# FlyAway (An Airline Booking Portal)

Course-end Project 2

DESCRIPTION

Project objective:

As a Full Stack Developer, design and develop an airline booking portal named as FlyAway. Use the GitHub repository to manage the project artifacts.

Background of the problem statement:

FlyAway is a ticket-booking portal that lets people book flights on their website.

The website needs to have the following features:

- A search form in the homepage to allow entry of travel details, like the date of travel, source, destination, and the number of persons.
- Based on the travel details entered, it will show the available flights with their ticket prices.
- Once a person selects a flight to book, they will be taken to a register page where they must fill in their personal details. In the next page, they are shown the flight details of the flight that they are booking, and the payment is done via a dummy payment gateway. On completion of the payment, they are shown a confirmation page with the details of the booking.

For the above features to work, there will be an admin backend with the following features:

- An admin login page where the admin can change the password after login, if he wishes
- A master list of places for source and destination
- A master list of airlines
- A list of flights where each flight has a source, destination, airline, and ticket price

The goal of the company is to deliver a high-end quality product as early as possible.

The flow and features of the application:

- Plan more than two sprints to complete the application
- Document the flow of the application and prepare a flow chart
- List the core concepts and algorithms being used to complete this application
- Implement the appropriate concepts, such as exceptions, collections, and sorting techniques for source code optimization and increased performance

You must use the following:

- Eclipse/IntelliJ: An IDE to code for the application
- Java: A programming language to develop the web pages, databases, and others
- SQL: To create tables for admin, airlines, and other specifics
- Maven: To create a web-enabled Maven project
- Git: To connect and push files from the local system to GitHub
- GitHub: To store the application code and track its versions

● Scrum: An efficient agile framework to deliver the product incrementally
● Search and Sort techniques: Data structures used for the project
● Specification document: Any open-source document or Google Docs

The following requirements should be met:
● The source code should be pushed to your GitHub repository. You need to document the steps and write the algorithms in it.
● The submission of your GitHub repository link is mandatory. In order to track your task, you need to share the link of the repository. You can add a section in your document.
● Document the step-by-step process starting from sprint planning to the product release.
● The application should not close, exit, or throw an exception if the user specifies an invalid input.
● You need to submit the final specification document which will include:
● Project and developer details
● Sprints planned and the tasks achieved in them
● Algorithms and flowcharts of the application
● Core concepts used in the project
● Links to the GitHub repository to verify the project completion

---

Project Planning:

- Define the project scope, requirements, and goals.
- Break down the development process into sprints.
- Create a project timeline and allocate tasks for each sprint.

User Interface Design:

- Design a visually appealing and user-friendly interface for the website.
- Create a homepage with a search form to input travel details.
- Design pages for flight search results, registration, payment, and confirmation.

Backend Development:

- Set up a development environment using Eclipse/IntelliJ.
- Use Java as the programming language for server-side development.
- Utilize Maven to create a web-enabled Maven project.
- Implement a web framework like Spring MVC or Java Servlets for request handling.
- Develop the necessary database schema using SQL for admin, airlines, flights, and user details.

Flight Search and Display:

- Implement a search algorithm that retrieves flights based on user input (source, destination, date).
- Fetch flight data from the database and display available flights with their ticket prices.
- Handle cases where no flights match the search criteria.

User Registration and Authentication:

- Create an admin login page with password change functionality.

- Implement user registration and store personal details securely.
- Provide authentication and session management for logged-in users.

Payment Integration:

- Set up a dummy payment gateway for testing purposes.
- Implement a payment flow where users can enter payment details securely.
- Simulate the payment process and generate a booking confirmation.

Admin Backend:

- Create an admin dashboard to manage the master lists of places, airlines, and flights.
- Allow admins to add, edit, or delete entries from these lists.
- Implement validation and error handling for admin operations.

Testing and Optimization:

- Perform unit testing for different components of the application.
- Implement exception handling to prevent application crashes.
- Optimize the source code using collections, sorting techniques, and efficient algorithms.
- Conduct user acceptance testing to ensure the application meets the requirements.

Documentation:

- Document the project details, including developer information.
- Outline the sprints completed and tasks achieved in each sprint.
- Provide flowcharts and algorithms used in the application.
- Include the GitHub repository link for verification.
- Create a specification document that summarizes the project.

**Requirements:**

## User-Facing Frontend:

- Homepage with a search form to enter travel details (date, source, destination, number of persons).
- Display available flights with ticket prices based on the entered travel details.
- Register page for users to provide personal details.
- Flight details page showing the selected flight and its information.
- Integration with a dummy payment gateway for payment processing.
- Confirmation page displaying the booking details.

## Admin Backend:

- Admin login page with password change functionality.
- Ability to manage master lists of places for source and destination.
- Ability to manage a master list of airlines.
- Manage a list of flights with details like source, destination, airline, and ticket price.

Technology Stack:

- Eclipse/IntelliJ as the IDE for coding.
- Java programming language for web development.
- SQL for creating database tables for admin, airlines, and other specifics.
- Maven for project management and dependency resolution.
- Git for version control and collaboration.
- Scrum as the agile framework for project management.
- Search and sort techniques for optimizing data retrieval and performance.

Goals:

- Develop a high-quality airline booking portal named FlyAway.
- Deliver the product incrementally through multiple sprints.
- Create a user-friendly and visually appealing interface for the website.
- Implement robust backend functionality for flight search, registration, payment, and confirmation.
- Provide an admin backend to manage master lists and flight details.
- Optimize the source code using appropriate algorithms and techniques.
- Ensure the application does not close, exit, or throw exceptions on invalid inputs.
- Document the project's progress, including sprints, algorithms, and flowcharts.
- Submit the final specification document and provide the GitHub repository link for verification.

**Sprint 1:**

Sprint Planning:

- Define project scope, goals, and requirements.
- Identify user stories and prioritize them.
- Assign tasks to team members.

User Interface Design:

- Design the homepage with a search form.
- Create basic HTML/CSS templates for the user-facing pages.
- Set up a responsive layout and navigation menu.

Backend Setup:

- Set up the development environment using Eclipse/IntelliJ.
- Create a Maven project structure.
- Configure the web framework (Spring MVC, Java Servlets, etc.).
- Set up the database schema using SQL.

Flight Search Functionality:

- Implement the flight search algorithm based on user input.

- Connect to the database and retrieve flight data.
- Display available flights with ticket prices on the search results page.

**Sprint 2:**

User Registration and Authentication:

- Create the registration page for users to enter personal details.
- Implement validation and secure storage of user information.
- Add authentication and session management for logged-in users.

Flight Selection and Payment:

- Develop the flight details page to show selected flight information.
- Integrate a dummy payment gateway for payment processing.
- Implement the payment flow and handle successful/failed payments.

Admin Backend - Master Lists:

- Design and create the admin login page.
- Implement password change functionality for the admin.
- Develop the backend for managing master lists of places and airlines.

**Sprint 3:**

Admin Backend - Flight Management:

- Create the admin dashboard for flight management.
- Implement CRUD (Create, Read, Update, Delete) operations for flights.
- Ensure proper validation and error handling for admin operations.

Testing and Optimization:

- Perform unit testing for different components of the application.
- Identify and fix bugs or issues.
- Optimize the source code using efficient algorithms and data structures.

Documentation and Finalization:

- Document the sprint progress, including completed tasks and challenges faced.
- Create flowcharts and algorithms used in the application.
- Prepare the final specification document summarizing the project.
- Ensure the GitHub repository is up to date with all the code changes.

---

**Sprint 1: Duration: 2 weeks**

**Tasks:**

Sprint Planning:

- Define project scope, goals, and requirements.
- Identify user stories and prioritize them.
- Assign tasks to team members.

User Interface Design:

- Design the homepage with a search form. (3 days)
- Create basic HTML/CSS templates for user-facing pages. (4 days)
- Set up a responsive layout and navigation menu. (2 days)

Backend Setup:

- Set up the development environment using Eclipse/IntelliJ. (1 day)
- Create a Maven project structure. (1 day)
- Configure the web framework (Spring MVC, Java Servlets, etc.). (2 days)
- Set up the database schema using SQL. (2 days)

Flight Search Functionality:

- Implement flight search algorithm based on user input. (3 days)
- Connect to the database and retrieve flight data. (2 days)
- Display available flights with ticket prices on the search results page. (3 days)

**Sprint 2: Duration: 2 weeks**

**Tasks:**

User Registration and Authentication:

- Create the registration page for users to enter personal details. (3 days)
- Implement validation and secure storage of user information. (2 days)
- Add authentication and session management for logged-in users. (2 days)

Flight Selection and Payment:

- Develop the flight details page to show selected flight information. (3 days)
- Integrate a dummy payment gateway for payment processing. (3 days)
- Implement the payment flow and handle successful/failed payments. (3 days)

Admin Backend - Master Lists:

- Design and create the admin login page. (2 days)
- Implement password change functionality for the admin. (1 day)
- Develop the backend for managing master lists of places and airlines. (4 days)

**Sprint 3: Duration: 2 weeks**

**Tasks:**

Admin Backend - Flight Management:

- Create the admin dashboard for flight management. (3 days)
- Implement CRUD operations for flights. (5 days)
- Ensure proper validation and error handling for admin operations. (3 days)

Testing and Optimization:

- Perform unit testing for different components of the application. (3 days)
- Identify and fix bugs or issues. (3 days)
- Optimize the source code using efficient algorithms and data structures. (4 days)

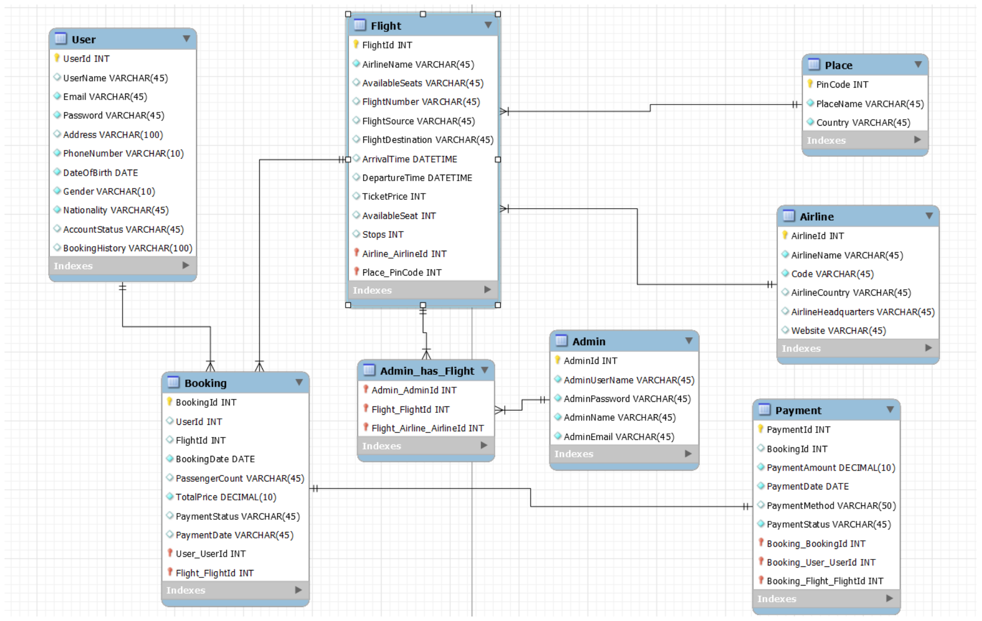
Documentation and Finalization:

- Document the sprint progress, including completed tasks and challenges faced. (2 days)
- Create flowcharts and algorithms used in the application. (2 days)
- Prepare the final specification document summarizing the project. (3 days)
- Ensure the GitHub repository is up to date with all the code changes. (1 day)

**Step 1**: **Open MY SQL workbench**: to create entity for User, Flight, Place, Airline, Booking, Admin & Payment.

- User: Represents a user of the application and stores their personal information.
- Flight: Represents a specific flight with attributes like flight number, source, destination, airline, ticket price, etc.
- Place: Represents a place or location, used for both source and destination in flight search.
- Airline: Represents an airline with attributes like name, code, and other relevant details.
- Booking: Represents a flight booking made by a user, including details like the user, flight, payment status, and booking confirmation.
- Admin: Represents an admin user with login credentials and privileges for managing the application.
- Payment: Represents a payment transaction made by the user for a booking, including details like payment amount, timestamp, etc.

File→ new model → add diagram



## Relationships between entities:

User and Booking: One-to-Many
Flight and Booking: One-to-Many
Place and Flight: Many-to-One
Airline and Flight: One-to-Many
Admin and Flight: Many-to-Many
Payment and Booking: One-to-One

**Step 2:** Create Various Entities

Entity1: Airline

Entity2: Flight

Entity3: Place

Entity4: Booking

Entity5: Payment

Entity6: Admin

Entity7: User

**Step 3**: Create Hibernate based configuration class

**Step 4**: Create Dao interfaces of all entities

DAO interfaces and DAO implementation classes are part of the DAO pattern (Data Access Object pattern) in Java. They work together to provide a structured way to interact with the data source, such as a database, and perform CRUD (Create, Read, Update, Delete) operations.

- DAO interfaces define the contract or interface that specifies the methods for accessing and manipulating data.
- They typically represent a specific entity or table in the database.
- DAO interfaces declare the CRUD operations and any other specific data access methods required by the application.
- The purpose of DAO interfaces is to provide a standardized and abstract way to interact with the data source, regardless of the specific implementation details.
- They promote loose coupling between the application and the data source by defining a set of methods that the data access layer must implement.
- DAO interfaces are usually implemented by DAO implementation classes.

**Step 5**: Create Dao Implementation java classes for all entitites

- DAO implementation classes are concrete classes that implement the DAO interfaces.
- They provide the actual implementation for the data access methods defined in the DAO interfaces.
- DAO implementation classes interact with the data source (e.g., database) to perform the requested operations.
- They contain the logic for executing database queries, handling transactions, and mapping data between the database and Java objects.
- The purpose of DAO implementation classes is to bridge the gap between the DAO interfaces and the underlying data source.
- They encapsulate the specific details of interacting with the database, such as SQL queries or ORM (Object-Relational Mapping) operations.
- DAO implementation classes handle the low-level database operations and provide the necessary data to the calling layers of the application.

Note: The main reasons for using DAO interfaces and DAO implementation classes are:

- Separation of Concerns: DAO interfaces separate the data access logic from the rest of the application. They define a clear boundary for data access operations and promote modular and maintainable code.
- Abstraction and Encapsulation: DAO interfaces provide an abstraction layer that allows the application to work with the data source through a consistent interface. DAO implementation classes encapsulate the specific details of data access, shielding the application from the underlying database technology.
- Flexibility and Interchangeability: By programming to the DAO interfaces, you can easily switch between different implementations of the DAO interfaces. For example, you could have different implementations for different databases or use a mock implementation for testing purposes.
- In summary, DAO interfaces define the contract for data access methods, while DAO implementation classes provide the concrete implementation of those methods. They facilitate structured data access, promote separation of concerns, and enable flexibility and maintainability in the data access layer of an application.

## Step 6 : Create Controller Servlet

The Controller Servlet is responsible for handling incoming requests from clients and directing them to the appropriate components of the application. It acts as the central point of control.

**Functionality**

- Receives HTTP requests from clients.
- Parses request parameters and data.
- Invokes the appropriate service methods based on the request.
- Sets request attributes and forwards the request to the corresponding JSP file for rendering the response.

## Step 7: create Model

The Model represents the data and business logic of the application. It encapsulates the state and behavior of the application's entities.

**Functionality**

- Defines the structure of the data entities.
- Provides methods to access, manipulate, and validate the data.

## Step 8: create Services Interface

The Services Interface defines the contract for the services that interact with the Model. It specifies the operations that can be performed on the data.

**Functionality**

- Declares the methods to interact with the Model.
- Provides an abstraction layer between the Controller and the Model.
- Allows for loose coupling and easy maintenance.

**Step 9: create Services Implementation**

The Services Implementation implements the Services Interface and provides the actual implementation of the service methods. It interacts with the Model to perform operations on the data.

**Functionality**

- Implements the methods defined in the Services Interface.
- Communicates with the Model to retrieve or manipulate data.
- Contains the application's business logic.

**Step 10: create JSP File**

The JSP (JavaServer Pages) file is responsible for rendering the response HTML. It receives data from the Controller Servlet and generates dynamic content to be sent back to the client.

**Functionality**

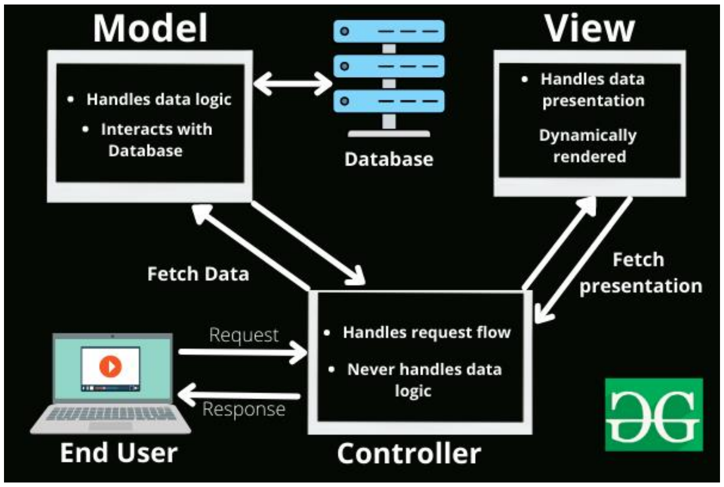
- Uses HTML and embedded Java code to generate dynamic content.
- Accesses request attributes set by the Controller Servlet to display data.

## MVC Architecture:

MVC (Model-View-Controller) is a software architectural pattern widely used in developing web applications. It separates the application logic into three interconnected components: the Model, the View, and the Controller. Each component has a specific role and responsibility within the application.

**Model**: The Model represents the application's data and business logic. It encapsulates the state and behavior of the application's entities, such as objects, data structures, and databases. The Model is responsible for managing the data, performing data operations (CRUD: Create, Read, Update, Delete), and enforcing business rules and validations. It is independent of the user interface and communicates with the View and Controller through interfaces or events.

**View**: The View is responsible for rendering the user interface (UI) and presenting the data to the user. It displays the information from the Model in a format that is understandable and visually appealing. The View can be a web page, a mobile app screen, or any other UI component. It receives data from the Controller or directly from the Model and generates the appropriate output for the user. In some cases, the View also captures user input and sends it to the Controller for further processing.



**Controller**: The Controller acts as an intermediary between the Model and the View. It receives user requests from the View, processes them, and updates the Model accordingly. It contains the application's logic for handling user actions, making decisions, and coordinating the flow of data. The Controller interprets the user input, invokes the appropriate methods in the Model to perform necessary operations, and selects the appropriate View to display the updated information to the user.

The flow of data and interactions in the MVC architecture typically follows this pattern:

- The user interacts with the View by triggering an event or action (e.g., clicking a button, submitting a form).
- The View notifies the Controller about the user's action.
- The Controller receives the notification, processes the request, and updates the Model as necessary.
- The Model notifies the View about the changes in the data.
- The View retrieves the updated data from the Model and refreshes the UI to display the new information.
- The user sees the updated View and can perform further actions, repeating the cycle.

The MVC architecture promotes separation of concerns, allowing developers to focus on specific aspects of the application without tightly coupling the different components. It enhances code reusability, maintainability, and testability by enabling independent development and modification of each component. Additionally, MVC provides a clear separation between the UI and the business logic, making it easier to support multiple UI platforms (web, mobile, desktop) that interact with the same underlying Model.