Technical Documentation and System Architecture for Airflow: A Flight Reservation Management Application

Development Team
Software Engineering Department
University/Organization Name
City, Country
email@domain.com

***Abstract*—This paper presents the comprehensive technical documentation of Airflow, a flight reservation management system. The system is designed to efficiently manage flight reservations, providing a robust platform that enables users to search, book, and manage their air travel. The application encompasses comprehensive flight management, from aircraft and seat administration to reservation and flight status control, delivering a complete experience for both end users and system administrators. This documentation details the database design methodology, entity-relationship modeling, system architecture, and implementation patterns used in the development process.**

***Index Terms*—flight reservation system, database design, entity-relationship model, software architecture, data access objects, ontology-based development**

I. Introduction

Modern air travel management requires sophisticated software systems capable of handling complex operations including flight scheduling, seat management, user authentication, and reservation processing. This paper presents the technical documentation for Airflow, a comprehensive flight reservation management application designed to address these requirements through a well-structured, scalable architecture.

The system integrates multiple components including user management, aircraft inventory, flight scheduling, and reservation processing. The application serves both end users seeking to book flights and administrators requiring comprehensive system management capabilities.

II. Data Persistence - Database Design

A. Development Methodology

The development of the entity-relationship model is based on ontologies as a conceptual foundation. This methodology enables the creation of a more coherent and semantically rich database design, ensuring that relationships between entities accurately reflect the business domain logic.

The development process followed these stages:

1) **Entity Declaration**: Identification and definition of the system's main entities
2) **Attribute Definition**: Specification of each entity's properties with their respective data types
3) **Relationship Establishment**: Development of connections between entities through cardinality analysis
4) **Model Validation**: Verification of design integrity and consistency

B. System Entities

The system comprises the following main entities:

1) Users: Represents both administrators and regular customers of the system. This entity centralizes user management with different privilege levels.

Attributes:

- `id_PK`: Unique user identifier (int, primary key)
- `name`: User's first name (varchar(40), not null)
- `last_name`: User's last name (varchar(40), not null)
- `email`: Email address (varchar(200), not null)
- `password`: Encrypted password (varchar(20), not null)
- `isSuperUser`: Administrative privileges indicator (boolean, not null)
- `created_at`: Account creation timestamp (datetime, not null)

2) Airplanes: Manages information about aircraft available in the fleet.

Attributes:

- `id_PK`: Unique aircraft identifier (int, primary key)
- `airline`: Airline owner (varchar(20), not null)
- `model`: Aircraft model (varchar(50), not null)
- `code`: Aircraft identification code (varchar(10), not null)
- `capacity`: Total passenger capacity (int, not null)
- `year`: Manufacturing year (year)

3) Cities: Represents cities that are part of the flight network, used to implement route graph logic.

Attributes:

- `id_PK`: Unique city identifier (int, primary key)
- `name`: City name (varchar(100), not null)
- `country`: Country where the city is located (varchar(100), not null)
- `code`: Airport code of the city (varchar(10), not null)

4) Flights: Manages information about scheduled flights in the system.

Attributes:

- `id_PK`: Unique flight identifier (int, primary key)
- `airplane_FK`: Reference to assigned aircraft (int, foreign key)
- `status_FK`: Current flight status (int, foreign key)
- `origin_city_FK`: Origin city (int, foreign key)
- `destination_city_FK`: Destination city (int, foreign key)
- `code`: Flight identification code (varchar(10), not null)
- `departure_time`: Departure date and time (timestamp, not null)
- `arrival_time`: Arrival date and time (timestamp, not null)
- `price_base`: Base flight price (decimal(10,2), not null)

5) Reservations: Manages reservations made by users.

Attributes:

- `id_PK`: Unique reservation identifier (int, primary key)
- `user_FK`: User who made the reservation (int, foreign key)
- `status_FK`: Current reservation status (int, foreign key)
- `flight_FK`: Reserved flight (int, foreign key)
- `reserved_at`: Reservation timestamp (timestamp, not null)

6) Seats: Manages available seats on aircraft.

Attributes:

- `id_PK`: Unique seat identifier (int, primary key)
- `airplane_FK`: Aircraft to which it belongs (int, foreign key)
- `reservation_FK`: Associated reservation, if exists (int, foreign key, nullable)
- `seat_number`: Seat number (varchar(10), not null)
- `seat_class`: Seat class (enum: ECONOMY, BUSINESS, FIRST)
- `is_window`: Indicates if it's a window seat (boolean)

7) Flight_Status: Catalog of possible states for flights.

Attributes:

- `id_PK`: Unique status identifier (int, primary key)
- `name`: Status name (varchar(15), not null)
- `description`: Detailed status description (varchar(100))

8) Reservations_Status: Catalog of possible states for reservations.

Attributes:

- `id_PK`: Unique status identifier (int, primary key)
- `name`: Status name (varchar(15), not null)
- `description`: Detailed status description (varchar(100))

C. Entity Relationships

The system establishes the following relationships between identified entities:

1) User-Reservation Relationships: **Cardinality**: 1:M (One to Many)

- A user can make zero or multiple flight reservations
- Each reservation is assigned to only one user
- This relationship ensures reservation traceability

2) Aircraft-Reservation Relationships: **Cardinality**: 1:M (One to Many)

- An aircraft can be assigned to multiple reservations
- Each reservation can have only one aircraft assigned
- A reservation may initially have no aircraft assigned

3) Aircraft-Seat Relationships: **Cardinality**: 1:M (One to Many)

- An aircraft is composed of multiple seats
- A seat belongs exclusively to a specific aircraft
- This relationship defines the physical configuration of each aircraft

4) Aircraft-Flight Relationships: **Cardinality**: 1:M (One to Many)

- An aircraft may or may not be assigned to an active flight
- A flight must have an aircraft assigned
- This relationship manages flight operations

5) Reservation-Seat Relationships: **Cardinality**: 1:M (One to Many)

- A reservation can include one or multiple seats
- A seat may or may not be assigned to a reservation
- This relationship enables group reservations

6) Reservation-Flight Relationships: **Cardinality**: M:1 (Many to One)

- A reservation is assigned to a specific flight
- A flight can have multiple registered reservations
- This relationship links reservations with scheduled flights

7) Status Relationships: **Cardinality**: 1:M (One to Many)

- A reservation has a unique status at any given time
- A status can be assigned to multiple reservations
- A flight has a specific status
- A flight status can apply to multiple flights

8) City-Flight Relationships: **Cardinality**: 1:M (One to Many)

- A flight must depart from an origin city
- A flight must land at a destination city
- A city can be the origin of multiple flights
- A city can be the destination of multiple flights
- This structure enables implementation of routing algorithms and graphs

D. Design Considerations

1) Referential Integrity: The design implements referential integrity constraints through foreign keys, ensuring data consistency and preventing references to non-existent entities.

2) Normalization: The model is in third normal form (3NF), eliminating redundancies and transitive dependencies, which optimizes storage and reduces update anomalies.

3) Scalability: The structure allows horizontal growth through implementation of appropriate indexes and partitioning of large tables such as `reservations` and `flights`.

4) Flexibility: The use of status tables (`flight_status` and `reservations_status`) provides flexibility to add new states without modifying the structure of main tables.

III. Data Access Object (DAO) Pattern

The system implements the DAO design pattern to abstract and encapsulate database access, clearly separating business logic from persistence logic. The implementation follows these principles:

A. DAOMethods Interface

A generic interface `DAOMethods<T>` is defined that establishes standard methods for CRUD operations:

- `ArrayList<T> getAll()`: Retrieves all objects of type T
- `T getById(int id)`: Retrieves a specific object by its identifier
- `void create(T object)`: Creates a new record in the database
- `void update(int id, T toUpdate)`: Updates an existing record
- `void delete(int id)`: Deletes a specific record

B. Specific Implementations

Each domain entity has its own DAO implementation that extends the generic interface:

- `FlightDAO`: Manages CRUD operations for flights
- `ReservationDAO`: Manages CRUD operations for reservations
- `UsersDAO`: Manages CRUD operations for users
- `AirplaneDAO`: Manages CRUD operations for aircraft
- `CityDAO`: Manages CRUD operations for cities
- `SeatDAO`: Manages CRUD operations for seats

This structure ensures uniformity in data access and facilitates code maintainability by centralizing persistence logic.

C. SQL Query Methods

DAOs implement various SQL queries to satisfy system requirements. Below are some relevant examples:

1) FlightDAO Queries: The FlightDAO class implements complex queries including flight searches with status information and multi-criteria filtering based on origin, destination, and date parameters.

2) ReservationDAO Queries: The ReservationDAO handles user-specific reservation retrieval and seat availability verification through join operations between multiple tables.

D. Database Connection Management

Database connection management is centralized in the `ConnectionDB` class, following the Singleton pattern to optimize resources. The connection configuration includes Unicode support, SSL configuration, and comprehensive exception handling.

1) Transaction Handling: The system implements transaction control for critical operations such as flight reservations, ensuring data integrity even in complex operations affecting multiple tables through commit/rollback mechanisms.

IV. System Architecture

A. General Description

The flight reservation application architecture is based on a layered structure that clearly separates the different system responsibilities. The main architectural layers are:

- **Presentation Layer**: User interface and interaction
- **Model Layer**: Data representation in the application
- **Persistence Layer**: Data access and management in the database
- **Service Layer**: Business logic
- **Utility Layer**: Tools for different layers

This layered architecture promotes separation of concerns, maintainability, and scalability while ensuring clear boundaries between different system components.

V. User Interface Design

A. Design Philosophy

The graphical user interface has been designed following principles of minimalism and simplicity, with the aim of providing an intuitive and modern user experience. The key elements that have guided the design are:

- **Simplicity**: Elimination of unnecessary elements to focus on essential functionality
- **Consistency**: Coherent use of visual elements and interaction patterns throughout the application
- **Accessibility**: Clear and intuitive interface that facilitates navigation for all users
- **Modern aesthetics**: Implementation of a contemporary design using the FlatLaf library

B. Technologies Used

- **Java Swing**: Base framework for building graphical components
- **FlatLaf**: External library that provides a modern and flat look to Swing components
- **Responsive Design**: Implementation of layouts that adapt to different screen sizes

C. Interface Structure

The user interface is organized following a hierarchical component architecture:

1) **MainFrame**: Main window that contains all other components
2) **Horizontal Menu**: Top bar with logo, title and navigation options
3) **Content Panel**: Central area with card system to display different sections
4) **Specific Panels**: Components dedicated to specific functionalities

D. Main Components

1) Flight Search Panel: Interface that allows users to search for available flights according to criteria such as origin, destination, and dates. Includes:

- Origin and destination city selectors
- Travel date selector
- Additional filters
- Search button

2) Flight Details Panel: Displays detailed information about a selected flight, including:

- Airline information and flight number
- Departure and arrival times
- Airport details
- Seat selection

3) Confirmation Panel: Allows the user to review and confirm their booking details before finalizing:

- Flight information summary
- Passenger details
- Payment information
- Confirmation or cancellation buttons

4) Dialog Components: The application implements specific dialog components to handle user authentication:

- **Login Dialog**: Provides a secure interface for users to authenticate and access their accounts
- **Register Dialog**: Allows new users to create accounts by entering their personal information

These dialog components follow the same minimalist design principles as the main interface, maintaining visual consistency throughout the application while serving their specialized functions.

E. Visual Design Elements

- **Color Palette**: Predominance of light tones with subtle accents to highlight important elements
- **Typography**: Use of sans-serif fonts to improve readability
- **Iconography**: Implementation of minimalist icons to represent common actions
- **Spacing**: Design with sufficient white space to reduce visual overload

F. User Interaction and Flow

The navigation flow in the application follows a linear and predictable pattern:

1) Search for available flights
2) Selection and display of flight details
3) Selection of seats and additional options
4) Review and confirmation of the booking
5) Display of confirmation and booking details

This sequential design guides the user through the booking process intuitively, reducing the learning curve and minimizing possible errors during the process.

G. Interface Screenshots

Below are screenshots of the application's main interfaces, demonstrating the implemented design principles and user flows:

docs/images/main_interface.png

Fig. 1. Main application interface showing the flight search panel

Fig. 2. Flight details panel showing selected flight information

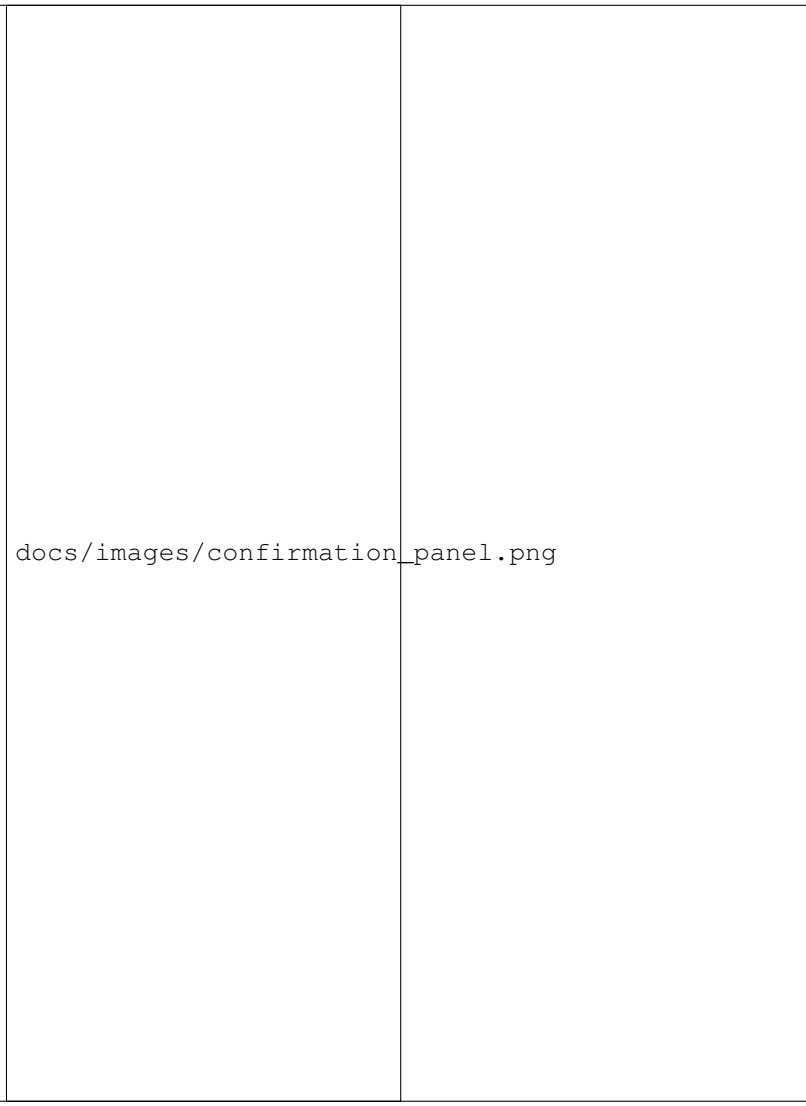

Fig. 3. Booking confirmation panel with reservation summary

These screenshots demonstrate how the design principles of simplicity, consistency, and modern aesthetics have been applied throughout the application, resulting in a clean and intuitive user interface.

REFERENCES

[1] Date, C.J., "An Introduction to Database Systems," 8th ed. Boston: Addison-Wesley, 2004.

[2] Fowler, M., "Patterns of Enterprise Application Architecture," Boston: Addison-Wesley, 2002.

[3] Silberschatz, A., Galvin, P.B., and Gagne, G., "Database System Concepts," 7th ed. New York: McGraw-Hill, 2019.