Software Engineering: Software Design

Student No: 100571823

# Wireframes

The following wireframes provide a guide for the layout of each of the pages of my intended software. A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# UML Diagrams

## Use Case Diagram

The following diagram displays the users and the system.

A diagram of a network

Description automatically generated

## Class Diagram

The following outlines the systems classes, showing their methods and relationships.

A group of white papers with black text

Description automatically generated

## Sequence Diagrams

Sequence showing the creation of a passenger in the system by a staff member:

A diagram of a diagram

Description automatically generated

Sequence showing the updating of passenger details in the system by a staff member:

A diagram of a diagram

Description automatically generated

Create Staff Member:

A diagram of a computer program

Description automatically generated with medium confidence

Edit Staff Details:

A diagram of a computer program

Description automatically generated

Airplane Assignment:

A diagram of a computer program

Description automatically generated with medium confidence

Sequence showing the booking of a flight in the system by a staff member:

A diagram of a computer

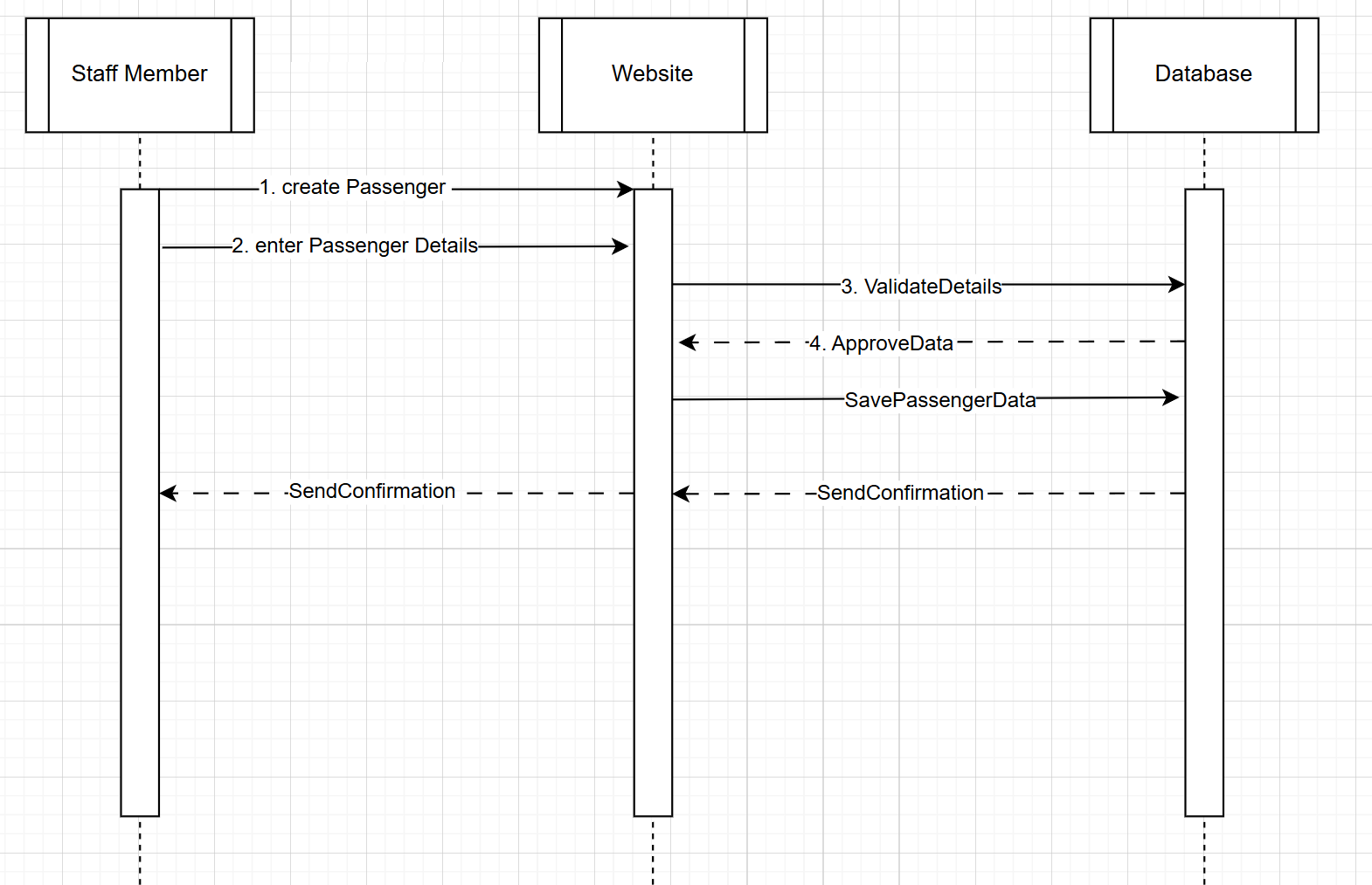
Description automatically generated with medium confidence

Flight Creation:A diagram of a diagram

Description automatically generated

## Communication Diagrams

Creation of a passenger:



Edit Passenger:

A diagram of a web site

Description automatically generated

Create Staff member:

A diagram of a diagram

Description automatically generated

Edit staff member:

A diagram of a diagram

Description automatically generated

Airplane Assignment:

A diagram of a diagram

Description automatically generated

Booking passenger to flight:

A diagram of a diagram

Description automatically generated

Creating a flight:

A diagram of a diagram

Description automatically generated

## Context Diagram

A diagram of a diagram

Description automatically generated

## Class Diagram

A diagram of a computer

Description automatically generated

## Deployment DiagramA diagram of a web service Description automatically generated

# Design Pattern

## Singleton

The Singleton Pattern ensures that a class has only one instance and provides a global point of access to it. In the airline management system, this pattern is particularly useful for managing the database connection. By ensuring there is only one instance of the database connection object, it prevents multiple connections that could lead to resource conflicts and inefficiency. This promotes efficient resource usage, simplifies configuration management, and ensures that changes to the connection settings affect all parts of the application simultaneously.

## Factory

The Factory Pattern provides a way to create objects without specifying the exact class of the object that will be created. In the context of the airline management system, this pattern is essential for passenger creation. Different types of passengers, such as economy class, business class, and first class passengers, have distinct attributes and preferences. Using the Factory pattern, a common interface can be defined for creating passenger objects, with specific implementations for each passenger type. For example, when a new passenger is added to the system, the Factory can generate an instance of the appropriate subclass (EconomyPassenger, BusinessPassenger, etc.) based on input parameters. This approach enhances flexibility and scalability, allowing the system to easily incorporate new types of passengers without significantly altering existing code as well as allowing better scalability in the future.

# Database Schema

A diagram of a data flow

Description automatically generated

# Architectural Model

To develop a comprehensive airline management system, adopting the Model-View-Controller (MVC) architecture is essential. In this architecture, the Model represents the data and business logic, managing entities like passengers, flights, staff, airplanes, and cities. The View handles the user interface, displaying flight schedules, booking information, and staff details via an HTML5-based web interface. The Controller acts as an intermediary, processing user inputs from the staff, updating the Model, and refreshing the View accordingly. Utilizing design patterns like Singleton for a single database connection, Factory for creating different staff roles, and Observer for updating the interface when data changes ensures reusability, maintainability, and decoupling.

The three-tier architecture is recommended in the following way -Presentation Tier: Staff workstations using browsers to interact with the system, Application Tier: A web server hosting the MVC-based application, processing business logic and the Data Tier: A MySQL database server managing data storage.

This setup enhances maintainability and scalability by isolating each layer, supports efficient handling of increasing loads, and secures the database by preventing direct access from users. This structured approach ensures that the system effectively manages airline operations, from passenger bookings and flight schedules to staff assignments and aircraft allocations.