

# FLIGHT BUDDY

The Future of Air Travel

By Zain Sheikh, Zelong Zhang, and Dongheok Lee



# Abstract

Here, we present to you, the “Flight Buddy” web application. In a world where the collection of data is becoming evermore necessary and the need to implement structures that use that data is a prerequisite to any successful organization or company, we have created a website that demonstrates the basic requirements needed to excel in this new world of data science. In this paper, we will demonstrate the service that our website intends to provide. First, there will be an introduction to the real-world problem with which we are concerned, as well as our proposed solution. We will then move swiftly forward to the design of the platform, followed by the implementation. Next, we will take a look at the API endpoints and their interactions with Postman. Finally, we will provide a user guide to help navigate and use the website as intended.

## Introduction

With the evident growth of commercial air travel in recent years, the intricacies of interacting with a database system that provides information on flight connections and airports has become increasingly complex. Travellers who want to find the optimal flights for themselves, based on numerous different preferences such as flight durations, flight distance, hotels, and so on may find it very difficult to do so efficiently. Furthermore, managers from different companies who are tasked to manage certain flight connections may find it very difficult to keep track of and organize their flight connections.

Our Proposed solution for these problems was to create a one-stop platform for both potential travellers as well as flight managers to access the complex database of air travel-related information in an organized and meaningful way.

*Flight Buddy* is a website that allows clients to make search queries on flight connections and navigate through a customized list of connections that fit their specific needs. Furthermore, flight managers are able to log into the system and manage all the connections related to them. We will delve deeper into the intricacies of the service we provide, as well as the roles of different types of users on our platform in the following *Design* section.

# Design

The design of our platform was created based on the functionalities we provide to the two distinct types of users:

## The Client

The client is able to register themselves onto our website and thus create an account, which they can log in and out of. After registration, the client is then given access to a number of functionalities. This includes viewing and updating their user information, as well as being able to make searches on flight connections. To make a search, the client simply has to fill in the form, specifying their preferences on time, date, price, hotels, and travel distance. The website then filters the database of flight connections and returns to the user, a list of connections that match their preferences. The platform uses a database to keep track of hotels, cities, airports, flight routes, and flight connections to provide an in-depth and meaningful service to the client.

## The Flight Manager

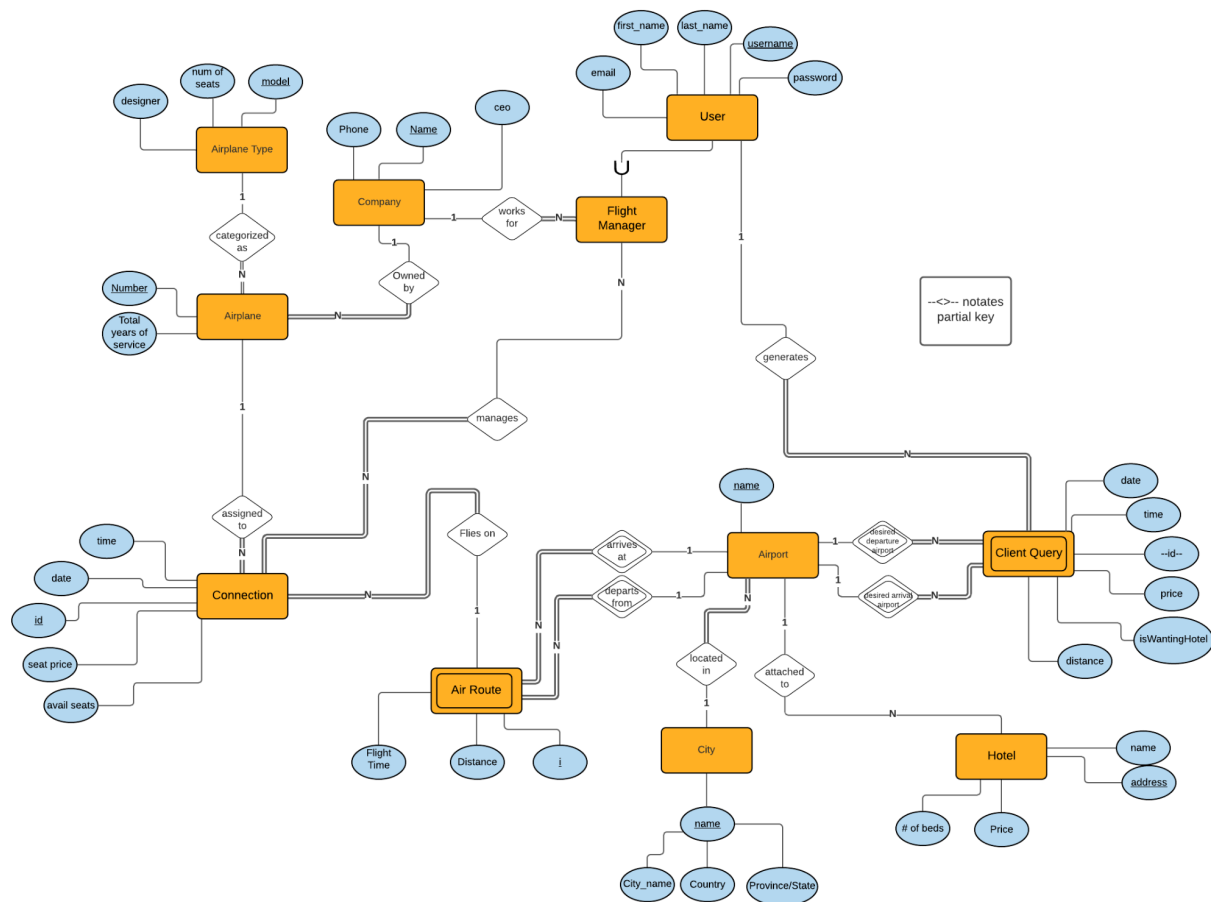
A user cannot make themselves a flight manager, but rather need to be added to the database of flight managers by an admin of the database. This provides security against fraud. Once a user has been elevated to a flight manager, they gain access to new features in addition to the basic ones offered to the client. Simply put, every user is a client and thus every flight manager is a client. But not every user/client is a flight manager.

Flight managers are given the extra functionality of being able to view all the connections that they manage. They can also add new connections as well as remove old connections. The database keeps track of connections and the air routes on which the connection travels on and the airplane which is used for that connection. All these details are provided so that the manager can use *Flight Buddy* to customize their experience with small details.

## The Industry

We chose to lean into the idea of scalability by implementing a database that provides information on all sorts of small details. As mentioned before, air routes and flight connections are kept track of in order to provide customized search results to the client, as well as to provide a space for managers to keep track of their flights. In addition to this, the website incorporates information on airplanes, airplane models, and companies. This is to provide scalability for the website to go from servicing individuals, to servicing the entire commercial air travel industry; keeping track of planes, connections, companies, and the interaction of all those entities. This

allows industry to use *Flight Buddy* to holistically understand the state and functionality of a multinational travel system.



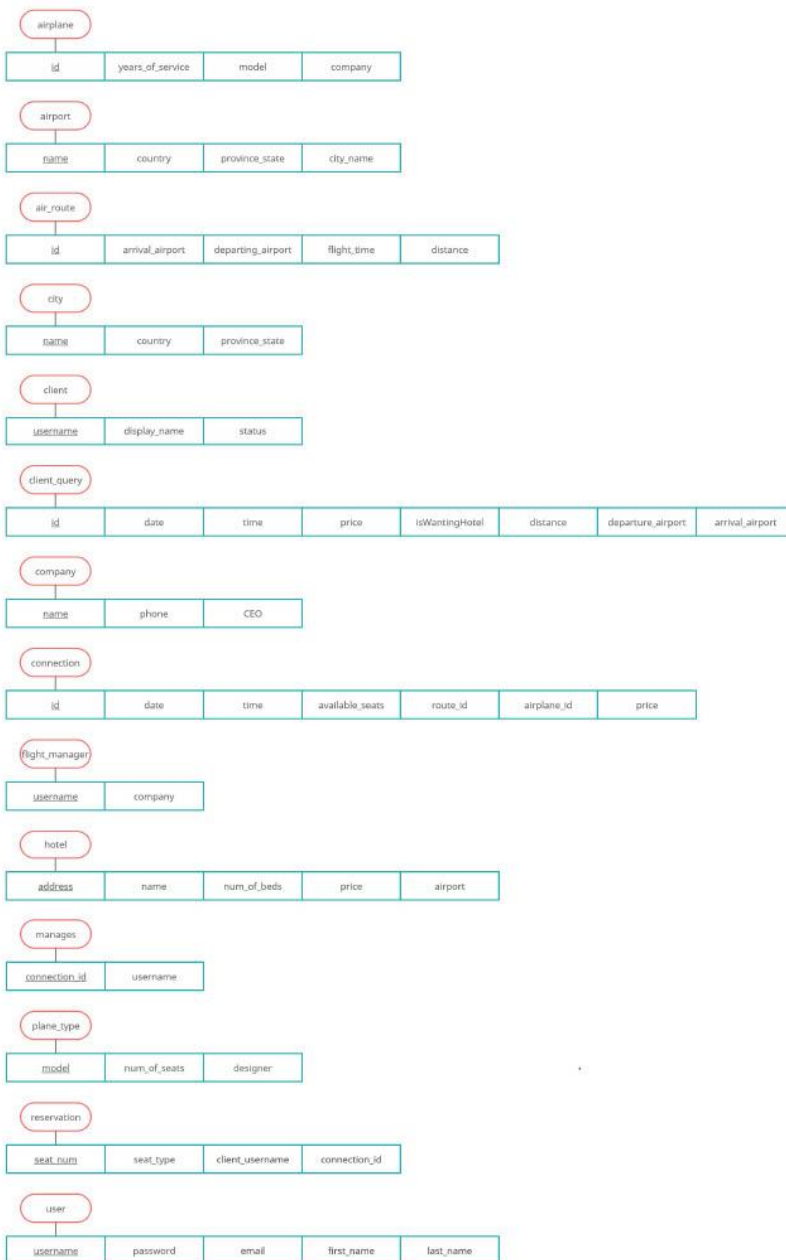
**figure 1.0: *Flight Buddy* Design Entity Relationship Diagram**

As seen in figure 1.0, the complex relationships between the finer details of the database are kept track of in this manner. This diagram was used to model the functionality and overall service of our website. The details on the necessity of these entities and their relationships has been explained in the *Design* section.

With this design model and functionality in mind, we then set out to implement these ideas into our actual working model.

# Implementation

Here is the relation model diagram to the implementation. It documents all the data structure of our database.



**Figure 2.0: Flight Buddy Relational Model**

This diagram was created to represent the structure of the database of Flight Buddy and how to implement it based on the structure of it. As represented in the figure 2.0, each red ovals represent the entities in the ERD diagram in figure 1.0, and the green rectangles represent the attributes and the key attributes if underlined. Some of the attributes in the entity are implemented to link between the two attributes. For example, the `departing_airport` and the `arrival_airport` from the entity `air_route` is used to link the relationship between `air_route` and `airport`. Furthermore, it has attributes like `flight_time`, and `distance` are the additional attributes and `id` is the unique key attribute. For any unusual decisions, due to our time limit, not everything that we planned in our proposal could be implemented. However, everything is still stored in the database. For example, the reservation is removed from our plan, but it still exists in our database.

## Front End Connection

For our front end, we used html, css and plain javascript. The website uses html and css to customize the look and feel of static data and javascript was used to dynamically display data and to make requests to our backend. For our request API, we used javascript's Fetch API.

## User Login Authentication

As a part of our design, we decided that our core features such as profile, connections, and searches would only be available to logged in users. Thus, we needed a way to ensure users are still logged in and operating in a single session. To do this, we incorporated the usage of json web tokens (jwt). Specifically, we used Firebase's `php-jwt` library which can be found at (<https://github.com/firebase/php-jwt>). Using a unique secret key, we are able encode and serve a jwt in response to a successful user login. When our frontend receives this jwt, it is kept in local storage and included in the header of any request which requires a logged in user status. Our API endpoints are able to decode the jwt and cross reference to see if the user who submitted the request and the submitted jwt matches. In this way, we also protect our endpoints from illegal access.

## Flight Schedule Search

After

## SQL Queries

For this implementation, we made use of SQL queries to access the database when trying to get, update, or delete information. To avoid SQL injections, we used mysqli's prepared statements. The query is "prepared" with placeholders (question marks) instead of being directly executed. To replace the placeholders, we bind real parameters but also pass in their expected type such that MySQL knows to treat them as strings, integers, doubles, or blobs.

### **connectioninc.php**

This file contains useful functions for dealing with flight connection related requests.

- `connectionExists($conn, $id)`

Query:

```
SELECT * FROM connection WHERE id = ?;
```

Checks if a connection exists in the database based on its primary key of id. Query selects all connections where id is equivalent to the id submitted.

- `routeExists($conn, $route)`

Query:

```
SELECT * FROM air_route WHERE id = ?;
```

Checks if an air route exists in the database based on its primary key of id. Query selects all air route rows where id is equivalent to the route submitted.

- `planeExists($conn, $plane)`

Query:

```
SELECT * FROM airplane WHERE id = ?
```

Checks if an airplane exists in the database based on its primary key of id. Query selects all airplane rows where id is equivalent to submitted plane.

- `createConnection($conn, $id, $date, $time, $availableSeats, $routeId, $airplaneId, $username)`

Query:

```
INSERT INTO connection (id, date, time, available_seats,  
route_id, airplane_id, price) VALUE (?, ?, ?, ?, ?, ?, ?);  
INSERT INTO manages (connection_id, username) VALUE (?, ?);
```

Creates a new connection to be managed by the username submitted. The first query adds a new connection to the connection table. The second query links the new connection to the user by adding a new entry into the manages table.

- `getAssocConnections($conn, $username)`  
Query:  

```
SELECT DISTINCT C.id, C.date, C.time, C.available_seats,  
C.route_id, C.airplane_id, C.price FROM connection AS C, manages  
AS M WHERE C.id = M.connection_id AND M.username = ?;
```

  
Finds all connections associated with the username submitted. The query joins two tables, connections and manages to obtain an intermediate table of every managed connection with their respective connection information. From this intermediate table, specific columns of the rows where username is equivalent to the submitted username are selected.
- `isConnectionManaged($conn, $connection_id, $username)`  
Query:  

```
SELECT * FROM manages WHERE connection_id = ? AND username = ?;
```

  
Checks if a flight is managed by the specific flight manager identified by username. Query selects all rows in manages where the connection id is equivalent to the submitted connection id and username is equivalent to the submitted username.
- `removeConnection($conn, $connection_id)`  
Query:  

```
DELETE FROM connection WHERE id=?;
```

  
Deletes a connection from the database. Query deletes any row in connection where id is equivalent to the submitted connection\_id.

## **profileinc.php**

This file contains useful functions for handling account and profile information.

- `updateAccount($connection, $username, $fName, $lName, $password, $email)`  
Query:  

```
UPDATE user SET first_name = ? WHERE username = ?;
```

  
Updates the first name of the desired user. Query sets the first\_name of all rows in user table where username is equivalent to the submitted username.  
  
Query:  

```
UPDATE user SET last_name = ? WHERE username = ?;
```

  
Updates the last name of the desired user. Query sets the last\_name of all rows in user table where username is equivalent to the submitted username.  
  
Query:  

```
UPDATE user SET password = ? WHERE username = ?;
```



Updates the password of the desired user. Query sets the password of all rows in user table where password is equivalent to the submitted username.

Query:

```
UPDATE user SET email = ? WHERE username = ?;
```

Updates the email of the desired user. Query sets the email of all rows in user table where username is equivalent to the submitted username.

- `getUserProfile($conn, $username)`

Query:

```
SELECT * FROM user WHERE username = ?;
```

Returns the desired user's account information. Query selects all columns from user where the row's username is equivalent to the given username.

### **searchinc.php**

This file contains useful functions for making search queries on the flight connections

- `getFlights($connection, $price, $flightTime, $maxDistance, $departure, $arrival, $date)`

Query:

```
SELECT DISTINCT C.id, C.price, C.time, C.date, departure_airport,
               arrival_airport, distance, flight_time, C.available_seats
FROM air_route AS A, connection AS C
WHERE
    C.route_id = A.id AND
    C.price <= ? AND
    C.date = ? AND
    A.flight_time <= ? AND
    A.distance <= ? AND
    A.departure_airport = ? AND
    A.arrival_airport = ? ;
```

Gets all the information from flight connections and air routes that meet the desired criteria. Query selects specific columns from a joint table between air\_route and connection where rows meet the specified requirements.

## **logininc.php**

This file contains useful functions for logging in and registering

- `isUsernameInDb($conn, $username)`

Query:

```
SELECT * FROM user WHERE username = ?;
```

This query returns all users from the database that have the mentioned username. The function will return true if such a tuple is found.

- `isEmailInDb($conn, $email)`

Query:

```
SELECT * FROM user WHERE email= ?;
```

This query returns all users from the database that have the mentioned email. The function will return true if such a tuple is found.

- `isUserRegistered($conn, $username, $password)`

Query:

```
SELECT * FROM user WHERE email= ? AND password = ?;
```

This query returns all users from the database that have the mentioned username AND password. The function will return true if such a tuple is found.

- `createUser($connection, $username, $fname, $lname, $password, $email)`

Query:

```
INSERT INTO user (username, password, email, first_name,  
last_name) VALUE (?, ?, ?, ?, ?);
```

Adds a tuple into the database, under the USER table, with the appropriate values.

- `isUserManager($connection, $username)`

Query:

```
SELECT * FROM flight_manager WHERE username = ?;
```

Cross references the username with the flight\_manager table and returns true if such a manager is found.

# API Endpoints

The following API documentation was created using Postman's API documentation tool. It fully documents all endpoints created in this project and all parameters, headers, and body fields needed to make requests to each of the endpoints. In addition, an example request and response has been provided for each endpoint.

It should be noted that some example requests are expected to return errors with the provided database. This is because a demo database was used to run the example requests to protect the actual database's security. The accounts "kevin" and "newaccount" are not in the provided database and would cause related requests to fail. Therefore, the example and example code provided should not be used to test the API. To test the endpoints, the tester should register with a new account, elevate it to flight manager in the database and then test with the tester account information.

The full documentation can be found at:

<https://documenter.getpostman.com/view/15322834/TzJsfdUU>

# User Guide

## Setting Up The Demo Website on Localhost

To get the website up and running on localhost, you will need to install the Xampp application. This will allow you to run an apache server and the MySQL server.

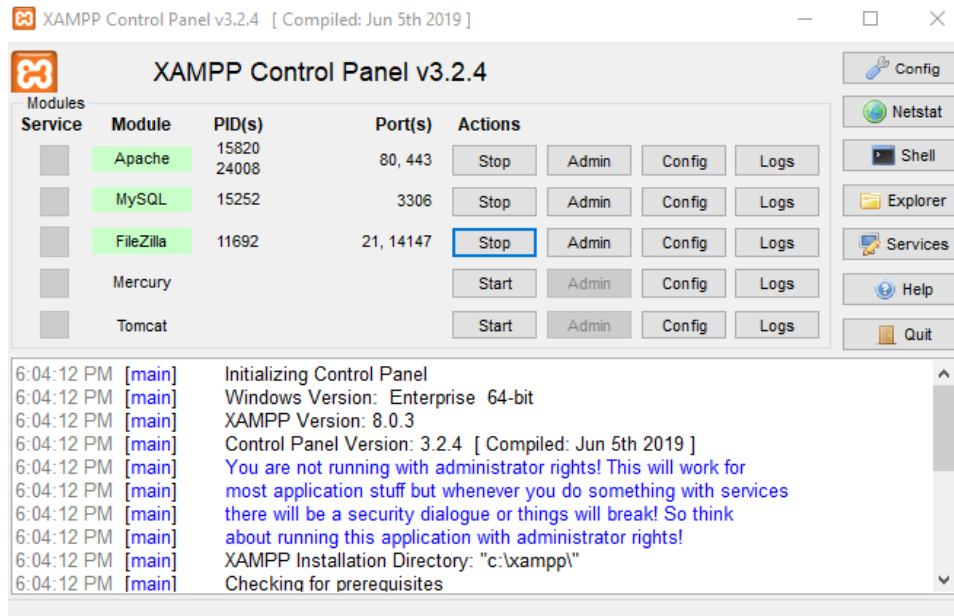


figure 6.0

After completing this step and starting the server, please download the entire “Flight-Buddy” zipped folder. Extract the contents and place the contents into the xampp/htdocs directory. Make sure you place the contents directly into this folder, without any other “Flight-Buddy” folder.

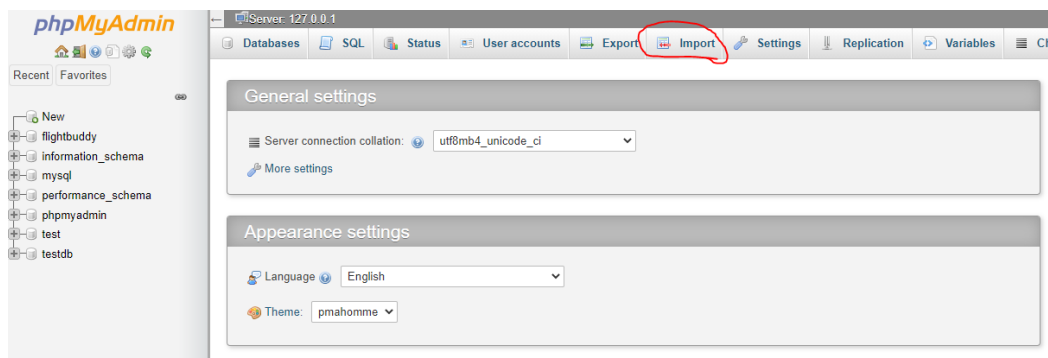
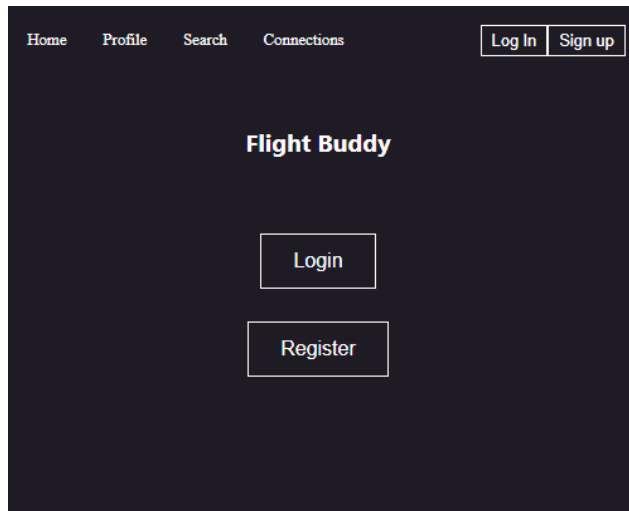


figure 6.1

In order to use a database with premade values, import the provided .sql file into an already-created database called “flightbuddy” in your localhost/phpmyadmin.

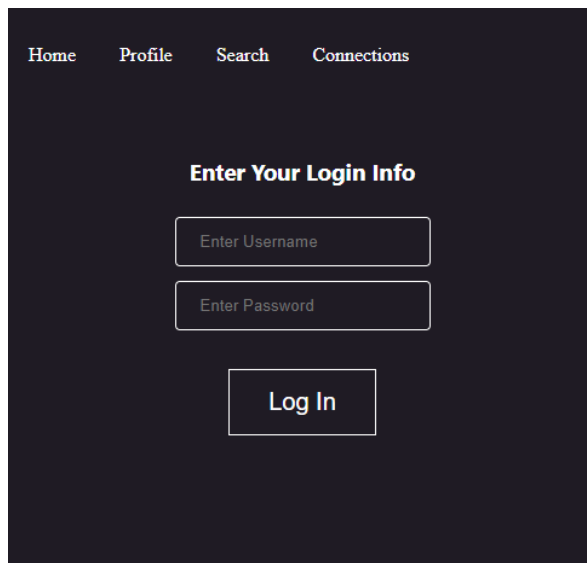
Now that you have the system set up, you can begin using the demo website! Because *Flight Buddy* offers services to two different types of users, a guide for each user is provided.

## Signing Up / Logging In



**figure 6.2:** <http://localhost/Home.html>

Once you have navigated to the website home page, you may choose to login to an existing account or register to a new account.



**figure 6.3:** <http://localhost/login.html>

If you have an existing account, you may log in with the appropriate information, otherwise you may navigate to the register page.

The image shows a web registration form on a dark background. At the top, there is a navigation bar with links: Home, Profile, Search, and Connections. Below this, the form is titled "Enter Your Information". It contains six input fields stacked vertically: "Enter Username", "Enter Email", "Enter First Name", "Enter Last Name", "Enter Password", and "Confirm Password". Each field is a light gray rectangle with rounded corners and a thin border. At the bottom of the form is a "Register" button, which is a light gray rectangle with rounded corners and a thin border, containing the text "Register" in a bold, sans-serif font.

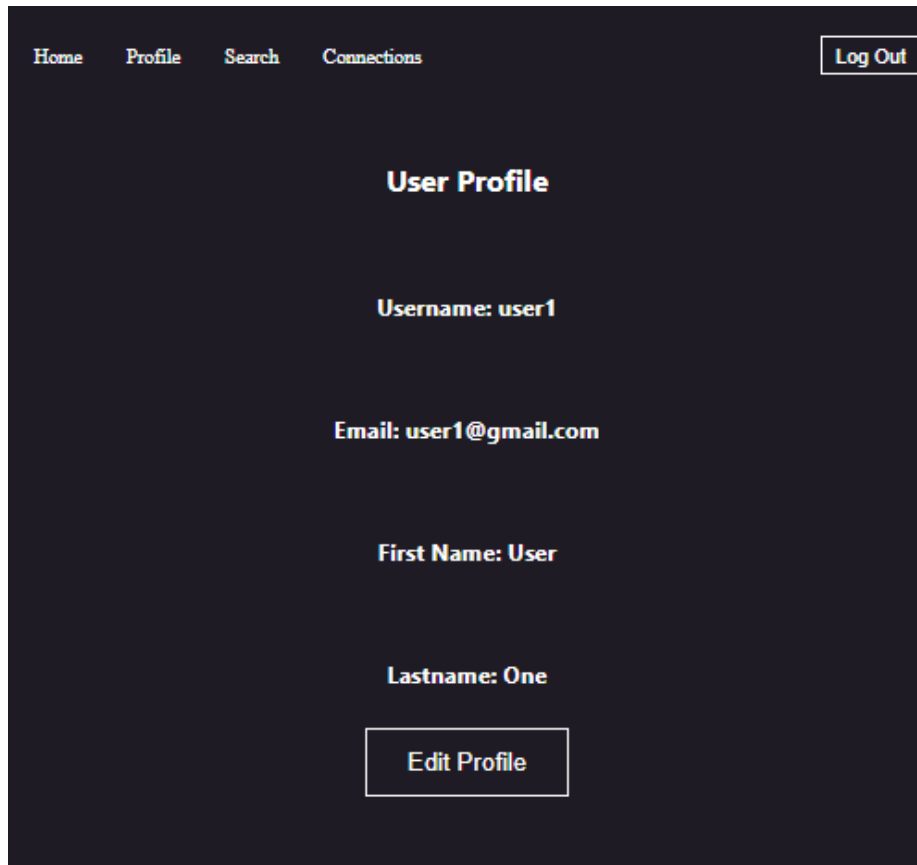
**figure 6.4:** <http://localhost/register.html>

On the registration page, you can sign up for a new account with a unique username and a valid email.

Congratulations, you now have an official *Flight Buddy* account!

You may now navigate to the login page and sign in to the system.

## Regular Client Functionalities

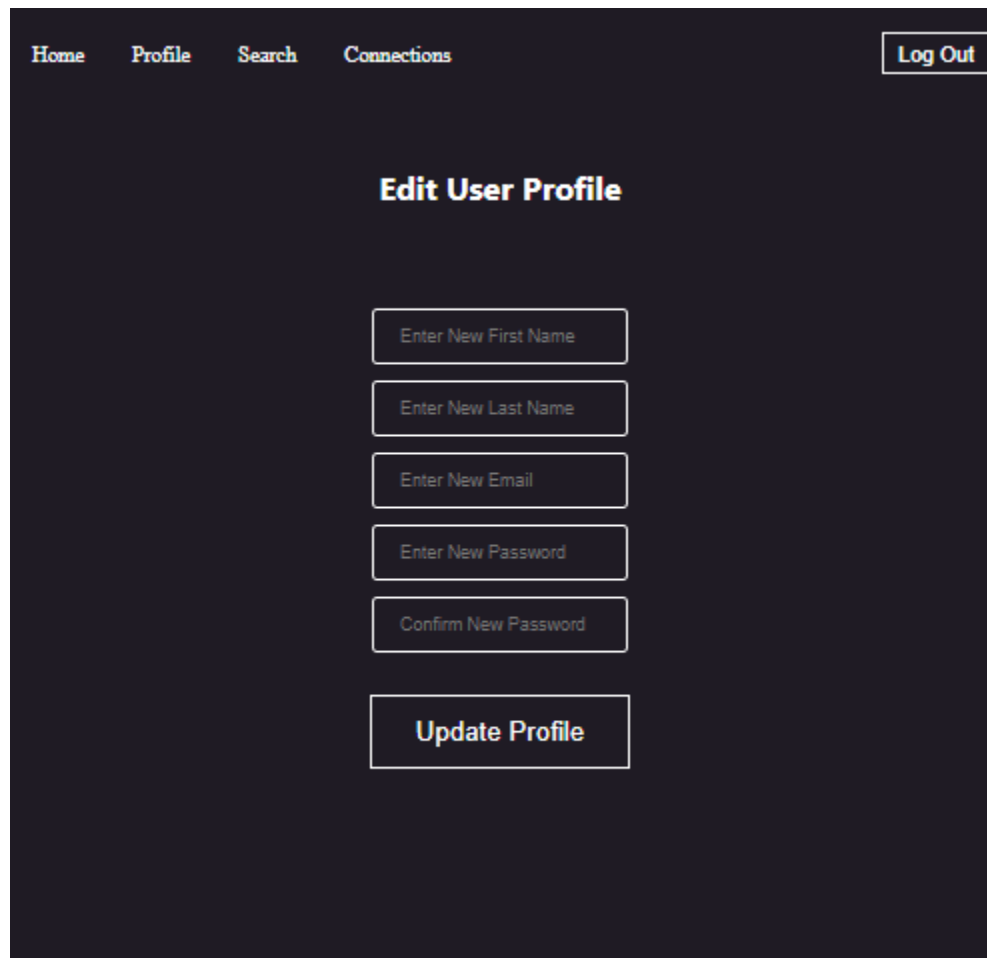


**figure 6.5:** <http://localhost/profile.html>

After logging in, you will be greeted with a profile page, where you can view your account information. As a client, you may choose to edit your profile, make search queries, or log out of the system.

The “Connections” functionality is blocked for regular clients and thus you must be a flight manager to access that page.

## Edit Profile



The screenshot shows a web application interface for editing a user profile. At the top, there is a dark navigation bar with links for 'Home', 'Profile', 'Search', and 'Connections'. A 'Log Out' button is located in the top right corner. The main content area has a dark background and features the title 'Edit User Profile' in white. Below the title, there are five text input fields stacked vertically, each with a light gray placeholder text: 'Enter New First Name', 'Enter New Last Name', 'Enter New Email', 'Enter New Password', and 'Confirm New Password'. At the bottom of the form is a large, light-colored button with the text 'Update Profile' in bold.

**figure 6.6:** <http://localhost/editProfile.html>

Here, you may enter new information to change your account information. This change will be reflected in the database, and thus also be reflected on your main profile page.



[Home](#) [Profile](#) [Search](#) [Connections](#) [Log Out](#)

## Search For Your Flight

Departure Airport

YYC

Arrival Airport

YYZ

Maximum Flight Distance

100000

Maximum Flight Duration

10000

Maximum Price

10000

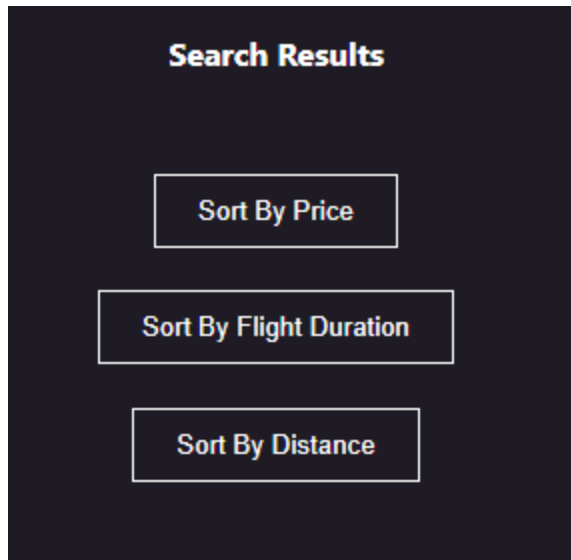
Flight Date

04/16/2021

Search

**figure 6.7:** <http://localhost/Search.html>

On the search page, you may fill out the form according to your needs. After hitting “Search”, you will be presented with a list of flight connections that meet your needs.

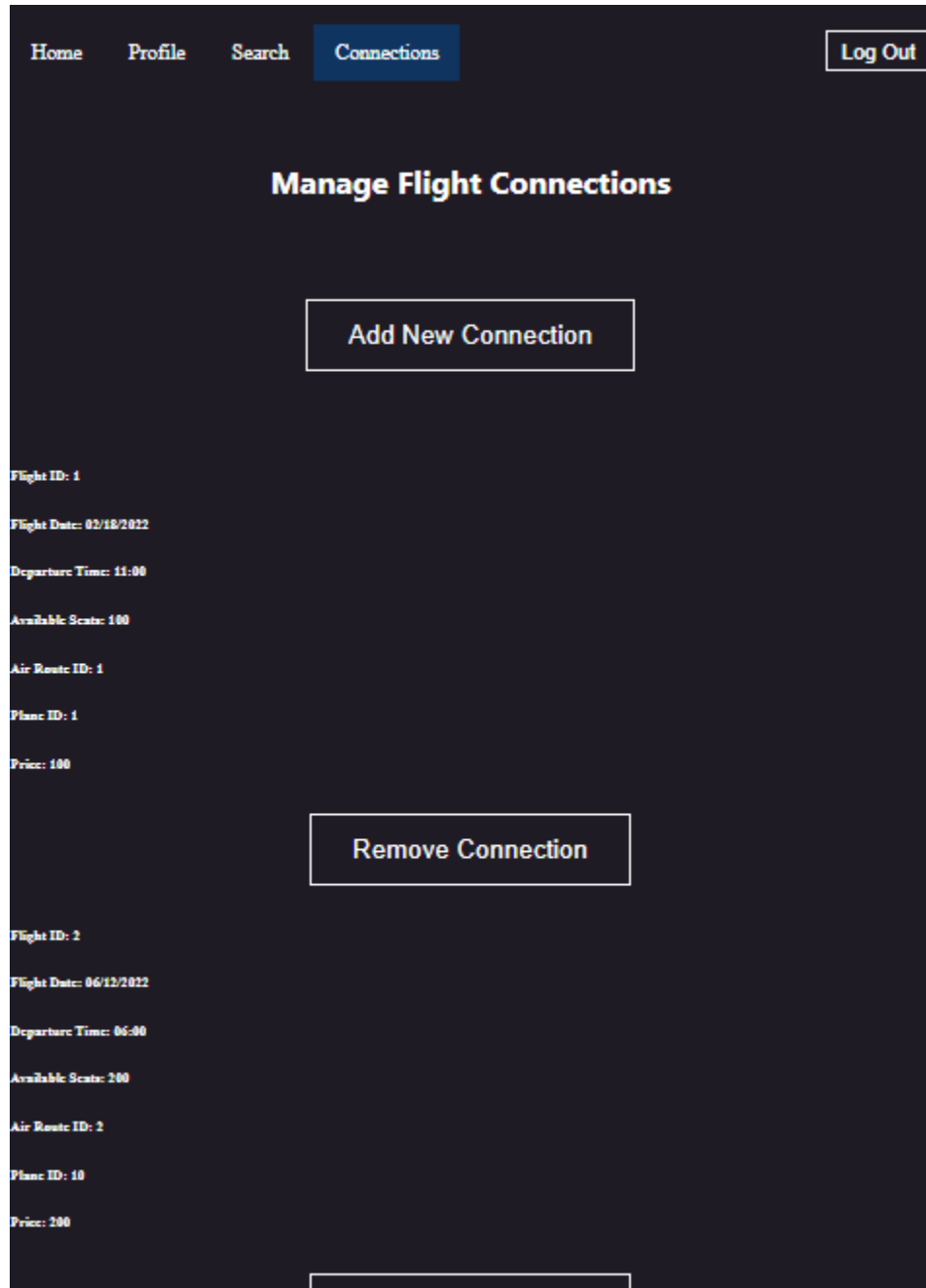


**figure 6.8:** <http://localhost/searchResult.html>

On the results page, you may choose to sort the flight connections by pressing on any of these three buttons.

## Flight Manager

To become a flight manager, you must be manually added into the system by a *Flight Buddy* admin.



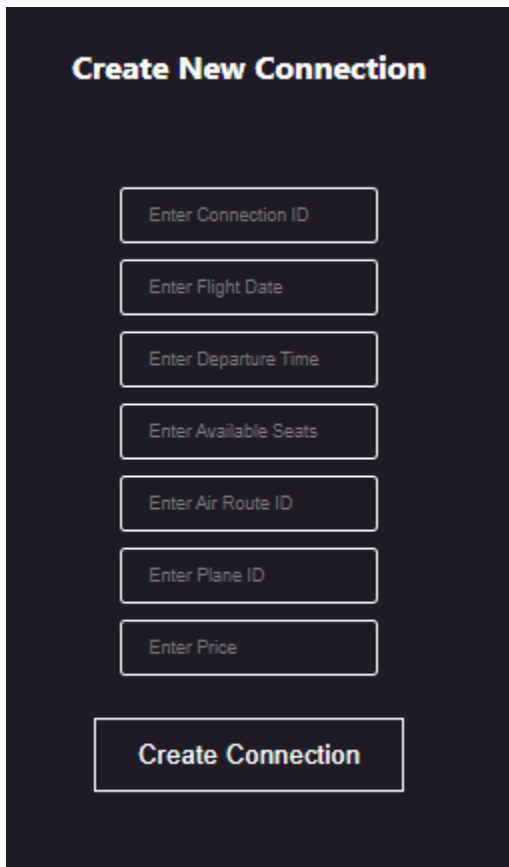
**figure 6.9:** <http://localhost/Connections.html>

Once you are given admin access, you will be able to use the “Connections” functionality.

On this page, you will be able to see all the connections that you manage. Furthermore, you can remove connections from your account by clicking “Remove Connection” under the connection you wish to delete.

You may also want to add a new connection, which you can do by clicking on the “Add New Connection” button.

### Adding Connection

A screenshot of a web form titled "Create New Connection" in a bold, black font. The form is set against a light gray background. It contains seven input fields, each with a light gray border and a light gray placeholder text. The fields are stacked vertically: "Enter Connection ID", "Enter Flight Date", "Enter Departure Time", "Enter Available Seats", "Enter Air Route ID", "Enter Plane ID", and "Enter Price". Below these fields is a large, light gray button with the text "Create Connection" in a bold, black font.

**figure 6.10:** <http://localhost/addConnection.html>

Creating a new connection is very simple. You simply have to fill out the form and click on “Create Connection”.

This new connection will now show up under your connections.

## Logging Out

Once you are done with your session and wish to securely log out, simply click the “Log Out” button on the top right-side of the screen.

Thank you for using *Flight Buddy* and we hope that you have an amazing experience!

-The Team.