Travel Agency Reservation Management (JavaFX)

Reda Mdair

2025

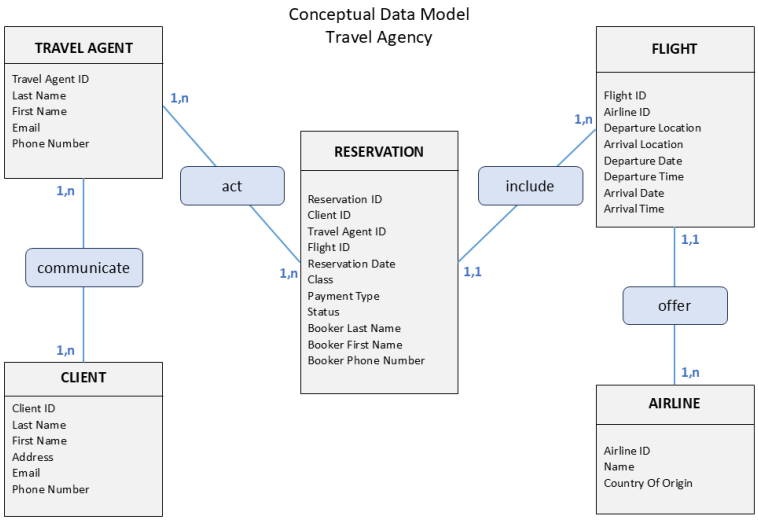
1. Introduction

This project aims to develop a Java application for managing travel agency reservations. The objective is to replace the use of Excel files with a modern solution based on a database, providing an intuitive interface for users.

Reservation details include the beneficiary client, the associated travel agent, the flight itinerary, and the airline operating the flight. Based on this, we have defined a conceptual data model (CDM) with five entities:

- Client (Client)
- Travel Agent (Travel Agent)
- Reservation (Reservation)
- Flight (Flight)
- Airline (Airline)

These entities are linked through primary and foreign key relationships. For example, a reservation is associated with a client, a travel agent, and a flight. The model can be represented as follows:



Reservations constitute the core unit of our project and are connected to other entities via foreign keys. Currently, it is assumed that the travel agency records all reservation-related information in a single row within an Excel file. While intuitive, this method is far from optimised: increased risk of human errors (typos or inconsistencies), lack of automation (absence of relationships between entities), etc.
Our solution assigns a unique identifier to each entity, enabling efficient referencing of information across different tables. Thus, for a given reservation, it will suffice to insert the identifiers of the relevant entities to retrieve all associated attributes, thanks to the integrity of relationships defined in the database.

The project follows a modular structure, making it easier to navigate and understand its various components. Below is the directory tree of its source data:

```
docker/                    contains the Docker container configuration
├── docker-compose.yml
└── initialization.sql   initialises the database directly from Docker Compose
src/                       main directory containing Java source files
├── module-info.java
├── config/                class managing the database connection
│   ├── DatabaseConnection.java
│   └── test.java
├── models/                defines the main entities of the project
│   ├── Client.java
│   ├── TravelAgent.java
│   ├── Airline.java
│   ├── Flight.java
│   └── Reservation.java
├── dao/                   contains data access objects for database interaction
│   ├── ClientDAO.java
│   ├── TravelAgentDAO.java
│   ├── AirlineDAO.java
│   ├── FlightDAO.java
│   └── ReservationDAO.java
├── fx/                    classes responsible for managing the user interfaces
│   ├── ClientFX.java
│   ├── TravelAgentFX.java
│   ├── AirlineFX.java
│   ├── FlightFX.java
│   ├── ReservationFX.java
│   ├── ExcelViewFX.java
│   └── Main.java.         main graphical interface class
└── executable.jar     file enabling access to the interface from a terminal
```

The `docker-compose.yml` file configures a MySQL container for the Java project. Its main details are as follows:

```yaml
environment:
  MYSQL_ROOT_PASSWORD: password
  MYSQL_DATABASE: project_travel_agency
  MYSQL_USER: project_travel_agency
  MYSQL_PASSWORD: password
ports:
  - "3307:3306"
volumes:
  - dbdata:/var/lib/mysql
  - ./initialization.sql:/docker-entrypoint-initdb.d/init.sql
```

This configuration simplifies the setup of a MySQL database pre-initialised with the necessary project data. The command `docker-compose up -d` starts the container with a pre-populated database (via the `initialization.sql` script located in the same directory).

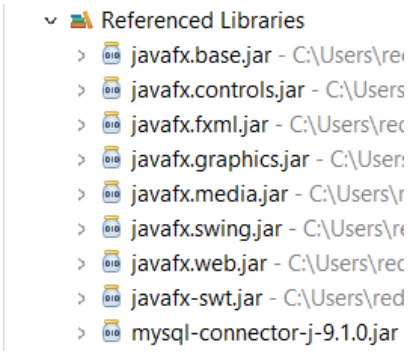
2. Project Management via Eclipse

The `src` directory contains all the project's source files, organised into different folders to improve code readability. This directory was created using the `Eclipse IDE`, which allowed for efficient project structuring and progressive testing of features. The fundamental files are as follows:

- `DatabaseConnection.java`: Responsible for establishing and managing connections with the MySQL database.
- `Main.java`: The application's entry point, which launches the main interface after the database is initialised.

Installing JavaFX

For the graphical interface, we downloaded the JavaFX library, available at this link. After installation, the associated JavaFX SDK libraries and a MySQL connector (download link) were added to the referenced libraries in Eclipse.



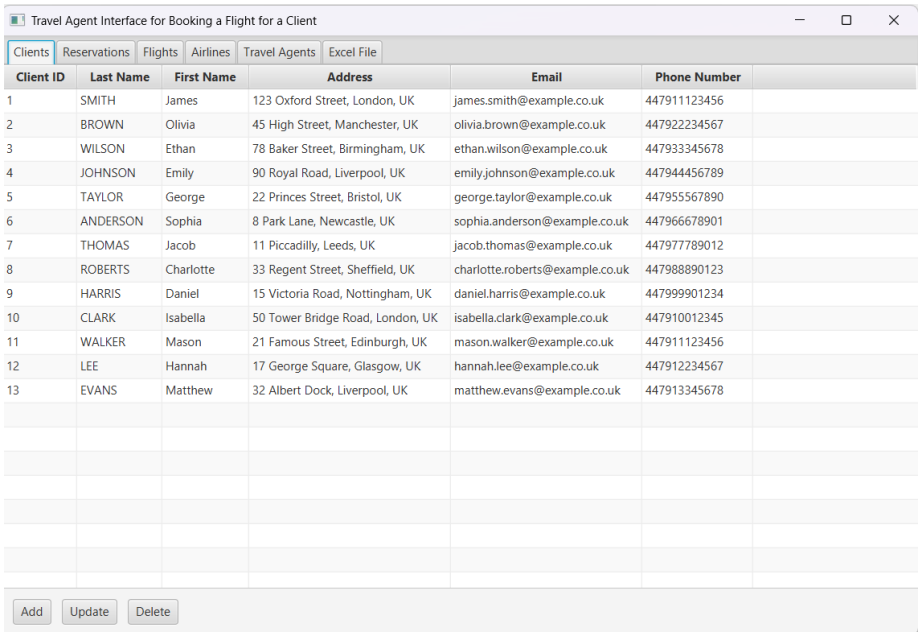
Organisation

The different packages work together seamlessly to produce a functional and modular application.

- The `models` package contains the main entities (e.g. `Client`, `Flight`, `Reservation`), which define the structure of the data handled by the application.
- The `dao` package implements CRUD operations (Create, Read, Update, Delete) for each entity. The DAO classes (`ClientDAO`, `FlightDAO`, etc.) interact directly with the instances of the `models` package classes, using their attributes and methods to convert retrieved database data into Java objects.
- The `fx` package groups the classes responsible for managing the user interfaces. These interfaces use the DAO class methods to display and modify data in real-time.

The workflow begins with user interaction in the graphical interface (e.g. adding a reservation), triggering a method in the `fx` package. This method interacts with the DAO classes, which transmit queries to the database via `DatabaseConnection`. Once the database is updated, the information is returned and displayed in the interface.

The graphical interface simulates the personal workspace of a travel agent. Upon a client request, the agent enters the client's details into the `Client` table and then adds a reservation in their name using available flights from the `Flight` table.
The interface opens as follows:



Client ID	Last Name	First Name	Address	Email	Phone Number
1	SMITH	James	123 Oxford Street, London, UK	james.smith@example.co.uk	447911123456
2	BROWN	Olivia	45 High Street, Manchester, UK	olivia.brown@example.co.uk	447922234567
3	WILSON	Ethan	78 Baker Street, Birmingham, UK	ethan.wilson@example.co.uk	447933345678
4	JOHNSON	Emily	90 Royal Road, Liverpool, UK	emily.johnson@example.co.uk	447944456789
5	TAYLOR	George	22 Princes Street, Bristol, UK	george.taylor@example.co.uk	447955567890
6	ANDERSON	Sophia	8 Park Lane, Newcastle, UK	sophia.anderson@example.co.uk	447966678901
7	THOMAS	Jacob	11 Piccadilly, Leeds, UK	jacob.thomas@example.co.uk	447977789012
8	ROBERTS	Charlotte	33 Regent Street, Sheffield, UK	charlotte.roberts@example.co.uk	447988890123
9	HARRIS	Daniel	15 Victoria Road, Nottingham, UK	daniel.harris@example.co.uk	447999901234
10	CLARK	Isabella	50 Tower Bridge Road, London, UK	isabella.clark@example.co.uk	447910012345
11	WALKER	Mason	21 Famous Street, Edinburgh, UK	mason.walker@example.co.uk	447911123456
12	LEE	Hannah	17 George Square, Glasgow, UK	hannah.lee@example.co.uk	447912234567
13	EVANS	Matthew	32 Albert Dock, Liverpool, UK	matthew.evans@example.co.uk	447913345678

Figure 1: Interface home screen

It allows for the display of all five project entities, as well as an Excel file summarising all reservation details in a single row.
Three buttons are available:

- `Add`: always available.
- `Edit`: requires prior selection of a row.
- `Delete`: requires prior selection of a row.
  Note: To delete a client, all their recorded reservations must first be deleted.

Travel agents interact only with client and reservation information. Some tables, containing static or administration-defined data, are read-only. Agents can only view them:

- **Flight**: Flight details (locations, schedules), defined by airlines.
- **Airline**: Information about airlines.
- **Travel Agent**: Agent details, managed by the agency's administration.

No buttons are available for these tables in the graphical interface. This distinction ensures optimal use of the application. Agents focus on managing clients and reservations, while sensitive or administrative data, such as flights and airlines, remain protected from accidental or unauthorized modifications.

Here is the interface displayed when modifying a reservation. A pre-filled form appears, allowing the user to edit the desired information.

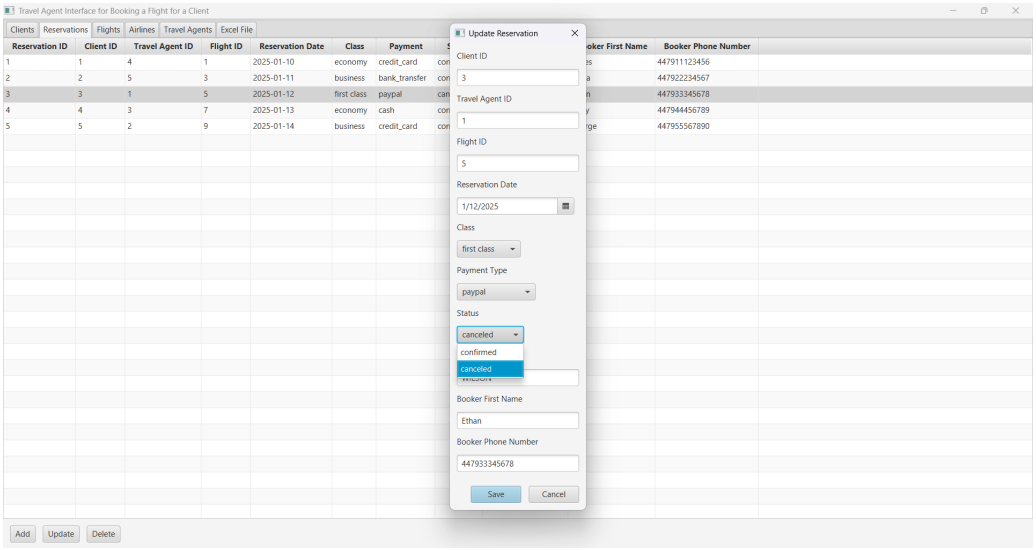


Figure 2: Modifying a reservation

A travel agent particularly uses this function when they need to cancel a client's reservation by changing its status from "confirmed" to "cancelled".
Once a reservation is added or modified, it is instantly updated in the Excel tab, which is structured as follows:

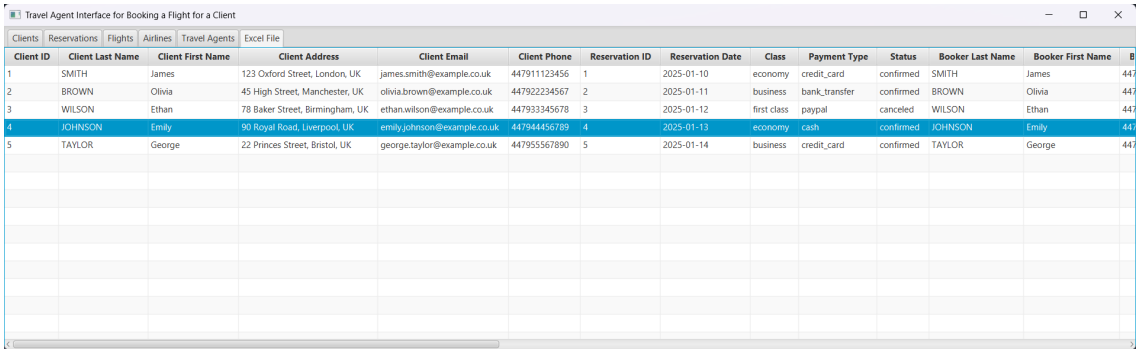


Figure 3: Excel tab (scroll to the right for additional columns)

This tab provides an overview of all entities related to a reservation (client, flight, travel agent, etc.), consolidated into a single row. It is generated using the `ExcelViewFX` class, which utilises joins between different database entities to produce a comprehensive view.

3. Project Management via Command Line

In addition to the graphical interface, the application also allows direct management of the MySQL database via a command-line terminal. This feature provides additional flexibility for administrators or developers who wish to execute manual SQL commands or automate tasks.

Accessing the Database

The database is hosted within a Docker container, simplifying its deployment and access. The steps we followed to interact with it are as follows:

1. Start the Docker container by executing the following command in the terminal:

   ```
   docker-compose up -d
   ```

2. Access the MySQL database within the container using one of the following commands (choose based on the required access rights):

   ```
   docker exec -it project_travel_agency mysql -u project_travel_agency -p

   docker exec -it project_travel_agency mysql -u root -p
   ```

3. Enter the configured password (specified as `password`) when prompted by the terminal.

Executing SQL Commands

Once connected, it is possible to execute SQL commands to interact with the data:

For example, to display the list of flights from the `Flight` table, use the following command:

SELECT * FROM Flight;

```
mysql> SELECT * FROM Flight;
+----------+-----------+--------------------+-----------------+---------------+---------------+-------------+-------------+
| FlightID | AirlineID | DepartureLocation | ArrivalLocation | DepartureDate | DepartureTime | ArrivalDate | ArrivalTime |
+----------+-----------+--------------------+-----------------+---------------+---------------+-------------+-------------+
|        1 |         1 | Dubai              | Paris           | 2025-07-20    | 14:00:00      | 2025-07-20  | 18:00:00    |
|        2 |         2 | Doha               | London          | 2025-07-21    | 15:30:00      | 2025-07-21  | 19:30:00    |
|        3 |         3 | Zurich             | Charleroi       | 2025-07-22    | 11:00:00      | 2025-07-22  | 12:40:00    |
|        4 |         4 | Casablanca         | Bern            | 2025-07-25    | 10:00:00      | 2025-07-25  | 12:55:00    |
|        5 |         5 | Paris              | Tokyo           | 2025-07-25    | 22:30:00      | 2025-07-26  | 17:00:00    |
|        6 |         6 | London             | Los Angeles     | 2025-07-26    | 21:00:00      | 2025-07-27  | 05:30:00    |
|        7 |         7 | Frankfurt          | Tokyo           | 2025-07-28    | 13:00:00      | 2025-07-29  | 01:00:00    |
|        8 |         8 | Atlanta            | Toronto         | 2025-08-01    | 09:00:00      | 2025-08-01  | 10:30:00    |
|        9 |         9 | Dublin             | Brussels        | 2025-08-02    | 08:00:00      | 2025-08-02  | 09:45:00    |
|       10 |        10 | Istanbul           | Rome            | 2025-08-03    | 12:00:00      | 2025-08-03  | 13:30:00    |
|       11 |        11 | Hong Kong          | San Francisco   | 2025-08-03    | 16:00:00      | 2025-08-04  | 08:00:00    |
|       12 |        12 | Tokyo              | Los Angeles     | 2025-08-06    | 19:00:00      | 2025-08-07  | 11:00:00    |
|       13 |        13 | Sydney             | Auckland        | 2025-09-01    | 09:00:00      | 2025-09-01  | 10:30:00    |
|       14 |        14 | Kuala Lumpur       | Bali            | 2025-09-02    | 11:00:00      | 2025-09-02  | 12:00:00    |
|       15 |        15 | Amsterdam          | Bordeaux        | 2025-09-03    | 16:30:00      | 2025-09-03  | 18:30:00    |
|       16 |        16 | Madrid             | Lisbon          | 2025-09-04    | 10:00:00      | 2025-09-04  | 11:00:00    |
|       17 |        17 | Vienna             | Warsaw          | 2025-10-05    | 14:00:00      | 2025-10-05  | 15:30:00    |
|       18 |        18 | New Delhi          | Mumbai          | 2025-10-06    | 09:00:00      | 2025-10-06  | 10:30:00    |
|       19 |        19 | Johannesburg       | Cape Town       | 2025-10-07    | 11:00:00      | 2025-10-07  | 12:00:00    |
|       20 |        20 | Singapore          | New York        | 2025-10-22    | 23:30:00      | 2025-10-23  | 17:00:00    |
+----------+-----------+--------------------+-----------------+---------------+---------------+-------------+-------------+
20 rows in set (0.00 sec)
```

This table is generated using foreign keys that reference the `Airline` table, which can be displayed using the following command:

SELECT * FROM Airline;

```
mysql> SELECT * FROM Airline;
+-----------+------------------------------+----------------------+
| AirlineID | Name                         | CountryOfOrigin      |
+-----------+------------------------------+----------------------+
|         1 | Emirates                     | United Arab Emirates |
|         2 | Qatar Airways                | Qatar                |
|         3 | Swiss International AirLines | Switzerland          |
|         4 | Royal Air Maroc              | Morocco              |
|         5 | Air France                   | France               |
|         6 | British Airways              | United Kingdom       |
|         7 | Lufthansa                    | Germany              |
|         8 | Delta Airlines               | United States        |
|         9 | Ryanair                      | Ireland              |
|        10 | Turkish Airlines             | Turkey               |
|        11 | Cathay Pacific               | Hong Kong            |
|        12 | All Nippon Airways           | Japan                |
|        13 | Qantas                       | Australia            |
|        14 | Malaysia Airlines            | Malaysia             |
|        15 | KLM                          | Netherlands          |
|        16 | Iberia                       | Spain                |
|        17 | Austrian Airlines            | Austria              |
|        18 | IndiGo                       | India                |
|        19 | South African Airways        | South Africa         |
|        20 | Singapore Airlines           | Singapore            |
+-----------+------------------------------+----------------------+
20 rows in set (0.00 sec)
```

Accessing the Graphical Interface

The JavaFX interface of the project can also be accessed from a command-line terminal. To achieve this, we exported the project from the Eclipse IDE using the `Runnable JAR File` option, selecting the library management option `Package required libraries into generated JAR`. This ensures that all dependencies, including JavaFX and MySQL connectors, are included in the JAR file.
The generated `executable.jar` file can be launched with the following command:

```
java -p "javafx-sdk-23.0.1/lib" --add-modules
javafx.controls,javafx.base,javafx.fxml,javafx.graphics,javafx.media,javafx.web
--add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED --add-exports
javafx.base/com.sun.javafx.event=ALL-UNNAMED -jar executable.jar
```

Command-Line Specificities

Access via the command line provides great flexibility for performing advanced operations that are not supported by the graphical interface. However, improper use can lead to errors or data loss if malformed SQL commands are executed.
This access is granted exclusively to administrators, while travel agents can only interact with the graphical interface. This method is essential for quickly diagnosing and resolving data-related issues or testing modifications to the database structure.

Furthermore, all tasks related to database privileges and access rights, such as creating new users, managing permissions, and handling tables that cannot be modified via the interface, must be performed exclusively through the terminal. This ensures secure and centralised control over authorisations, which is crucial for system robustness.