Project Report

SOEN 6441 (W)

Team Members:
Peter Sakr - 40237311
Abhishek Amola - 40105405

Table of Contents

Project Description

The project we implemented uses data from the airlabs.co api for tracking live flights and airlines. We designed a query system that keeps track of airlines and flights to allow users to query the data based on airline, flight and airport information.

Project Deliverables

For the design and development of the project, we submitted the following:

- Source code (available on GitHub at https://github.com/abhiamola/SOEN-6441-Project)
- Requirements Document (available on GitHub): Describes the requirements we followed while designing and implementing the project.
- Software Architecture Document (available on GitHub): Describes the architecture as well as the models we designed and used to create the project.
- Video Demo (submitted with this document)
- This Report

Used Software, Libraries and Frameworks

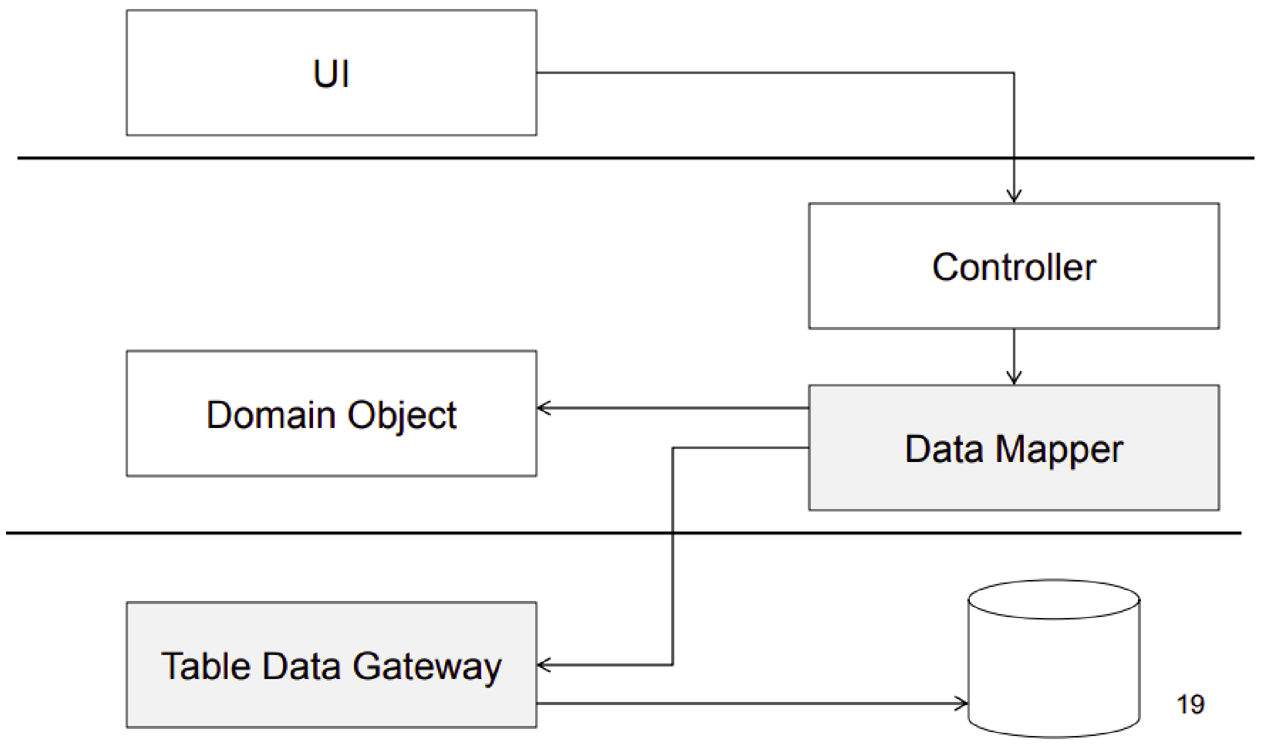
We used the following while developing the project:

- MySQL Server: to act as the local database and store the relations.
- NodeJS: used to create the backend server as well as run the frontend code.
- Express.js: allows creating API endpoints for the backend server that can be reached using HTTP requests.
- ReactJS: used to create the frontend user interface.
- axios: library used to send HTTP requests and receive the responses in the frontend.
- Mocha: testing framework used to create the tests as well as run them.
- Should.js: assertion framework used in test cases.

Implemented Patterns

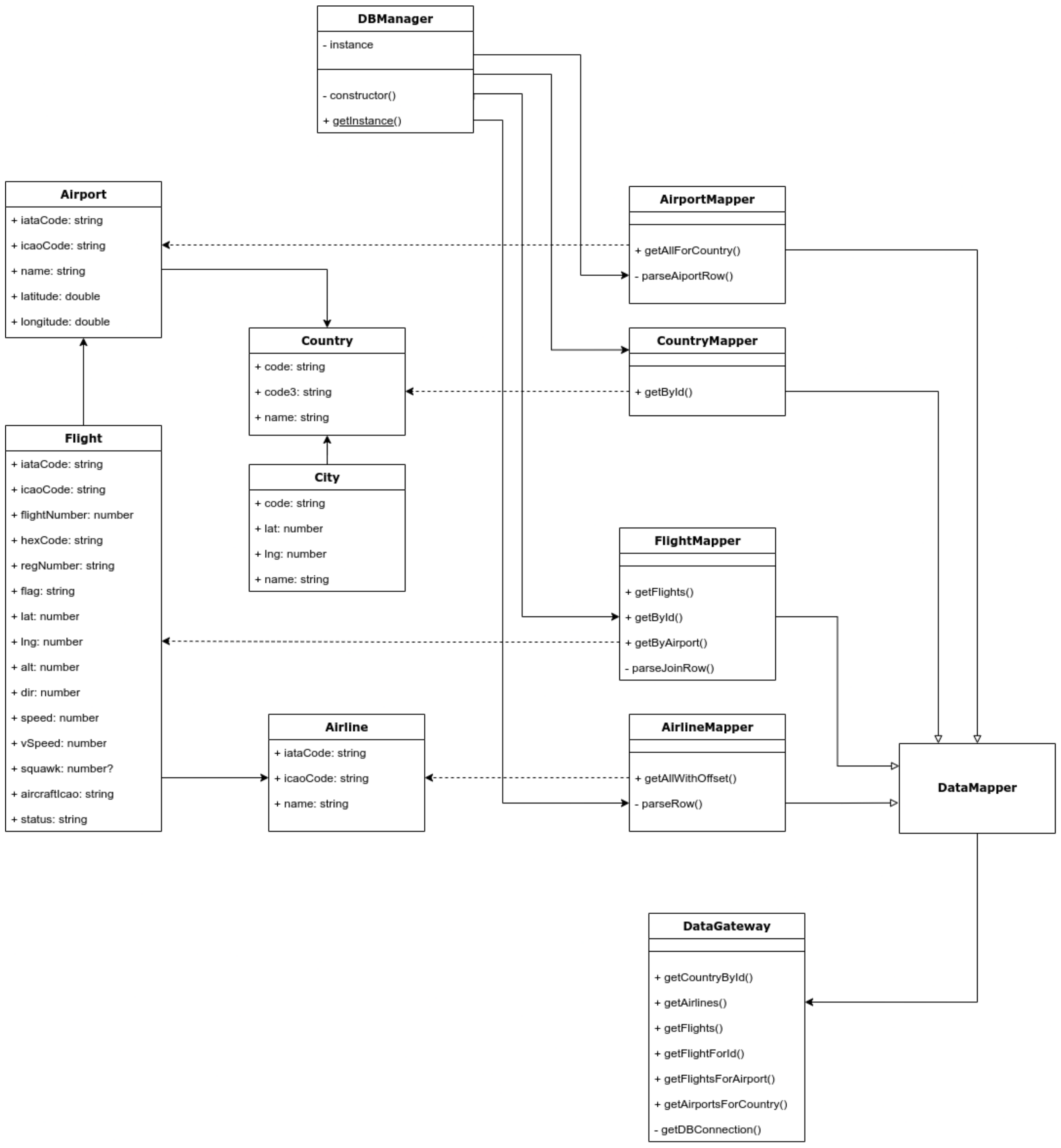
We implemented the **Client-Server architecture** by developing both a frontend and a backend server application. They communicate using HTTP requests that carry queries as well as response data that is shown to the user.

In our backend server we implemented both the **Table Data Gateway**, as well as the **Data Mapper** patterns:



To do so, we created a class called DataGateway (which acts as the data gateway) which is the only class in the backend that interacts with the database (performs queries). In addition, each domain object class has a respective data mapper which calls the table data gateway's methods and maps the returned raw rows to our domain objects. The controller in this diagram represents our backend's routes and endpoints. The user uses the UI to send requests to the controller, which calls and returns the respective functions from the correct data mapper.

The following class diagram shows the implemented classes as well as their methods.

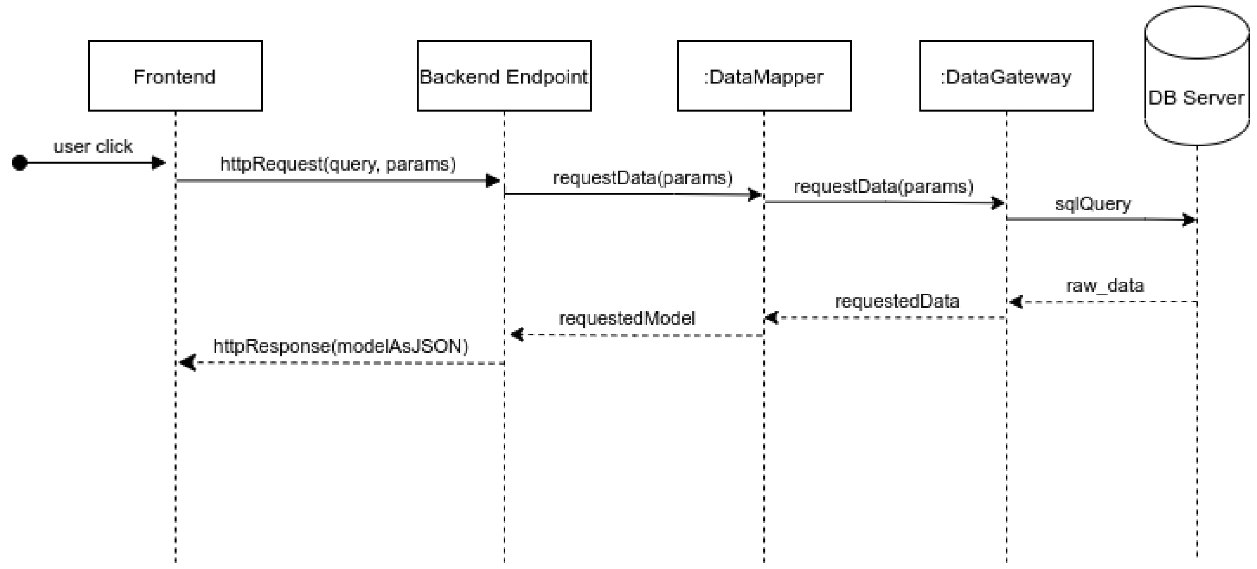


To make sure that there is only a single connection to the database active at all times, we opted to make the DBManager class a **Singleton** class. This allows us to make sure that this class will only be instantiated once and give access to the endpoints to send messages to the different mappers.

Implemented Design

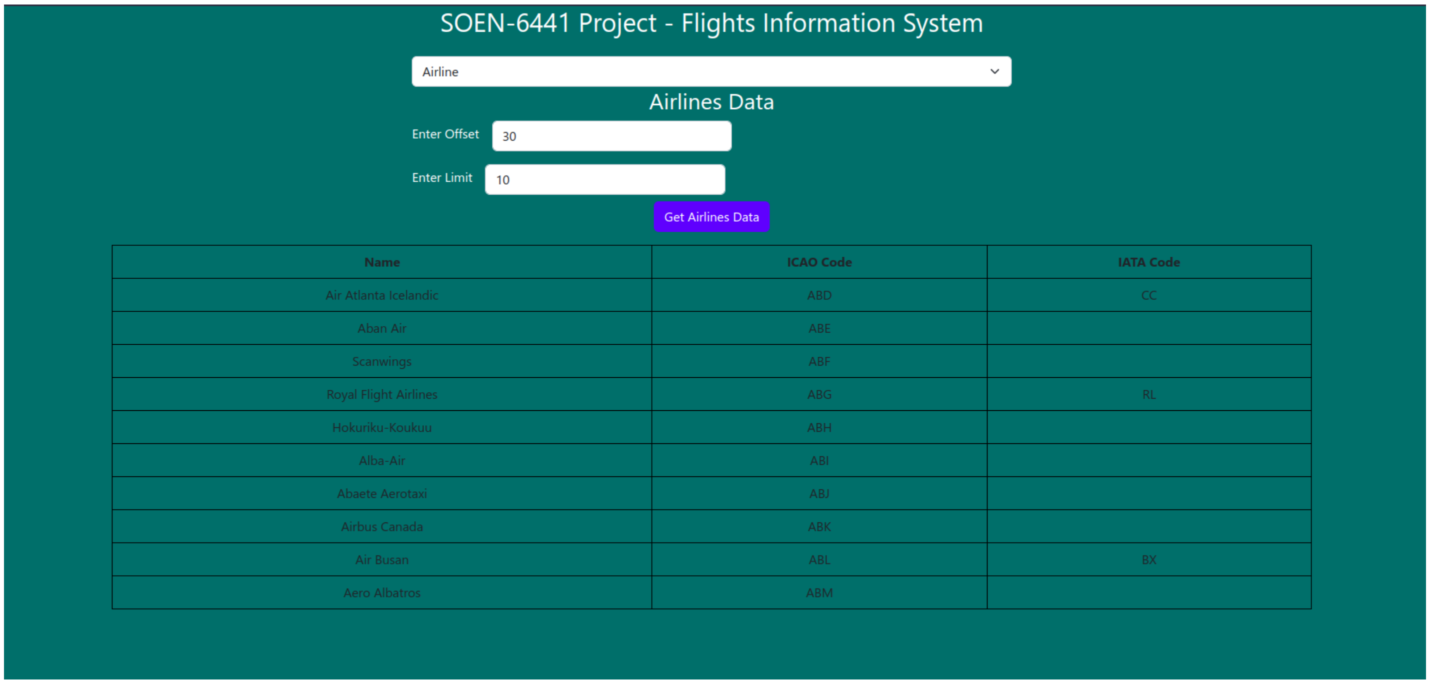
Sequence Diagram

Through our class diagram, and overall architecture of the backend system, the following diagram illustrates how a user would interact with the project:



Frontend

We developed the frontend using ReactJS for the interface and axios to handle the HTTP requests to the backend. The following screenshots illustrate the functionality of the frontend.


SOEN-6441 Project - Flights Information System

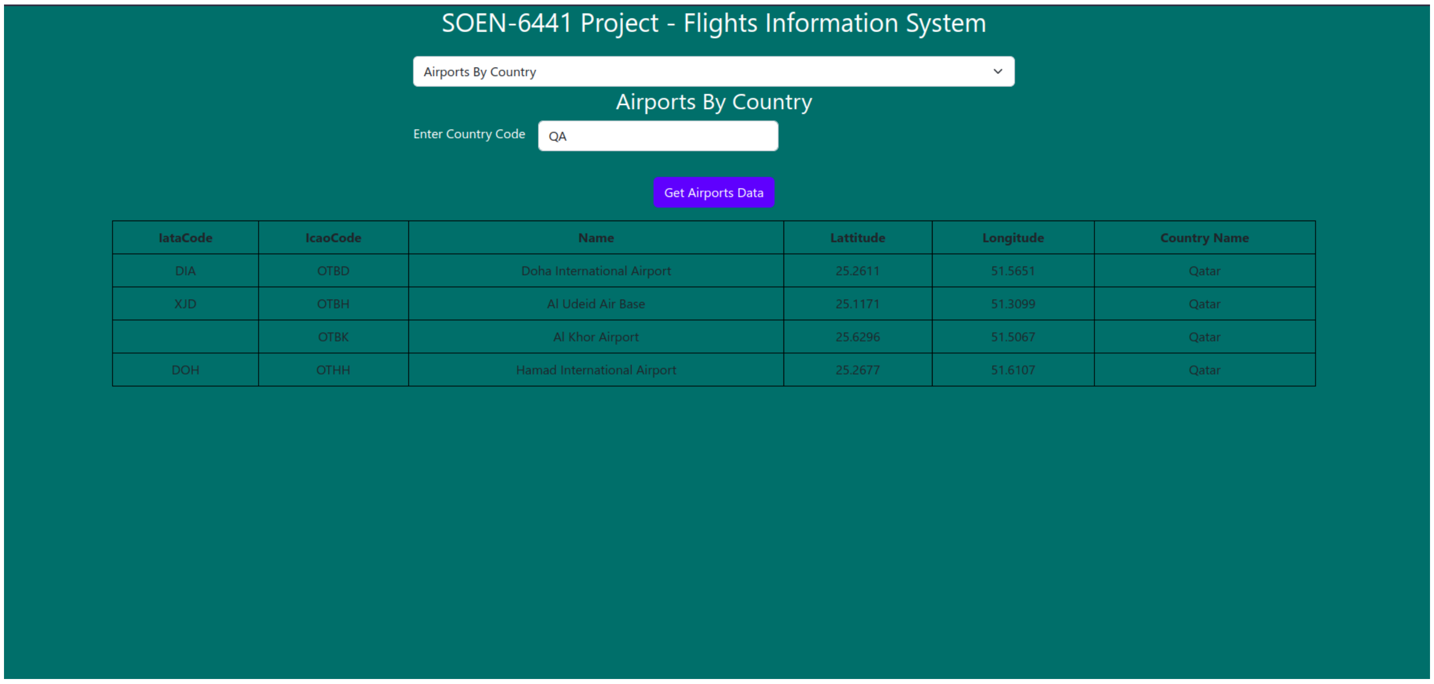
Airline

Airlines Data

Enter Offset: 30

Enter Limit: 10

Name	ICAO Code	IATA Code
Air Atlanta Icelandic	ABD	CC
Aban Air	ABE	
Scanwings	ABF	
Royal Flight Airlines	ABG	RL
Hokuriku-Koukuu	ABH	
Alba-Air	ABI	
Abaete Aerotaxi	ABJ	
Airbus Canada	ABK	
Air Busan	ABL	BX
Aero Albatros	ABM	

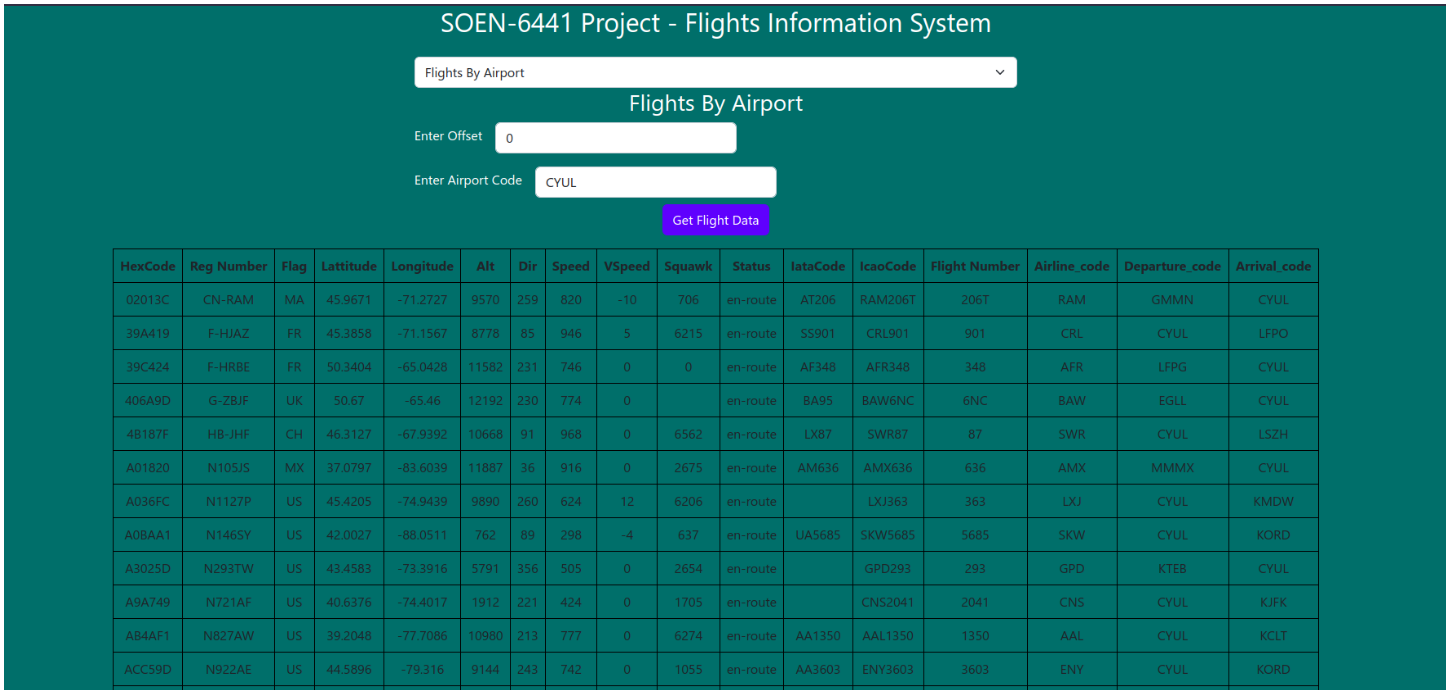

SOEN-6441 Project - Flights Information System

Airports By Country

Airports By Country

Enter Country Code: QA

IataCode	IcaoCode	Name	Lattitude	Longitude	Country Name
DIA	OTBD	Doha International Airport	25.2611	51.5651	Qatar
XJD	OTBH	Al Udeid Air Base	25.1171	51.3099	Qatar
	OTBK	Al Khor Airport	25.6296	51.5067	Qatar
DOH	OTHH	Hamad International Airport	25.2677	51.6107	Qatar



SOEN-6441 Project - Flights Information System

Flights By Airport

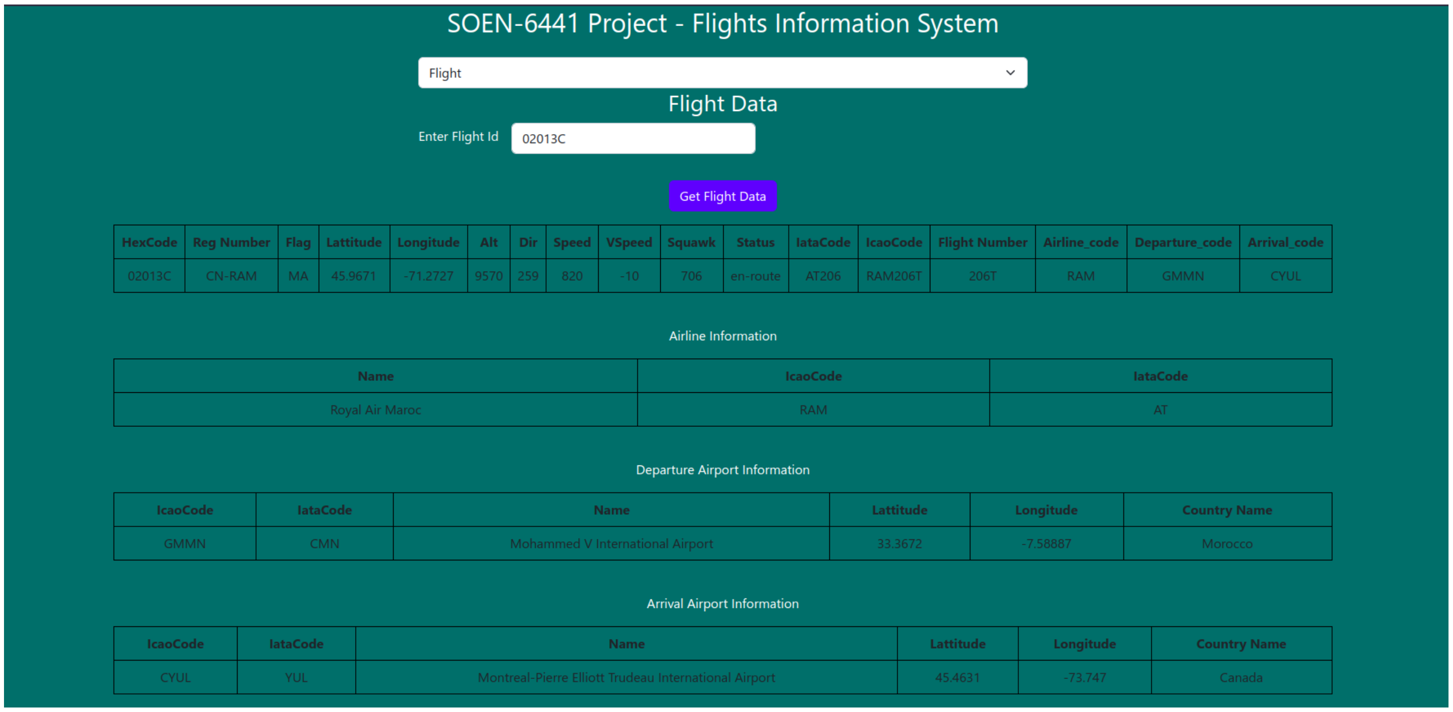
Flights By Airport

Enter Offset 0

Enter Airport Code CYUL

Get Flight Data

HexCode	Reg Number	Flag	Lattitude	Longitude	Alt	Dir	Speed	VSpeed	Squawk	Status	IataCode	IcaoCode	Flight Number	Airline_code	Departure_code	Arrival_code
02013C	CN-RAM	MA	45.9671	-71.2727	9570	259	820	-10	706	en-route	AT206	RAM206T	206T	RAM	GMMN	CYUL
39A419	F-HJAZ	FR	45.3858	-71.1567	8778	85	946	5	6215	en-route	SS901	CRL901	901	CRL	CYUL	LFPO
39C424	F-HRBE	FR	50.3404	-65.0428	11582	231	746	0	0	en-route	AF348	AFR348	348	AFR	LFPG	CYUL
406A9D	G-ZBJF	UK	50.67	-65.46	12192	230	774	0		en-route	BA95	BAW6NC	6NC	BAW	EGLL	CYUL
4B187F	HB-JHF	CH	46.3127	-67.9392	10668	91	968	0	6562	en-route	LX87	SWR87	87	SWR	CYUL	LSZH
A01820	N105JS	MX	37.0797	-83.6039	11887	36	916	0	2675	en-route	AM636	AMX636	636	AMX	MMMX	CYUL
A036FC	N1127P	US	45.4205	-74.9439	9890	260	624	12	6206	en-route		LXJ363	363	LXJ	CYUL	KMDW
A0BAA1	N146SY	US	42.0027	-88.0511	762	89	298	-4	637	en-route	UA5685	SKW5685	5685	SKW	CYUL	KORD
A3025D	N293TW	US	43.4583	-73.3916	5791	356	505	0	2654	en-route		GPD293	293	GPD	KTEB	CYUL
A9A749	N721AF	US	40.6376	-74.4017	1912	221	424	0	1705	en-route		CNS2041	2041	CNS	CYUL	KJFK
AB4AF1	N827AW	US	39.2048	-77.7086	10980	213	777	0	6274	en-route	AA1350	AAL1350	1350	AAL	CYUL	KCLT
ACC59D	N922AE	US	44.5896	-79.316	9144	243	742	0	1055	en-route	AA3603	ENY3603	3603	ENY	CYUL	KORD



SOEN-6441 Project - Flights Information System

Flight

Flight Data

Enter Flight Id 02013C

Get Flight Data

HexCode	Reg Number	Flag	Lattitude	Longitude	Alt	Dir	Speed	VSpeed	Squawk	Status	IataCode	IcaoCode	Flight Number	Airline_code	Departure_code	Arrival_code
02013C	CN-RAM	MA	45.9671	-71.2727	9570	259	820	-10	706	en-route	AT206	RAM206T	206T	RAM	GMMN	CYUL

Airline Information

Name	IcaoCode	IataCode
Royal Air Maroc	RAM	AT

Departure Airport Information

IcaoCode	IataCode	Name	Lattitude	Longitude	Country Name
GMMN	CMN	Mohammed V International Airport	33.3672	-7.58887	Morocco

Arrival Airport Information

IcaoCode	IataCode	Name	Lattitude	Longitude	Country Name
CYUL	YUL	Montreal-Pierre Elliott Trudeau International Airport	45.4631	-73.747	Canada

Testing

We tested two major parts of our codebase:

1. Model creation and data instantiation
2. Data mappers and data passing.

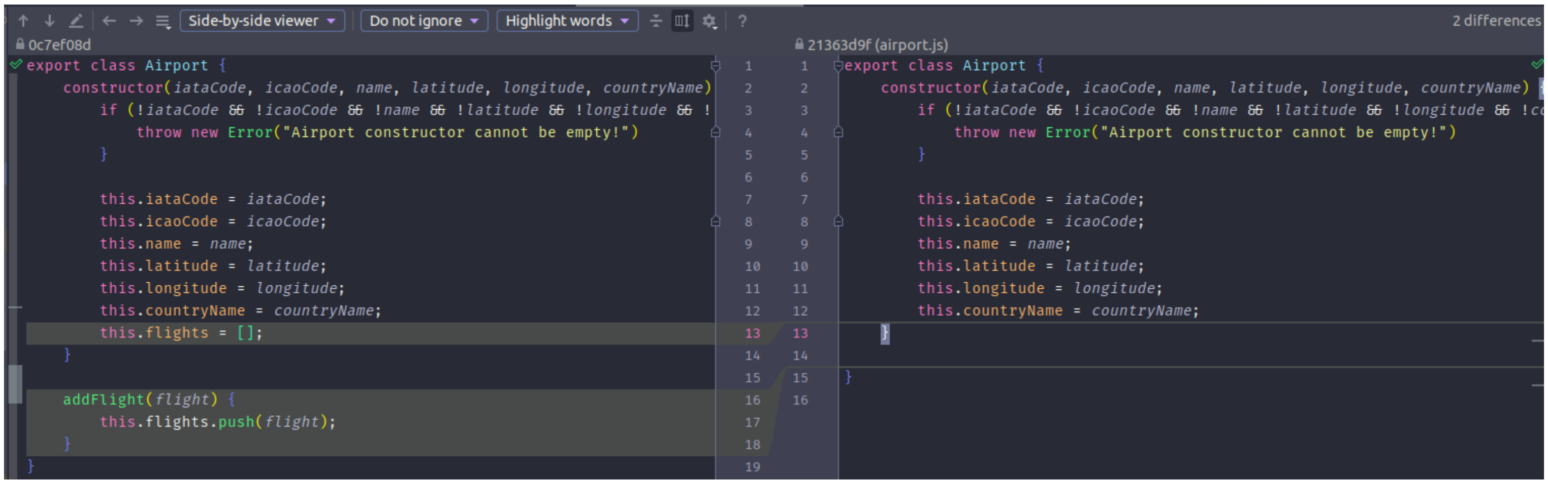
To do so, we relied on the “mocha” testing framework and the “should” assertion library. For the first set of test cases (model creation), we asserted that creating “empty” models should throw an error (which is a result of a check we added to each constructor, forcing at least one of the constructor parameters to be defined and not null.

For the second set of tests, we created a set of fake data rows that would match how the queries would return them from the database. We then created a mock DataGateway class that would “query” this set of data instead of the actual database. Finally, we used the actual data mappers and data manager (linked to this fake gateway class) to make sure that the mappers correctly map the raw data to the models. The assertions here are simply to make sure that the data inside the returned objects exactly match the fake data.

Refactoring Strategies Used

While developing the project, the following refactoring strategies were used:

1. Change Bidirectional Association to Unidirectional: in the initial designs of the project, the Airport class previously held an instance of all flights that were scheduled to leave or depart from it. However, when querying data and mapping it to the models, this created a cyclical dependency. To fix this, we removed the association from airports to the corresponding flights. The following image shows the git comparison before (left) and after (right) the change.



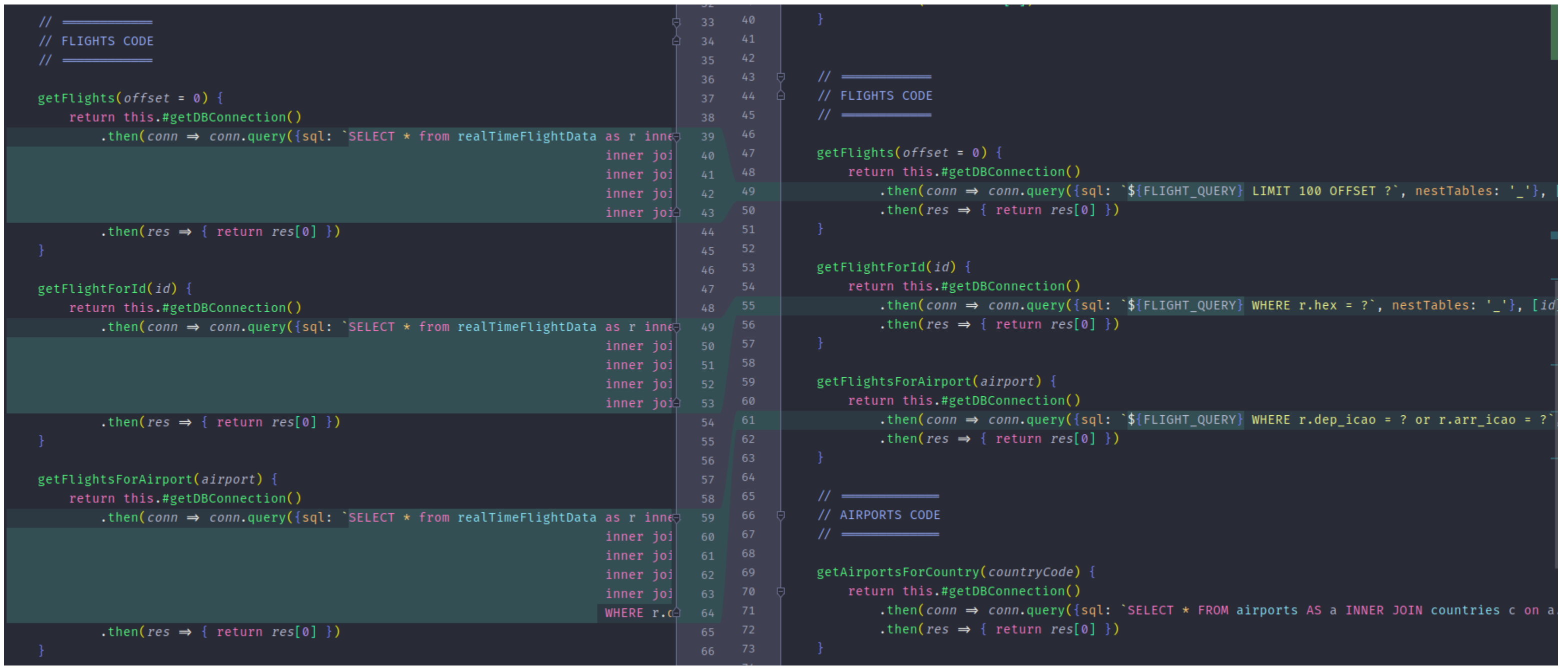
0c7ef08d

```
export class Airport {
    constructor(iataCode, icaoCode, name, latitude, longitude, countryName)
        if (!iataCode && !icaoCode && !name && !latitude && !longitude && !
            throw new Error("Airport constructor cannot be empty!")
        }

        this.iataCode = iataCode;
        this.icaoCode = icaoCode;
        this.name = name;
        this.latitude = latitude;
        this.longitude = longitude;
        this.countryName = countryName;
        this.flights = [];
    }

    addFlight(flight) {
        this.flights.push(flight);
    }
}
```

21363d9f (airport.js)

```
export class Airport {
    constructor(iataCode, icaoCode, name, latitude, longitude, countryName)
        if (!iataCode && !icaoCode && !name && !latitude && !longitude && !c
            throw new Error("Airport constructor cannot be empty!")
        }

        this.iataCode = iataCode;
        this.icaoCode = icaoCode;
        this.name = name;
        this.latitude = latitude;
        this.longitude = longitude;
        this.countryName = countryName;
    }
}
```

2. Replace magic string with symbolic constant: while developing the DataGateway class, we noticed many of the SQL queries were very similar, and were making the code file long. As such, we replaced each of these occurrences with a string constant that is declared once. The following screenshot shows the git difference before and after the changes.



```
// ============
// FLIGHTS CODE
// ============

getFlights(offset = 0) {
    return this.#getDBConnection()
        .then(conn ⇒ conn.query({sql: `SELECT * from realTimeFlightData as r inne
                                                                  inner joi
                                                                  inner joi
                                                                  inner joi
                                                                  inner joi
        .then(res ⇒ { return res[0] })
}

getFlightForId(id) {
    return this.#getDBConnection()
        .then(conn ⇒ conn.query({sql: `SELECT * from realTimeFlightData as r inne
                                                                  inner joi
                                                                  inner joi
                                                                  inner joi
                                                                  inner joi
        .then(res ⇒ { return res[0] })
}

getFlightsForAirport(airport) {
    return this.#getDBConnection()
        .then(conn ⇒ conn.query({sql: `SELECT * from realTimeFlightData as r inne
                                                                  inner joi
                                                                  inner joi
                                                                  inner joi
                                                                  inner joi
                                                                  WHERE r.c
        .then(res ⇒ { return res[0] })
}
```

```
}


// ============
// FLIGHTS CODE
// ============

getFlights(offset = 0) {
    return this.#getDBConnection()
        .then(conn ⇒ conn.query({sql: `${FLIGHT_QUERY} LIMIT 100 OFFSET ?`, nestTables: '_'},
        .then(res ⇒ { return res[0] })
}

getFlightForId(id) {
    return this.#getDBConnection()
        .then(conn ⇒ conn.query({sql: `${FLIGHT_QUERY} WHERE r.hex = ?`, nestTables: '_'}, [id
        .then(res ⇒ { return res[0] })
}

getFlightsForAirport(airport) {
    return this.#getDBConnection()
        .then(conn ⇒ conn.query({sql: `${FLIGHT_QUERY} WHERE r.dep_icao = ? or r.arr_icao = ?`
        .then(res ⇒ { return res[0] })
}

// =============
// AIRPORTS CODE
// =============

getAirportsForCountry(countryCode) {
    return this.#getDBConnection()
        .then(conn ⇒ conn.query({sql: `SELECT * FROM airports AS a INNER JOIN countries c on a
        .then(res ⇒ { return res[0] })
}
```