.
- Relationship: The user initiates a search for an existing service and interacts with the user interface.
- Attributes:
    - Service Attributes: service_id, service_name, type, available_times, description, price, provider_id

Basic Scenario

1. A User wants to search for services on the Samwise Service App home page.
2. The System displays a search feature with filtering options.
3. The User selects one or more of the available filtering options.
4. The System applies the chosen filters to the search results.
5. The System displays results that match the User's criteria.
6. The User selects a result.
12. The System displays details about the selected result.
13. The use case ends.



3.3 Realization of Use Case 5: Manage User Account

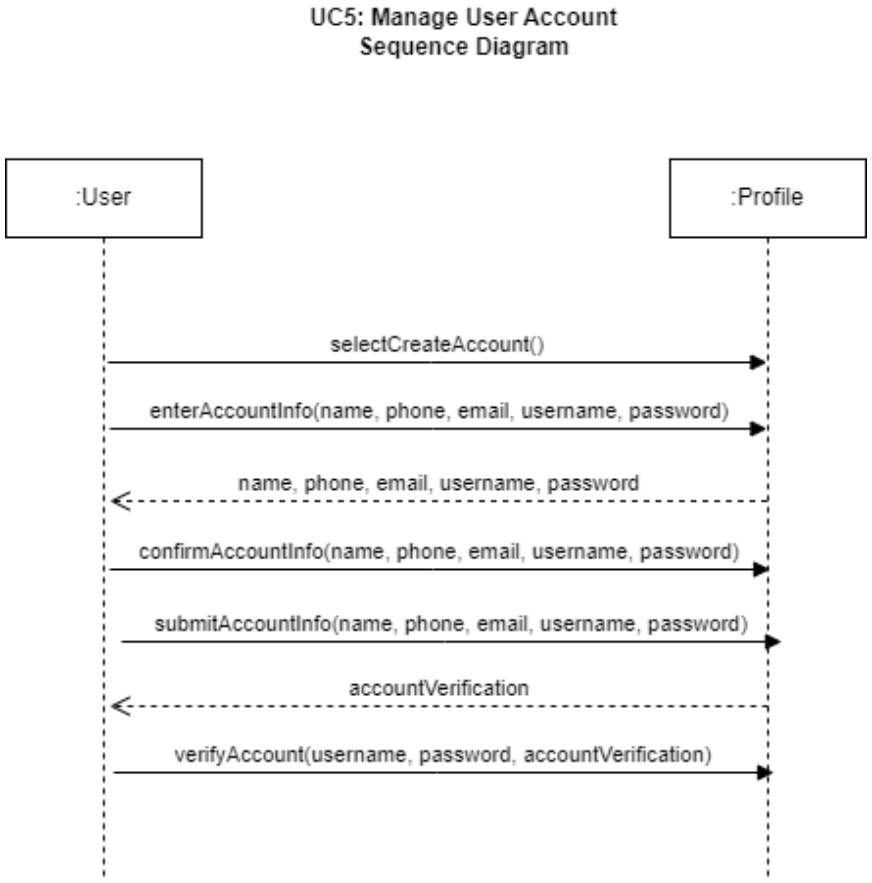
Users

- The participants consist of any individual who wants to create an account to use the system.
- Behavior: The user creates a new account .
- Relationship: The user initiates the creation of an account, inputs their account information, and verifies their credentials.
- Attributes:
    - User Attributes: user_id, username, password, email, phone_number, status, account_type

Basic Scenario

1. The use case begins when the User accesses the application and wants to create a new account.
2. The System displays a page to create a new account by entering in account information.
3. The User inputs account information.
4. The System displays the account information.
5. The Systems asks the User to confirm and submit.
6. The User confirms and submits.
7. The System sends a verification to the User's contact information.
8. The System displays a verification screen for the User to enter a verification.
9. The User enters the verification.
10. The System verifies the User's identity.
11. The System directs the User to the sign in page.

12. The use case ends.



3.4 Realization of Use Case 9: Manage Service

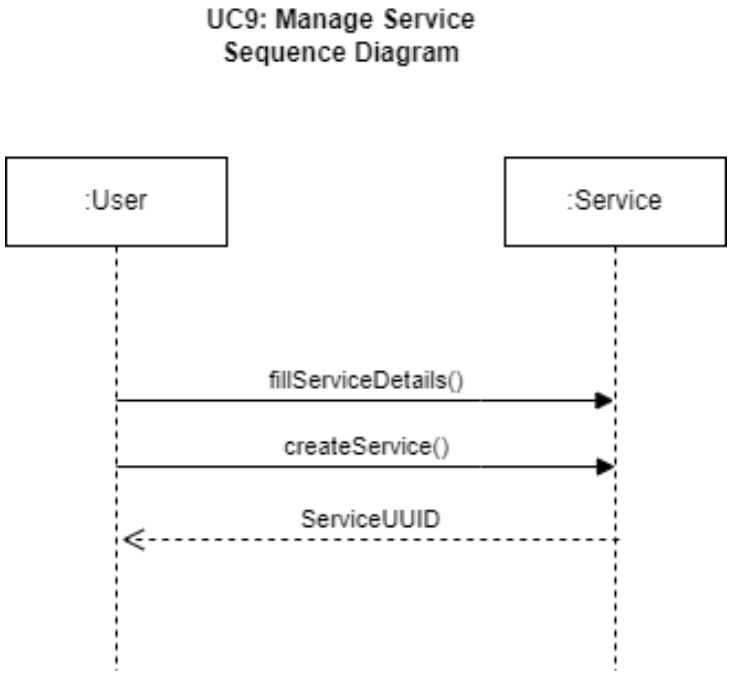
Users

- The participants consist of the Service Provider who wants to manage (CRUD) a service they offer.
- Behavior: The Service Provider manages the service they provide.
- Relationship: The user initiates the creation, editing, and deletion of a service; inputs service details information; and interacts with the user interface.
- Attribute:
    - User Attributes: user_id, username, password, email, phone_number, status, account_type
    - Service Attributes: service_id, service_name, type, available_times, description, price, provider_id

Basic Scenario

1. The Service Provider wants to manage a service.
2. The System displays a page where the User can enter the service information.
3. The Service Provider enters Service Details.
4. The Service Provider submits the Service Details.
5. The System generates a UUID for the created service.
6. The System displays the created service.

7. The use case ends.



3.5 Realization of Use Case 12: Approve Appointment

Users

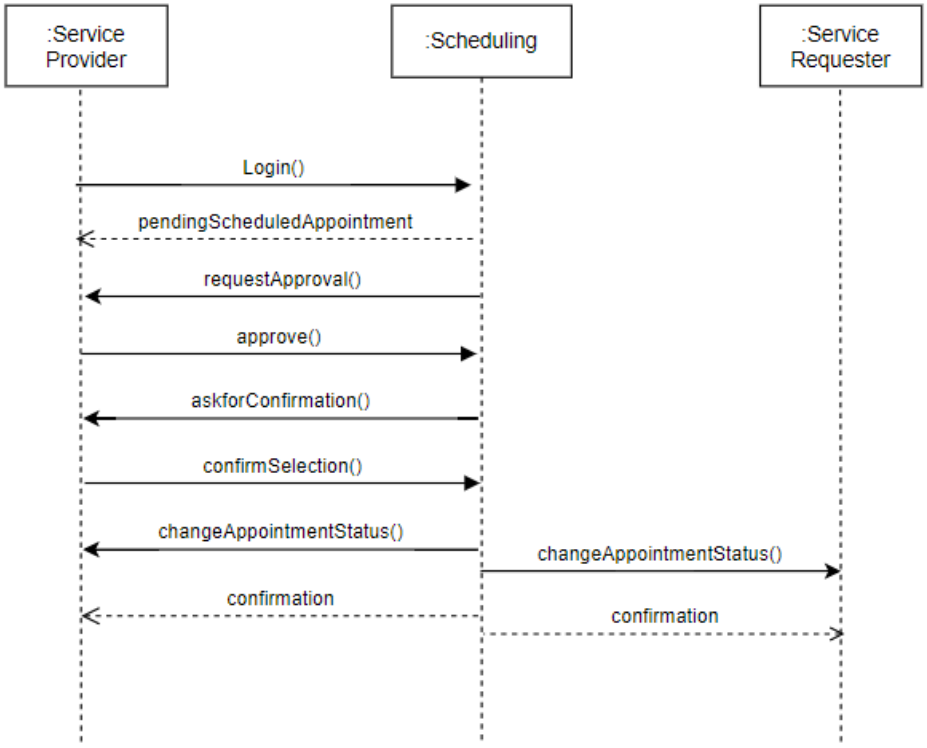
- The participants consist of the Service Provider who wants to approve or decline a pending scheduled appointment that they offered.
- Behavior: The Service Provider approves or declines their offered service which was requested by a Service Requester.
- Relationship: The user interacts with the user interface; the user selects a status for the pending service status.
- Attributes:
    - User Attributes: user_id, username, password, email, phone_number, status, account_type
    - Scheduling Attributes: scheduling_id, requester_id, provider_id, reservation_date, scheduled, approved

Basic Scenario

1. The use case begins after the Service Provider logs into the System.
2. The system shows a confirmation that the Service Provider has a Pending Scheduled Appointment that needs approval.
3. The Service Provider views the Pending Schedule appointment details.
4. The System presents options to Approve or Decline.
5. The Service Provider wants to approve the appointment.
6. The Service Provider selects Approve.
7. The System asks the Service Provider to confirm the selection.
8. The Service Provider confirms the selection.

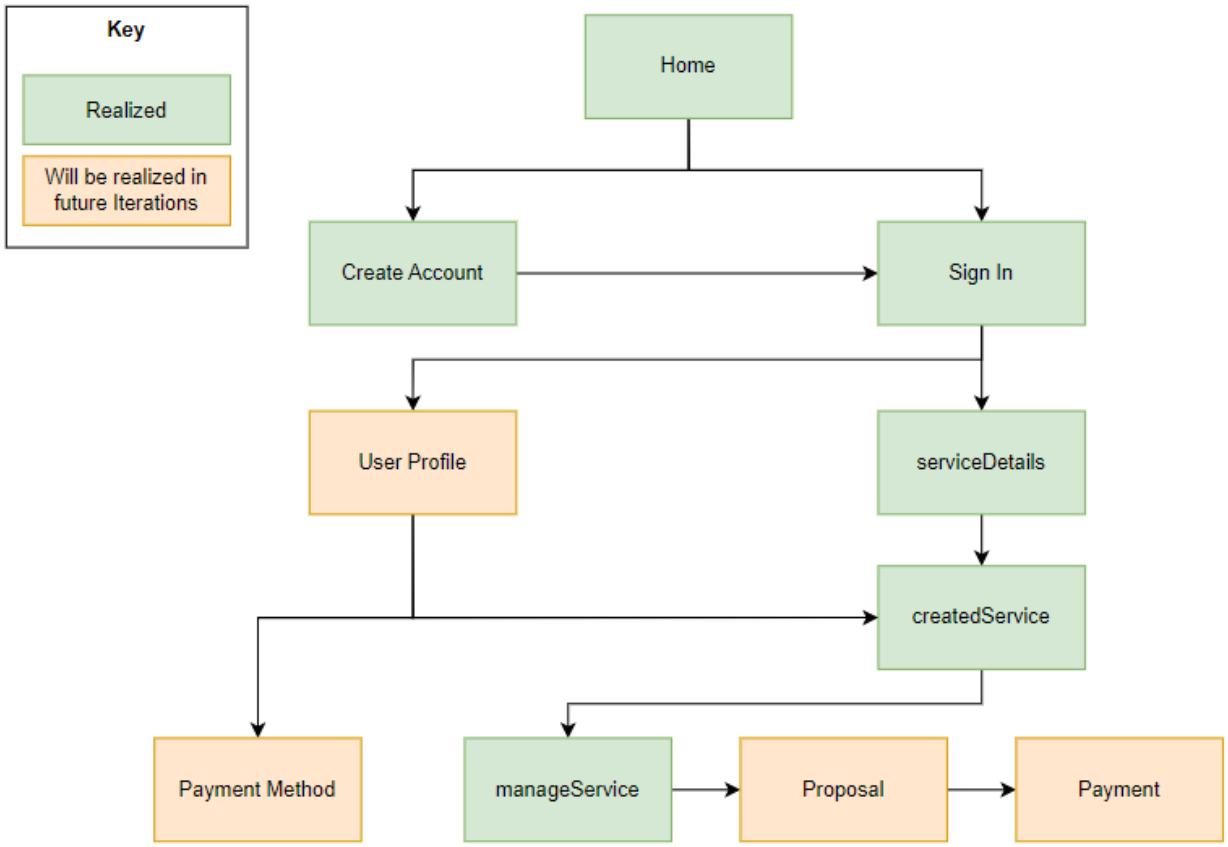
9. The System changes the scheduled appointment status from Pending to Approved.
10. The System sends a confirmation to the Service Provider.
11. The System sends a confirmation to the Service Requester.
12. The use case ends.

UC12: Approve Appointment
Sequence Diagram



4. Page Navigation

This section depicts the organization of navigation within the application. The realization status of the pages are indicated by the corresponding color key.



5. Source Code Organization

This section outlines the organization of the source code for the Samwise Service App. The primary file, index.js, plays a crucial role in managing business logic layers, route handling, and database interactions. Below is an overview of its key components:

- **Server Initialization:**
  The server listens on port 3000 for localhost. Server-side code connects to the MySQL database on startup, creating necessary tables (users, services, scheduling).

- **User Registration (/register):**
  Handles POST requests to register a new user, validates password confirmation, and inserts user data into the users table.

- **User Login (/login):**

Handles POST requests for user login, queries the database to check credentials, and redirects to the service page if successful.

- **Service Management (/service):**
  Handles service editing, scheduling, approval, decline, searching, details, creation, and deletion.

- **Service Search (/search):**
  Handles POST requests for searching services.

- **Service Details (/service/details):**
  Displays details of the searched services.

- **Service Creation (/service/create):**
  Handles POST requests to create a new service, generating a UUID for the service.

- **Service Deletion (/service/delete):**
  Handles POST requests to delete a service.
- **Static Files (/):**
  Serves static files from the 'public' directory and sends the 'index.html'.

- **Default Route (/service):**
  Sends the 'manageServices.html' files.

- **Static Files (/public):**
  Contains images and CSS files providing user interface features for the front-end.

Client-Side Code: At the /public folder, there are images and css files which provide user interface features for the front-end. The client-side code consists of HTML pages:

- **Index.html:**
  Contains the main page front-end source code, including creating a new account and login user features.

- **ManageServices.html:**
  Provides front-end features for searching with service type, creation, deletion, and editing features of services.

- **CreatedService.html:**
  Features front-end code for the successful creation of a new service.

- **/views/ServiceDetails.ejs:**
  Contains front-end codes to display detailed information about each service after searching.