FlyNow

Team #40

Team Members:

Armando Benavidez

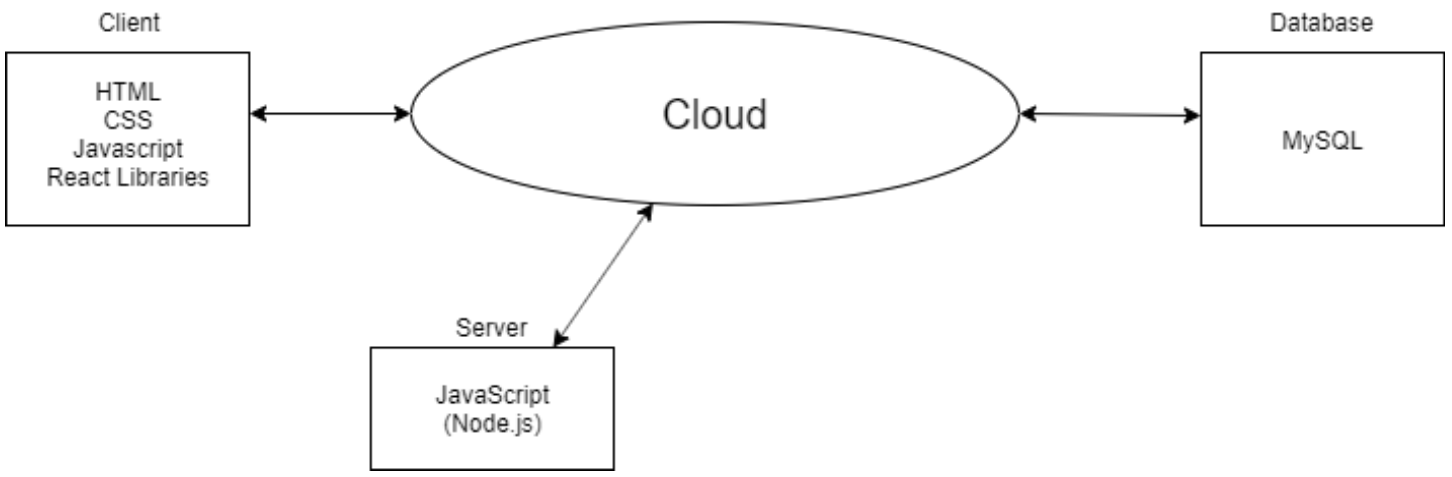
Juan Carlos Sandoval

Nithil Sethupathi

Project Description

- The aim of this project is to design a Database Application that can hold flight data and passenger information and act as a hub for managing flights. The goal is to give users the ability to book, modify, and cancel flights, as well as viewing their current flights and previous records. Admins will be able to view users' data and modify them as needed. This application will have a database that has tables with account information for users and admins, and also tables for flight information. This information will be available to those with permission to view it and it will be secured so that those without permission can't see it.
- The stakeholders would be mainly airline executives but also possibly aviation colleges. It is important to have the airline executives as stakeholders because they want a product that is tailored for their specific needs, so their input is important in designing the project. Another stakeholder could also be an aviation college, where they could study passenger and flight data, additionally a government entity could be a stakeholder such as the Federal Aviation Administration or Department of Transportation, where they could also analyze passenger and flight data it would be important for them to be stakeholders because if they analyze the data they could suggest improvements for a better flying experience.

System Environment



Hardware and Software Used

- MacBook Pro
- macOS
- Visual Studio Text Editor

RDBMS

- MySQL 8.0.15

Application Languages

- HTML, CSS, JavaScript, React.js, Node.js, SQL

Functional Requirements

The web application is designed to be accessed by multiple sets of user types: Consumers and Admins. All three types of users will have a different set of views and functionalities from each other. Since consumers are the core target, and we need more consumers, we will focus more on the consumer side of things initially. The difference between consumers and admins will mainly differ in the UI and access to data. Admins will have access to the data so that they can study the taste of consumers, whereas consumers are the data providers. So, the whole application will be developed by prioritizing the need of consumers.

- **Sign-up**
    - Consumers
        - Consumers can sign up using their Email address, first name, last name, and password. This will give them an account of which they will be able to keep track of their booking, save flights, and view previous records.
    - Administrator
        - Admin will be also able to sign up with email, name, password. This will give them an account that allows them to view flight data, and some user data, with their admin privileges.
- **Login**
    - Consumers
        - Consumers can log in using their Email address and password that they signed up with. They get access to all of the user functionality, such as searching for, booking, modifying and canceling flights. They also get access to view their current flights and previous booking history.
- **Administrator**
        - Administrator gets to access the complete data and statistics of the website. They are able to see things like flight data, users (but not their passwords), and statistics.

- **Search for flights**
    - To search for flights the user will need to specify whether it is a one-way trip or round trip. An origin and destination must be provided with a selected date(s). If an invalid origin or destination is inputted the user will be provided with a message stating that it is not a valid choice. Searching properly will result in a list of available/potential flights for the user.
- **Book a flight**
    - In order for the user to book a flight, the name of the flier(s) will need to be entered, date of birth will need to be provided, along with a valid email address to send out reservation details. If the reservation is successful the user will see a reservation confirmation page with an on-screen message saying, "Your reservation is confirmed" or "Reservation was successful" and will send a unique reservation via email. In the event the reservation is not successful there will be a message saying "Reservation Unsuccessful, please try again later" or some sort of error message.
- **Save/Delete a flight**
    - If the user desires, they can save a flight to their portal, but must first log into their account by selecting save flight when searching for flights. If they wish to book it in the future it will be on their user portal. If the user no longer wants that flight on their portal, then they can remove it.
- **Change/Cancel a flight**
    - Admins will also have the ability to change a user's flight or cancel it for any valid reason.
- **Current/previous bookings**
    - The user can log in to their account and go into their records. From here, they can see their currently booked flights or look at their records for past flights.
    - Admins may also be able to see users' currently booked flights or past flights.

<u>Non-functional Issues</u>

For the Graphical User Interface (GUI)

- Likely be using the React library for JavaScript to simplify the process of creating a clean-looking and cohesive UI. We may even use a framework such as the Material-UI framework. Material UI is based on Google's material design and it will be used to give our application a professional and coherent look.
- The first screen the user is greeted upon would be the main page which would be presented with options represented as buttons which would be to search for a flight, book a flight, change/cancel a flight, my account, and popular destinations. To keep the options distinct each button will have its own color. It is important to have these items on the main screen/landing page because the user is likely wanting to choose one of those options.

- If the user clicks the search for the flight, they would be directed to another portal that will ask for various inputs to process the request.
- If the user clicks to book the flight, it will ask the user to confirm, and it will add that flight to records.
- If an admin clicks creation/deletion, it will prompt for flight information and then options to delete or create the flight(s).
- Selecting the records button, it will direct the user a page which will be able to show your current and past trips.

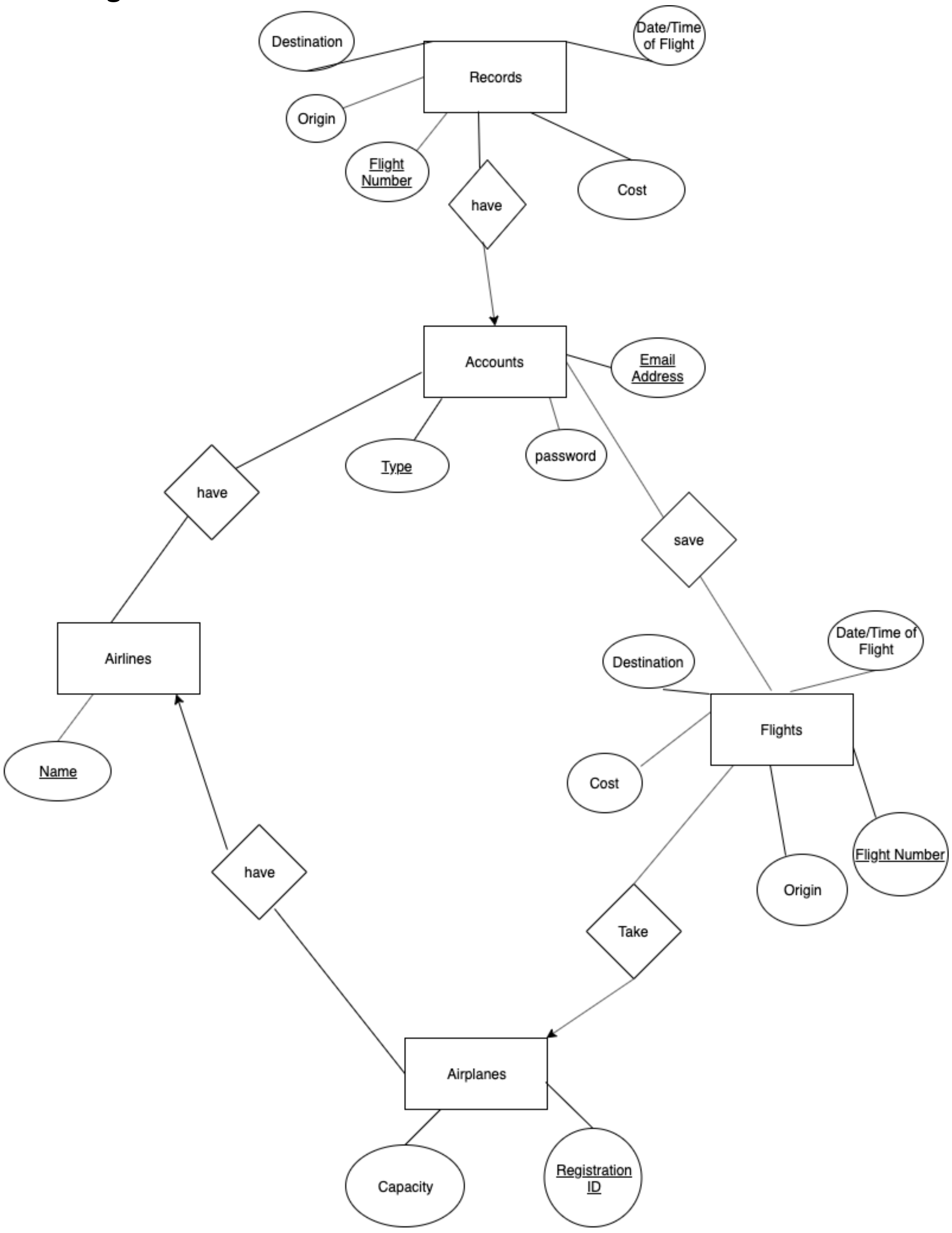
Security and Access

- We will use different portals for consumers, admin. The information will be limited depending on the user type. Information provided will be different based on the user type.

Levels of Access

- User’s portals will be completely distinct with different levels of permissions and from the admin portals.
- Admin: Will have complete access to all records with no restrictions. This account type will also have a protection mechanism after 3 attempts account will become locked for 4 hours and will not require a password reset.
- User: Will only have access to only their own portal/account.
- User can search and book their own flights and take a look at their own current/past records.

ERD Diagram


Destination
Date/Time of Flight
Records
Origin
Flight Number
have
Cost
Accounts
Email Address
Type
password
have
save
Airlines
Destination
Date/Time of Flight
Flights
Name
Cost
Origin
Flight Number
have
Take
Airplanes
Capacity
Registration ID

Entities and Attributes Description

- **Airline:** An airline will have a unique name and will have airplanes to take customers to destinations.
- **Airplanes:** Airplanes, defined by their registration numbers, go on flights to take customers to destinations.
- **Account:** An account will have a name associated with it, along with a unique email address that ties it to the account. A customer will be able to view their past and future bookings in their account. In addition, each account will have a type of an account such as customer who can only make bookings, search flights and look at their flight history. Admins will have a unique name, along with administrative privileges which will allow full access to all records.
- **Flights:** Will have a unique destination that customers can book. Each flight also has seat capacity which comes from the airplane being used. If the flight capacity becomes zero, then the customer will not be able to book that flight. Each flight will depart from an origin on a certain date along with the time to a specific destination with arrival time. Cost will be determined by popularity and destination ie. distance from home airport.
- **Records:** is the booking data, whenever the user books a new flight and wants to see the reservation it will be held in records.

Relationships

- Airlines have accounts
- Airlines have airplanes
- Airplanes take flights
- Accounts save flights
- Accounts ha ve records

Attributes of Entities

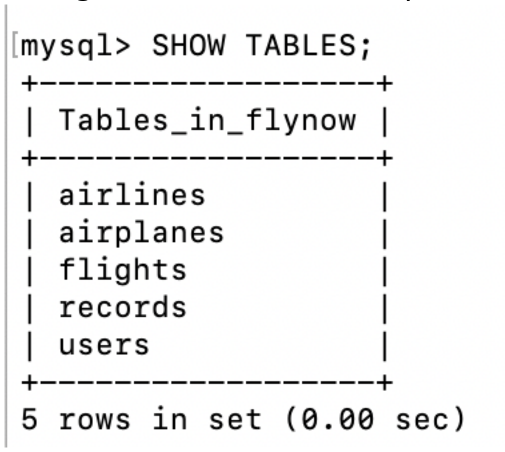
- Airlines: Name
- Airplanes: RegistrationID, Capacity
- Accounts: Email address, password, type
- Flights: Flight Number, DateOfFlight/TimeOfFlight, Origin, Destination, Cost
- Records: Flight Number, DateOfFlight/TimeOfFlight, Origin, Destination, Cost

Schemas

Accounts(Type, Email Address, Password)
Have(Type, Email Address, Credit Card)
Have(Name, Email Address, Type)
Airlines(Name)
Have(Name,RegistrationID)
Airplanes(Capacity, RegistrationID)
Take(RegistrationID,FlightNumber)
Records(FlightNumber,Cost,Destination,Date/TimeOfFlight, Origin)
Flights(FlightNumber,Cost,Destination,Date/TimeOfFlight, Origin)
Save(FlightNumber, Type, Email Address)

GETTING STARTED

1. First if not already installed on your machine you need to install NodeJS since this will be powering the FlyNow application. Link if needed:
   https://nodejs.org/en/
2. Once you have installed Node JS then you need to install Node JS Express which is our backend for the project. This can be done via your terminal or command window.
3. For our database we used MySQL which needs to be installed. Once installed you need to create a database named flynow with the following tables as shown below and your password should be password in order to be compatible. If you do not make your password 'password' you need to go into flynow->backend->routes->database.js and change the username and password as needed to allow access to your database.



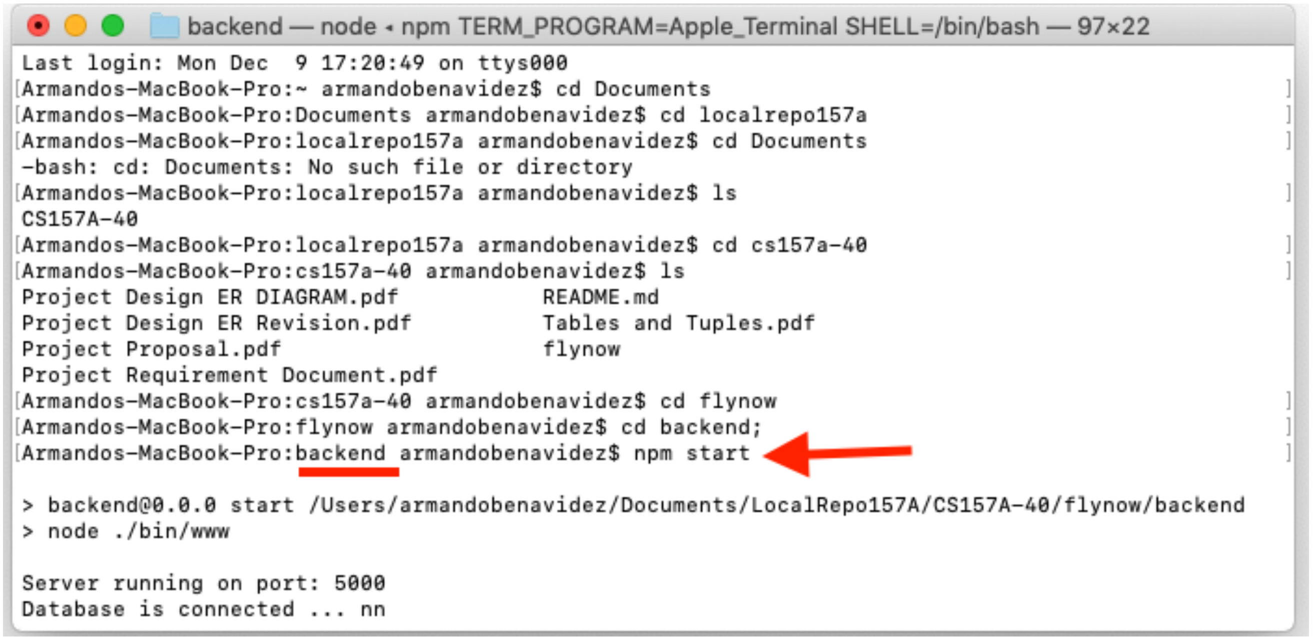
```
mysql> SHOW TABLES;
+------------------+
| Tables_in_flynow |
+------------------+
| airlines         |
| airplanes        |
| flights          |
| records          |
| users            |
+------------------+
5 rows in set (0.00 sec)
```

To create the tables, each table has the following:

```
//AIRPLANES
CREATE TABLE airplanes (
registrationID INT(3) PRIMARY KEY AUTO_INCREMENT,
capacity INT(3) NOT NULL
  );
//AIRLINES
CREATE TABLE airlines(
Name VARCHAR(30) PRIMARY KEY
);
//FLIGHTS
CREATE TABLE flights (
flightNo INT(3) UNSIGNED PRIMARY KEY,
origin VARCHAR(30) NOT NULL,
destination VARCHAR(30) NOT NULL,
date_time DATETIME NOT NULL,
 cost DECIMAL(6, 2) NOT NULL
 );
//RECORDS
```

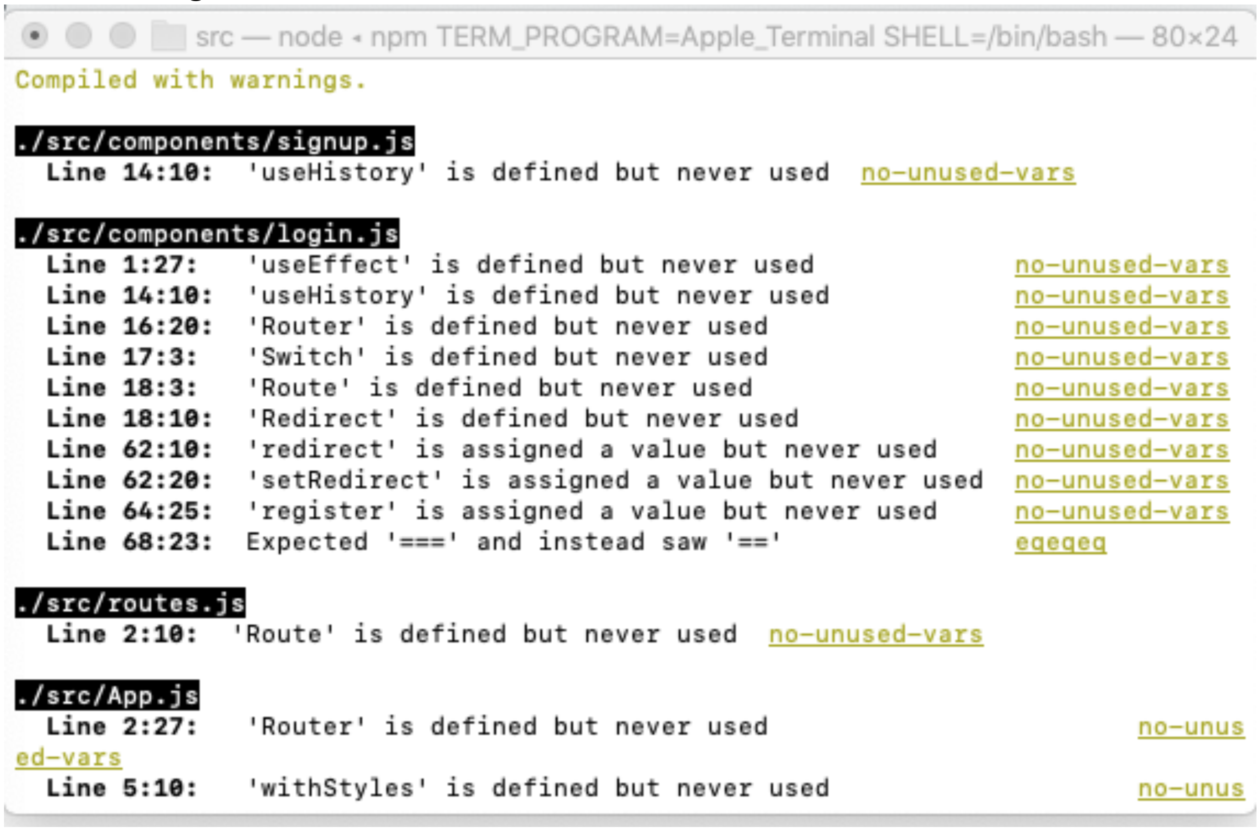
```
CREATE TABLE records (
flightNo INT(3) UNSIGNED PRIMARY KEY,
origin VARCHAR(30) NOT NULL,
destination VARCHAR(30) NOT NULL,
date_time DATETIME NOT NULL,
 cost DECIMAL(6, 2) NOT NULL
 );

//USERS
CREATE TABLE `users` (
`firstName` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
`lastName` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
`email` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
`password` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
`created` datetime NOT NULL,
`modified` datetime NOT NULL,
PRIMARY KEY (`email`)
); ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci

INSERTING TO TABLE:
//AIRPLANES
INSERT INTO airplanes(registrationID, capacity) VALUES (1, 143);
//AIRLINES
INSERT INTO airlines(name) VALUES(‘Delta’);
//FLIGHTS
INSERT INTO flights(flightNo, origin, destination, date_time, cost) VALUES (675, 'San
Francisco', 'Los Angeles', '2020-02-15 17:30:00', 125.75 );
//USERS & RECORDS
No need to insert because that will be done via our application.
```

4. Once you have completed the items above on your machine you need to go ahead and download or clone the FlyNow repository from GitHub.
5. Once you have cloned the repository or downloaded master files in your system then you need to use the terminal window or command line to go into the directory where the flynow folder is located.
6. Once you are in the flynow directory you need to go into backend, and once you are there you must type in the terminal ‘npm start’ so that the backend starts. This is important because this is where we start the local server and database at the end if everything is processed appropriately you will get the following as shown below.


backend — node ◂ npm TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash — 97×22

```
Last login: Mon Dec  9 17:20:49 on ttys000
Armandos-MacBook-Pro:~ armandobenavidez$ cd Documents
Armandos-MacBook-Pro:Documents armandobenavidez$ cd localrepo157a
Armandos-MacBook-Pro:localrepo157a armandobenavidez$ cd Documents
-bash: cd: Documents: No such file or directory
Armandos-MacBook-Pro:localrepo157a armandobenavidez$ ls
CS157A-40
Armandos-MacBook-Pro:localrepo157a armandobenavidez$ cd cs157a-40
Armandos-MacBook-Pro:cs157a-40 armandobenavidez$ ls
Project Design ER DIAGRAM.pdf           README.md
Project Design ER Revision.pdf          Tables and Tuples.pdf
Project Proposal.pdf                    flynow
Project Requirement Document.pdf
Armandos-MacBook-Pro:cs157a-40 armandobenavidez$ cd flynow
Armandos-MacBook-Pro:flynow armandobenavidez$ cd backend;
Armandos-MacBook-Pro:backend armandobenavidez$ npm start

> backend@0.0.0 start /Users/armandobenavidez/Documents/LocalRepo157A/CS157A-40/flynow/backend
> node ./bin/www

Server running on port: 5000
Database is connected ... nn
```

7. Next you need to open another terminal/command window to go into the flynow directory and go into the 'src' folder. Once you are in the src folder then you need to type in 'npm start'. Almost identical as previous step except for this time you are going into src folder vs previously you went into backend folder. A successful start will result in the following screen below.

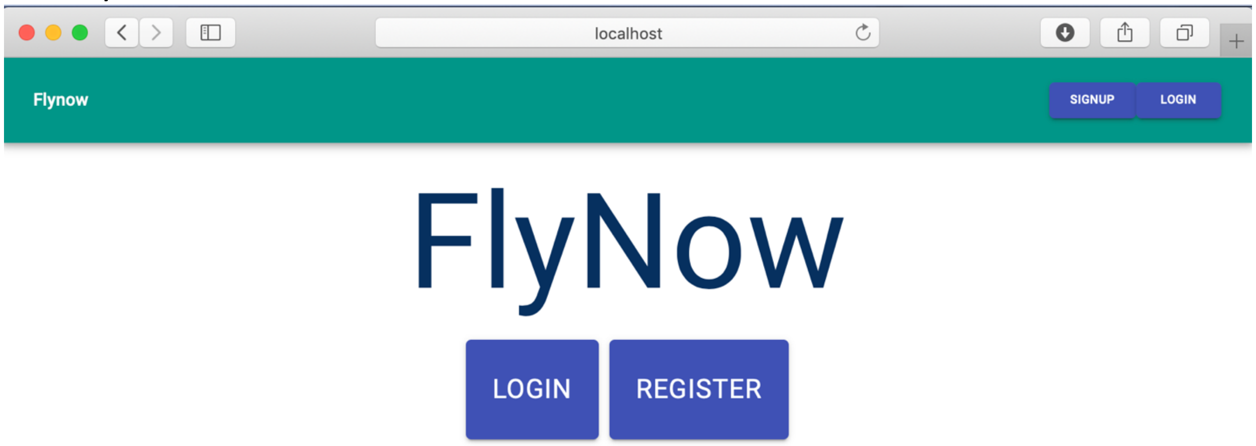

src — node ◂ npm TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash — 80×24

```
Compiled with warnings.

./src/components/signup.js
  Line 14:10:  'useHistory' is defined but never used  no-unused-vars

./src/components/login.js
  Line 1:27:   'useEffect' is defined but never used           no-unused-vars
  Line 14:10:  'useHistory' is defined but never used          no-unused-vars
  Line 16:20:  'Router' is defined but never used              no-unused-vars
  Line 17:3:   'Switch' is defined but never used              no-unused-vars
  Line 18:3:   'Route' is defined but never used               no-unused-vars
  Line 18:10:  'Redirect' is defined but never used            no-unused-vars
  Line 62:10:  'redirect' is assigned a value but never used   no-unused-vars
  Line 62:20:  'setRedirect' is assigned a value but never used no-unused-vars
  Line 64:25:  'register' is assigned a value but never used   no-unused-vars
  Line 68:23:  Expected '===' and instead saw '=='             eqeqeq

./src/routes.js
  Line 2:10:  'Route' is defined but never used  no-unused-vars

./src/App.js
  Line 2:27:   'Router' is defined but never used                    no-unus
ed-vars
  Line 5:10:   'withStyles' is defined but never used                no-unus
```

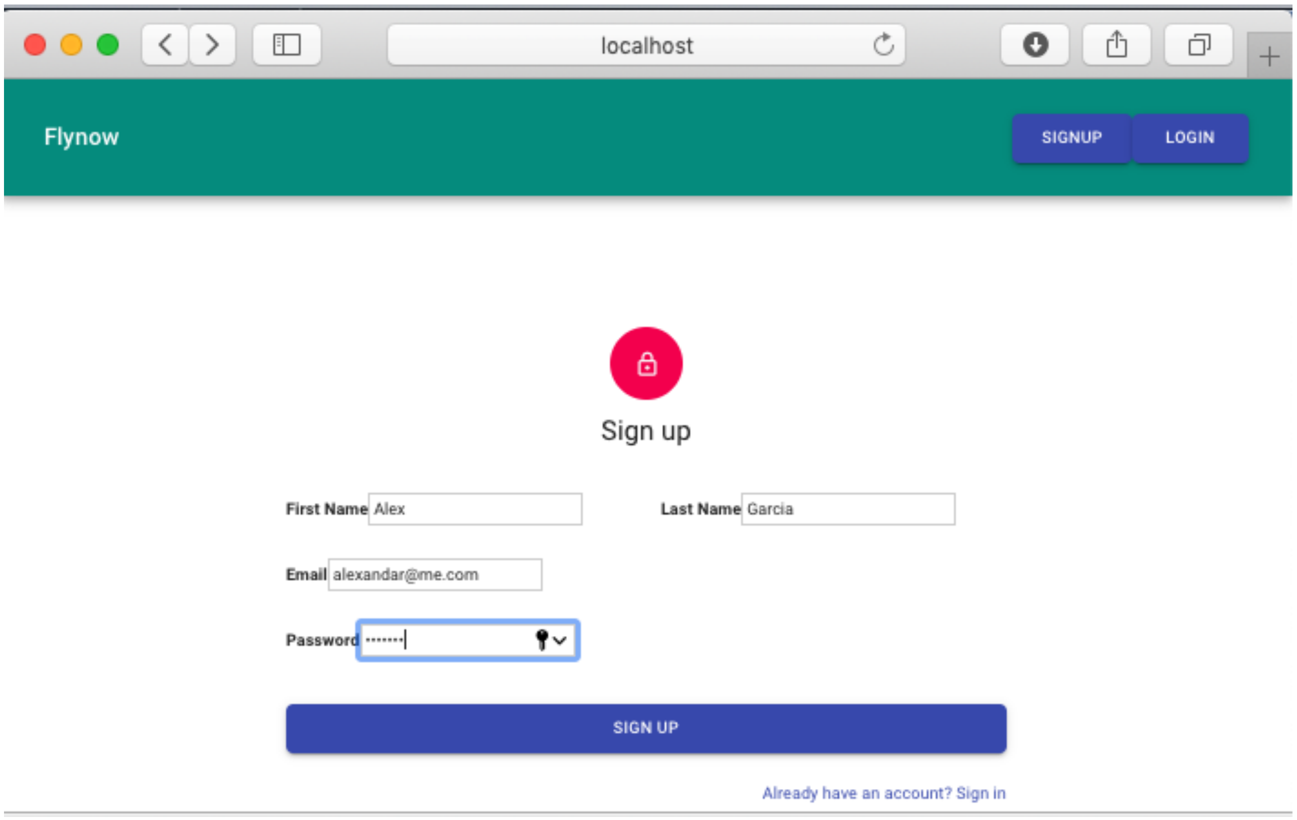
8. Once all these are started your web browser will be opened to the main landing page.

MAIN LANDING PAGE

9. When you are at the home page you can either Login to your dashboard or sign in to your current account.



10. If you click on SIGN UP, then you will be brought up to a sign-up dashboard where you can sign up with an account so you can get access to search for flights. You will need to enter your first name, last name, and email with a password to sign up.



First Name Alex

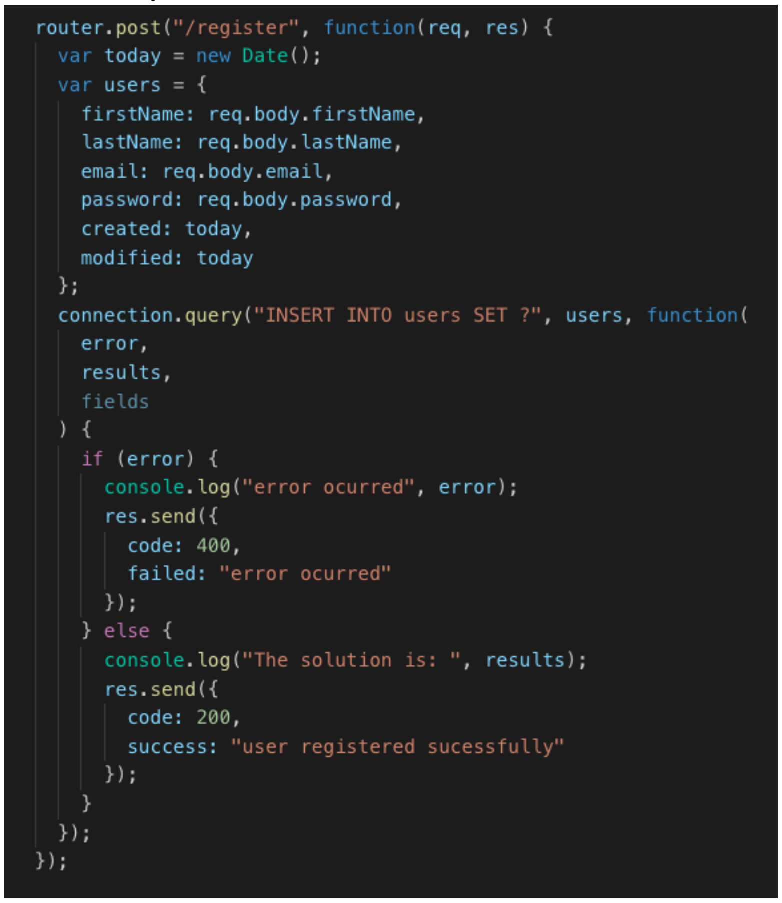
Last Name Garcia

Email alexandar@me.com

Password •••••••

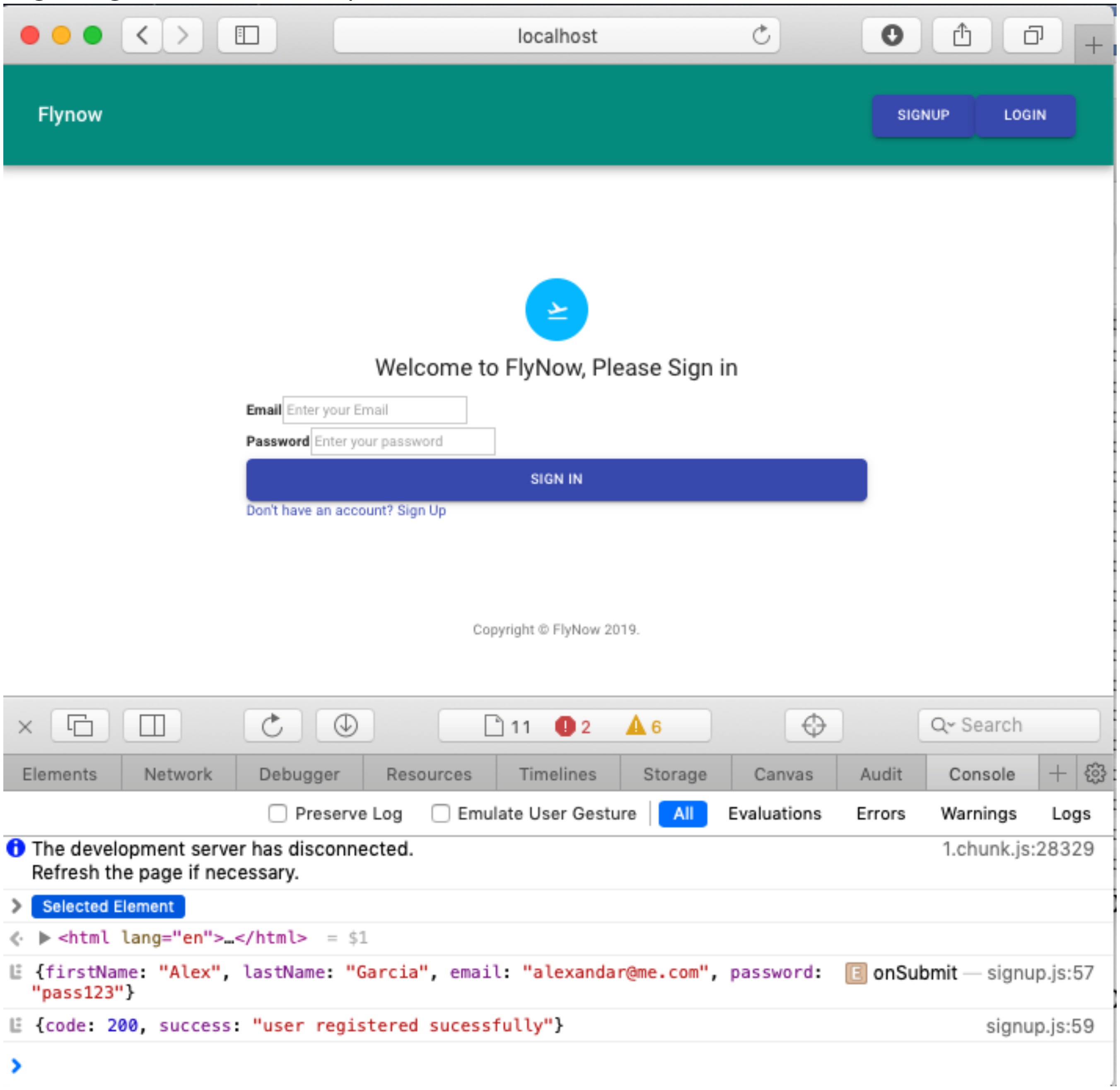
A successful sign up will return you the log in page as shown below. We can also see that the user has been successfully registered in the database.

SQL Query Code:



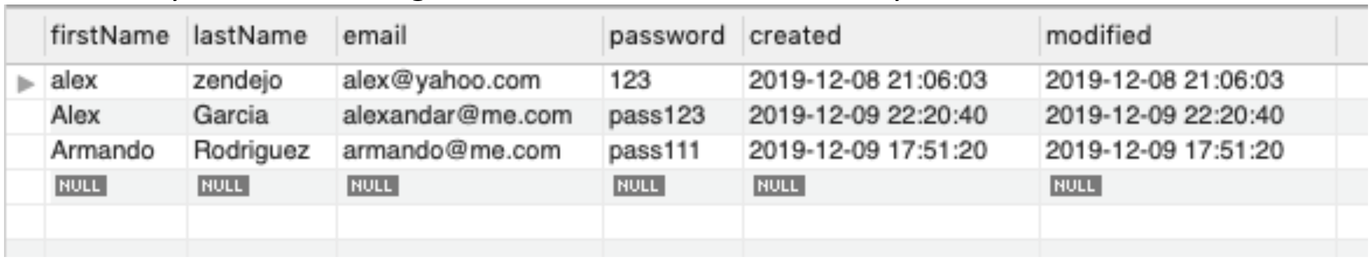
```
router.post("/register", function(req, res) {
  var today = new Date();
  var users = {
    firstName: req.body.firstName,
    lastName: req.body.lastName,
    email: req.body.email,
    password: req.body.password,
    created: today,
    modified: today
  };
  connection.query("INSERT INTO users SET ?", users, function(
    error,
    results,
    fields
  ) {
    if (error) {
      console.log("error ocurred", error);
      res.send({
        code: 400,
        failed: "error ocurred"
      });
    } else {
      console.log("The solution is: ", results);
      res.send({
        code: 200,
        success: "user registered sucessfully"
      });
    }
  });
});
```

Login Page with success response:



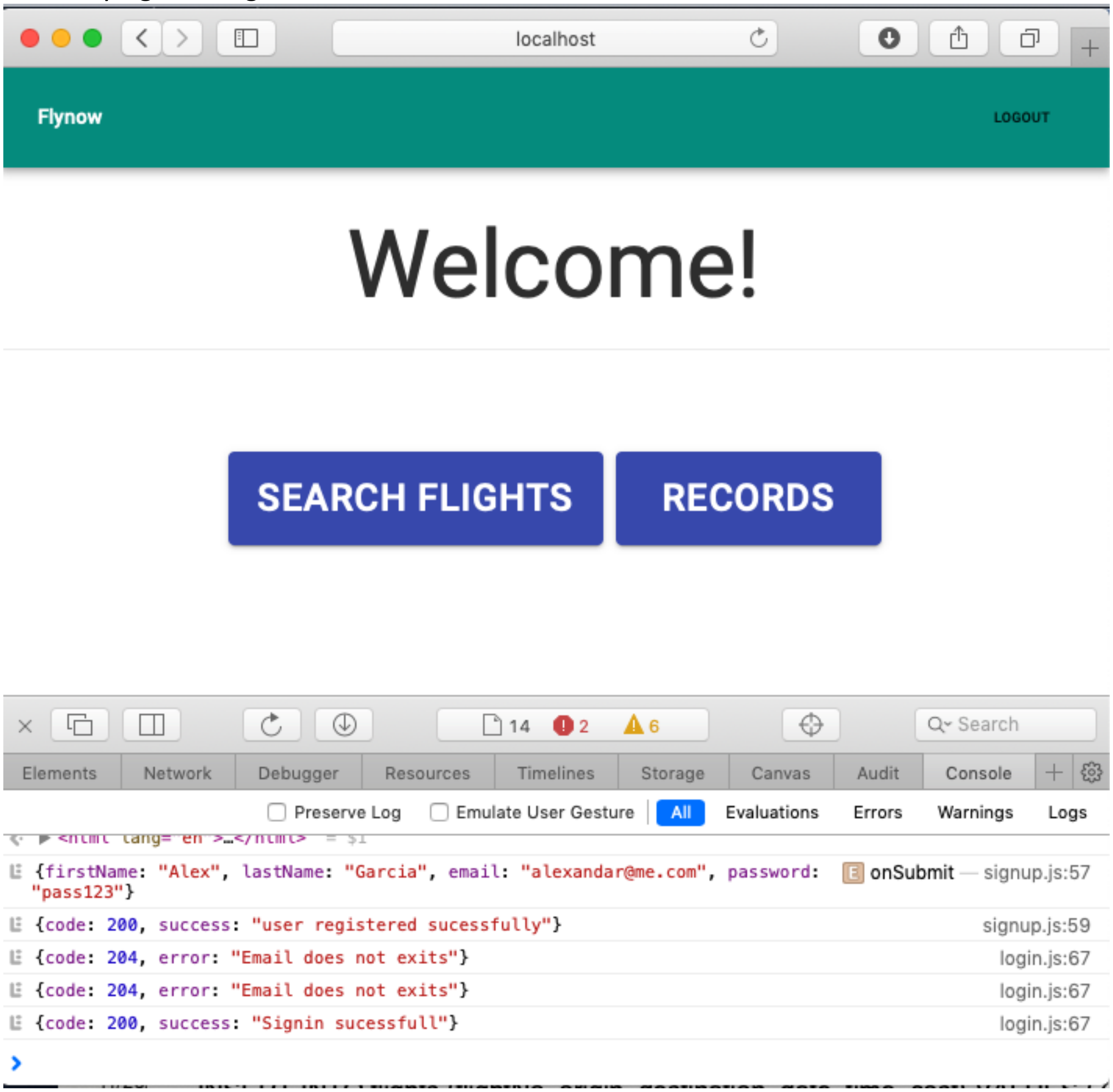
```
The development server has disconnected.                    1.chunk.js:28329
Refresh the page if necessary.
<html lang="en">…</html>  = $1
{firstName: "Alex", lastName: "Garcia", email: "alexandar@me.com", password:    onSubmit — signup.js:57
"pass123"}
{code: 200, success: "user registered sucessfully"}                             signup.js:59
```

Additionally, it has been registered in our user's table in MySQL as shown below:



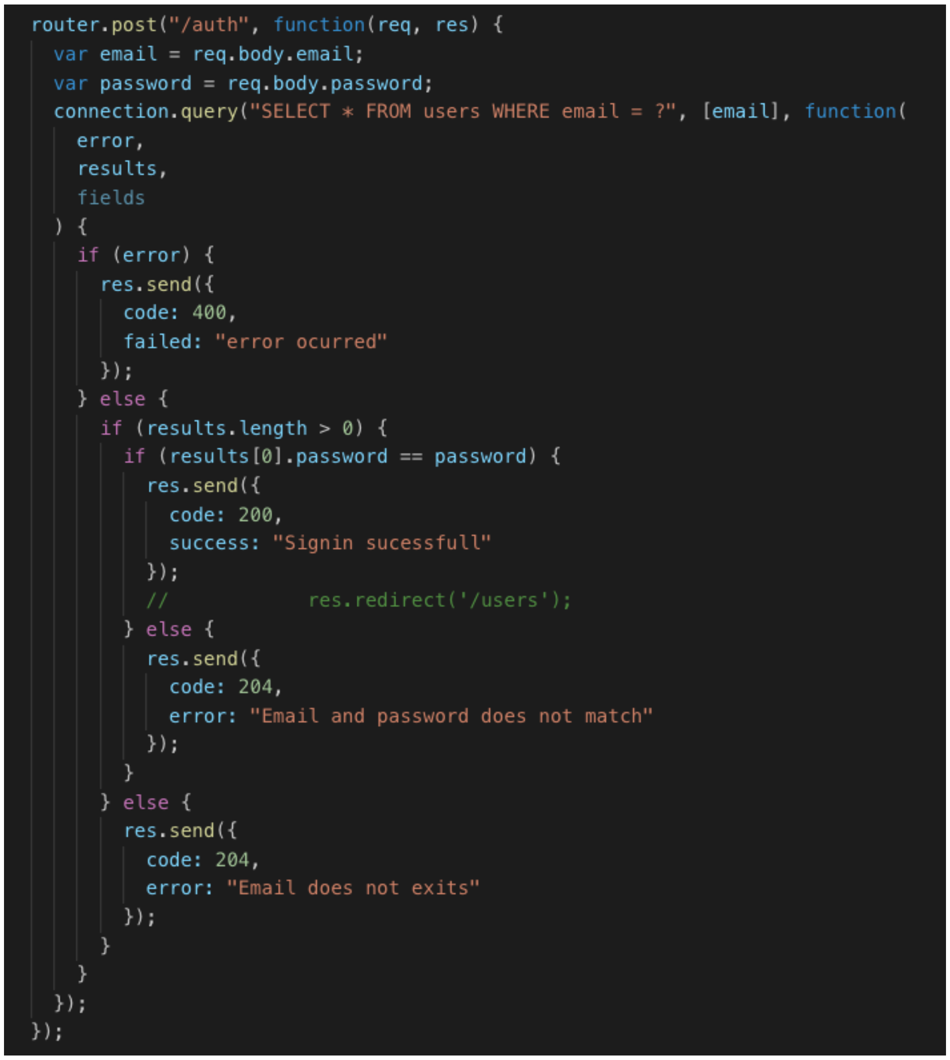
firstName	lastName	email	password	created	modified
alex	zendejo	alex@yahoo.com	123	2019-12-08 21:06:03	2019-12-08 21:06:03
Alex	Garcia	alexandar@me.com	pass123	2019-12-09 22:20:40	2019-12-09 22:20:40
Armando	Rodriguez	armando@me.com	pass111	2019-12-09 17:51:20	2019-12-09 17:51:20
NULL	NULL	NULL	NULL	NULL	NULL

11. Now you are the log in page, and you should be able to sign in with your credentials you just created previously with your account. If login is a success you will be brought to the next landing page which is a Welcome dashboard. If credentials don't match you will get a pop-up error saying "Wrong credentials".



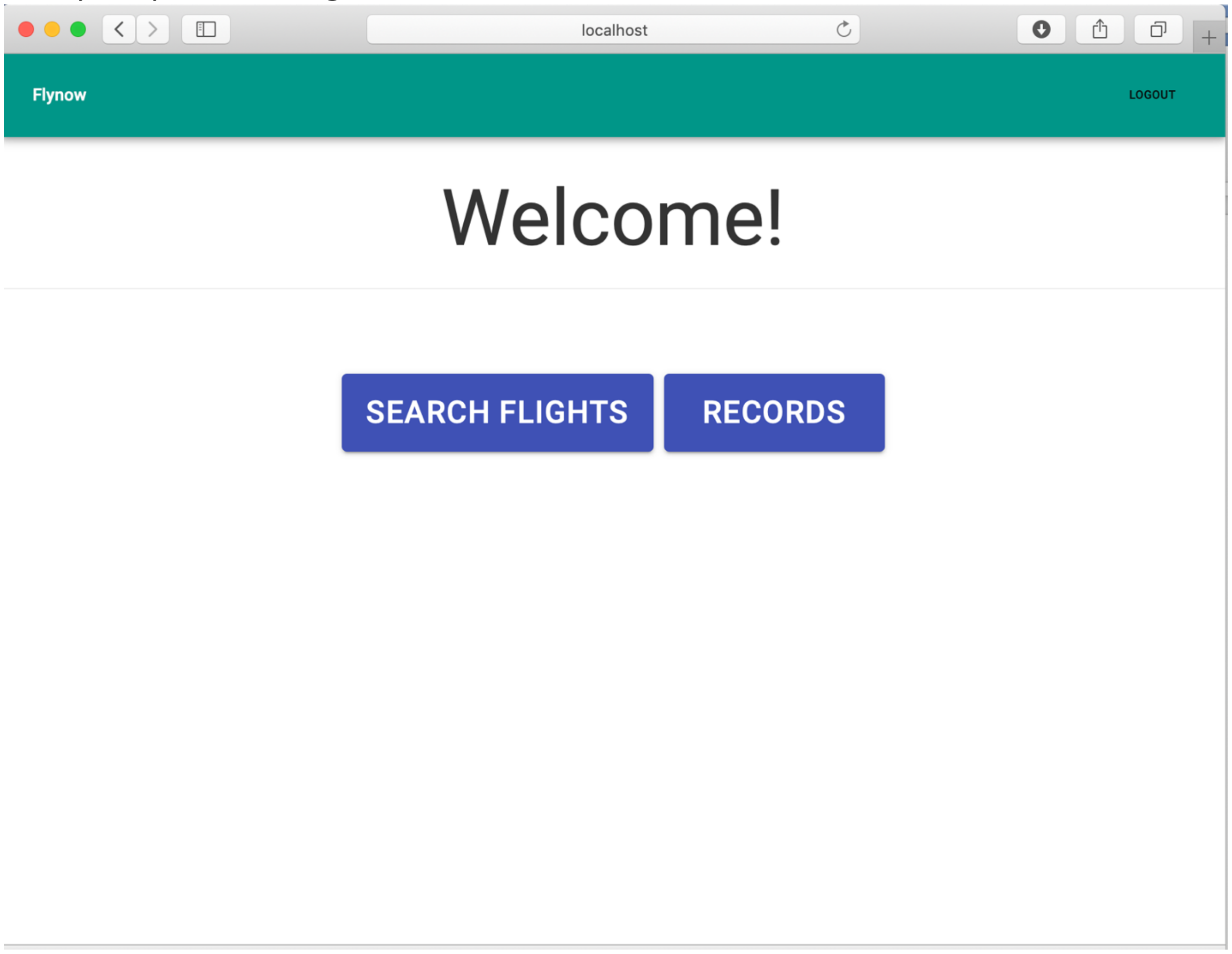
```
{firstName: "Alex", lastName: "Garcia", email: "alexandar@me.com", password:     onSubmit — signup.js:57
"pass123"}
{code: 200, success: "user registered sucessfully"}                              signup.js:59
{code: 204, error: "Email does not exits"}                                        login.js:67
{code: 204, error: "Email does not exits"}                                        login.js:67
{code: 200, success: "Signin sucessfull"}                                         login.js:67
```

Query code for Authorization:

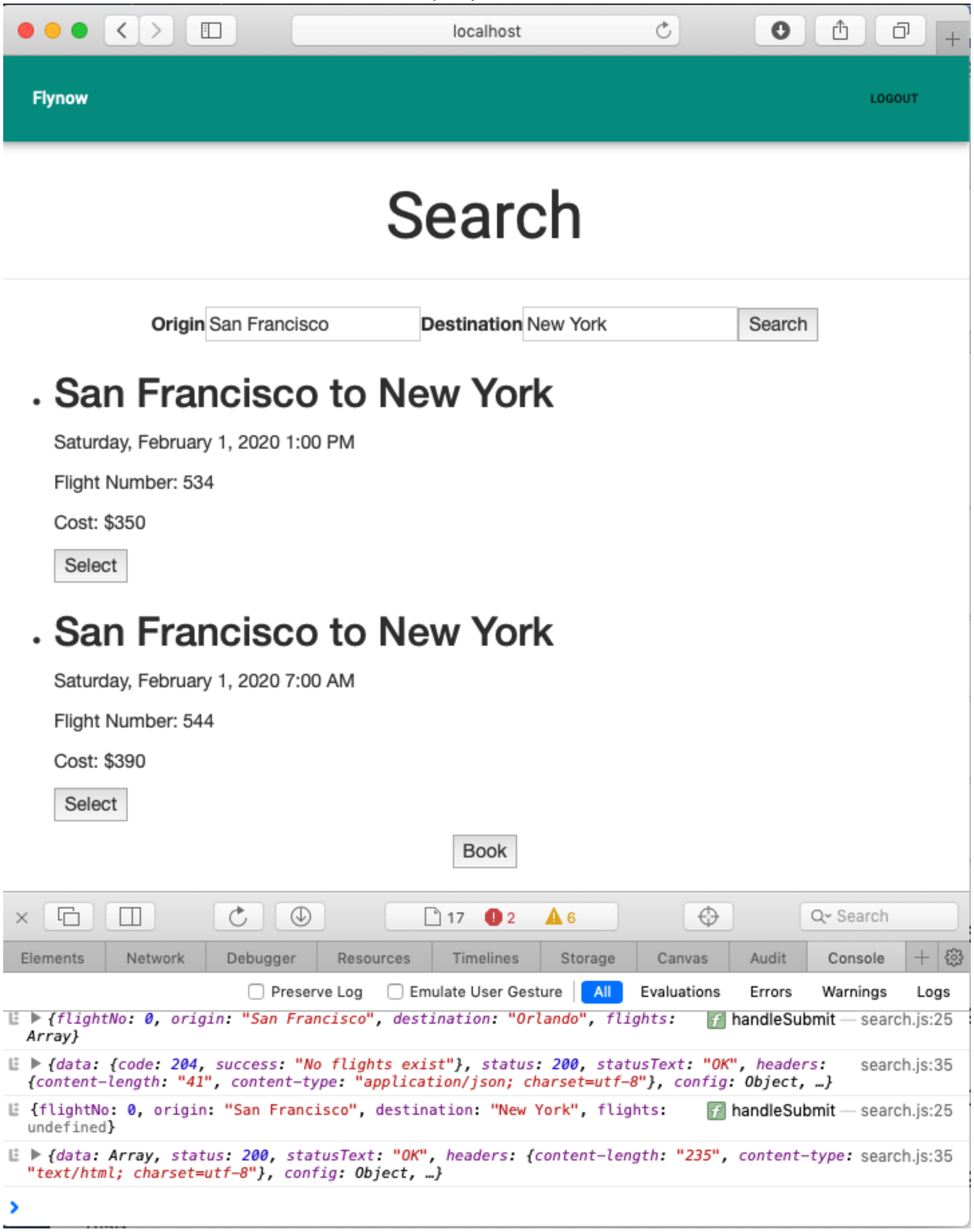


```
router.post("/auth", function(req, res) {
  var email = req.body.email;
  var password = req.body.password;
  connection.query("SELECT * FROM users WHERE email = ?", [email], function(
    error,
    results,
    fields
  ) {
    if (error) {
      res.send({
        code: 400,
        failed: "error ocurred"
      });
    } else {
      if (results.length > 0) {
        if (results[0].password == password) {
          res.send({
            code: 200,
            success: "Signin sucessfull"
          });
          //            res.redirect('/users');
        } else {
          res.send({
            code: 204,
            error: "Email and password does not match"
          });
        }
      } else {
        res.send({
          code: 204,
          error: "Email does not exits"
        });
      }
    }
  });
});
```

12. From the welcome dashboard you have two options you can either search for a flight or view your past bookings.



13. If you select search flights you will be brought to the search page where you will be able to search for a flight from a particular destination. Let's say we want to go from San Francisco to New York then these results will display.



Search

Origin San Francisco Destination New York Search

- San Francisco to New York

  Saturday, February 1, 2020 1:00 PM

  Flight Number: 534

  Cost: $350

  Select

- San Francisco to New York

  Saturday, February 1, 2020 7:00 AM

  Flight Number: 544

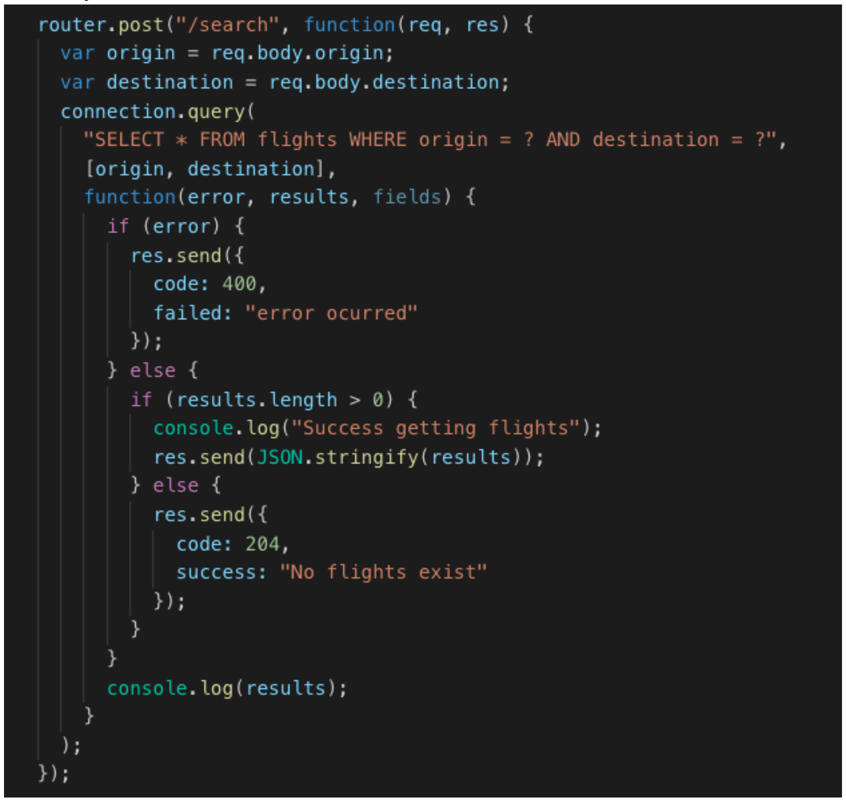
  Cost: $390

  Select

Book

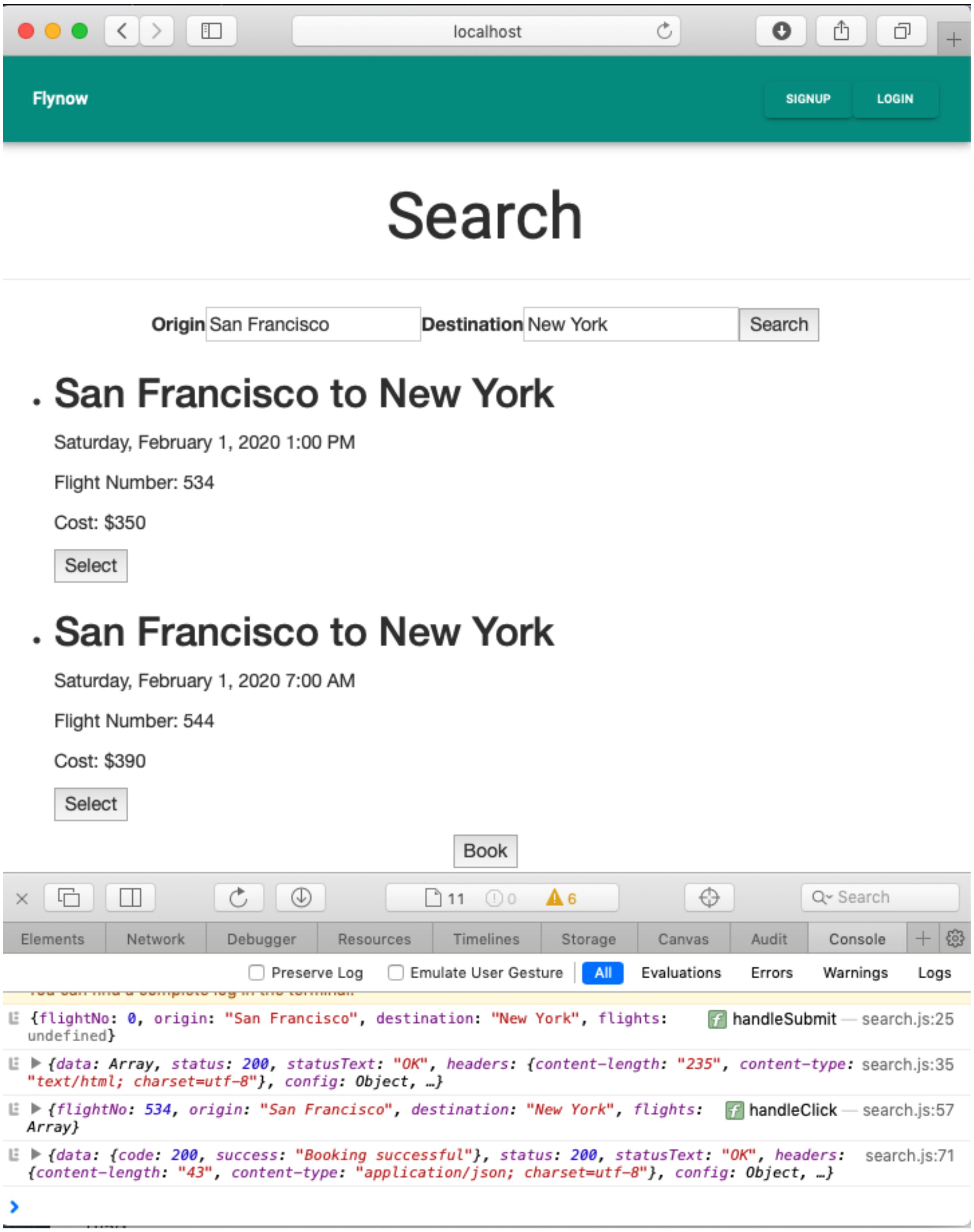
```
▶ {flightNo: 0, origin: "San Francisco", destination: "Orlando", flights:    handleSubmit — search.js:25
Array}
▶ {data: {code: 204, success: "No flights exist"}, status: 200, statusText: "OK", headers:    search.js:35
{content-length: "41", content-type: "application/json; charset=utf-8"}, config: Object, …}
{flightNo: 0, origin: "San Francisco", destination: "New York", flights:    handleSubmit — search.js:25
undefined}
▶ {data: Array, status: 200, statusText: "OK", headers: {content-length: "235", content-type: search.js:35
"text/html; charset=utf-8"}, config: Object, …}
>
```

Query code for search:



```
router.post("/search", function(req, res) {
  var origin = req.body.origin;
  var destination = req.body.destination;
  connection.query(
    "SELECT * FROM flights WHERE origin = ? AND destination = ?",
    [origin, destination],
    function(error, results, fields) {
      if (error) {
        res.send({
          code: 400,
          failed: "error ocurred"
        });
      } else {
        if (results.length > 0) {
          console.log("Success getting flights");
          res.send(JSON.stringify(results));
        } else {
          res.send({
            code: 204,
            success: "No flights exist"
          });
        }
      }
      console.log(results);
    }
  );
});
```

14. If you want to book this flight then you have to click select and then book and a message will appear saying booking successful and are brought back to your dashboard. If you want to see your booking, then you click on records from your home dashboard and should appear there.



Search

Origin San Francisco **Destination** New York Search

- San Francisco to New York

  Saturday, February 1, 2020 1:00 PM

  Flight Number: 534

  Cost: $350

  Select

- San Francisco to New York

  Saturday, February 1, 2020 7:00 AM

  Flight Number: 544

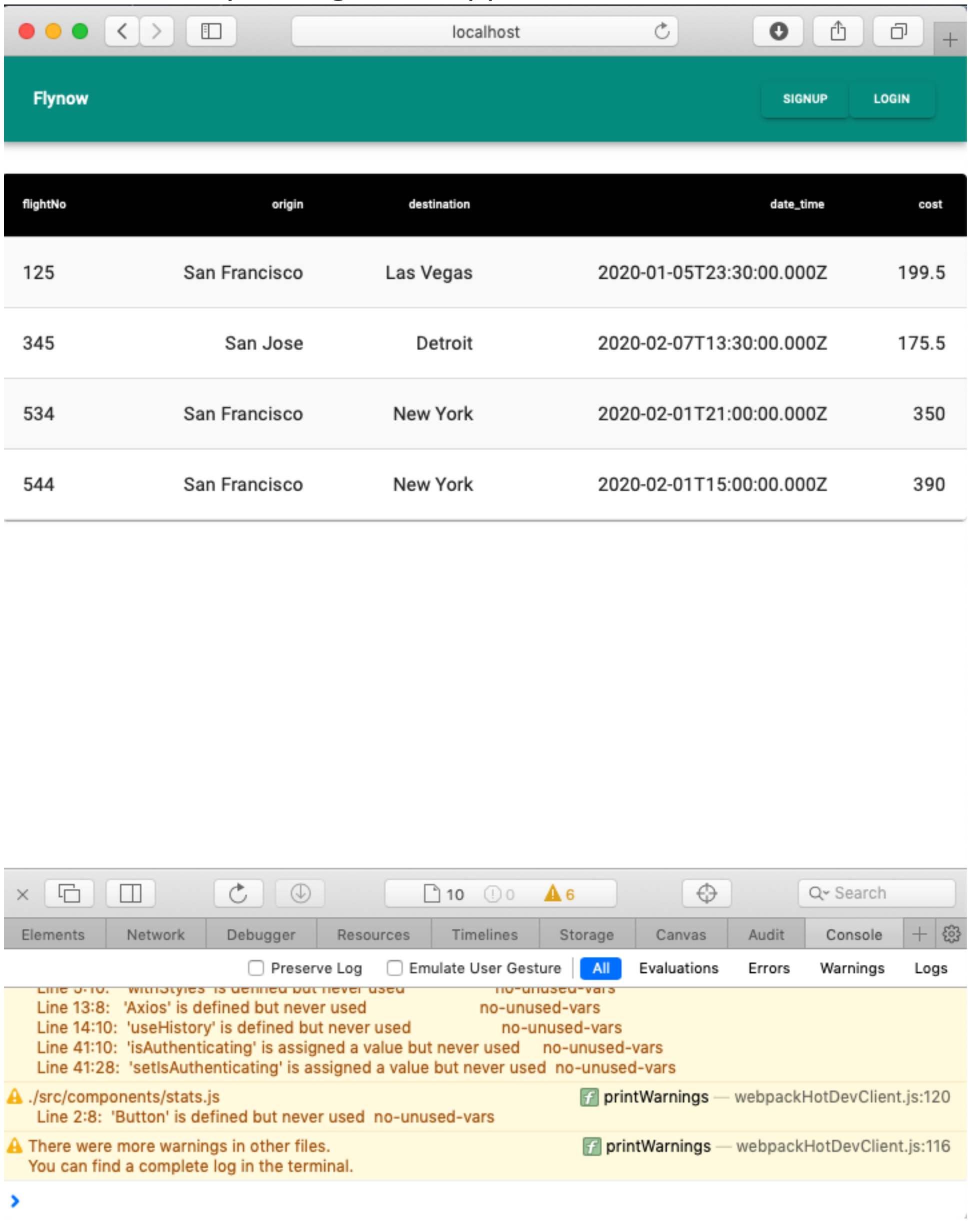
  Cost: $390

  Select

Book

```
{flightNo: 0, origin: "San Francisco", destination: "New York", flights:      handleSubmit — search.js:25
undefined}
▶ {data: Array, status: 200, statusText: "OK", headers: {content-length: "235", content-type: search.js:35
"text/html; charset=utf-8"}, config: Object, …}
▶ {flightNo: 534, origin: "San Francisco", destination: "New York", flights:  handleClick — search.js:57
Array}
▶ {data: {code: 200, success: "Booking successful"}, status: 200, statusText: "OK", headers:   search.js:71
{content-length: "43", content-type: "application/json; charset=utf-8"}, config: Object, …}
```

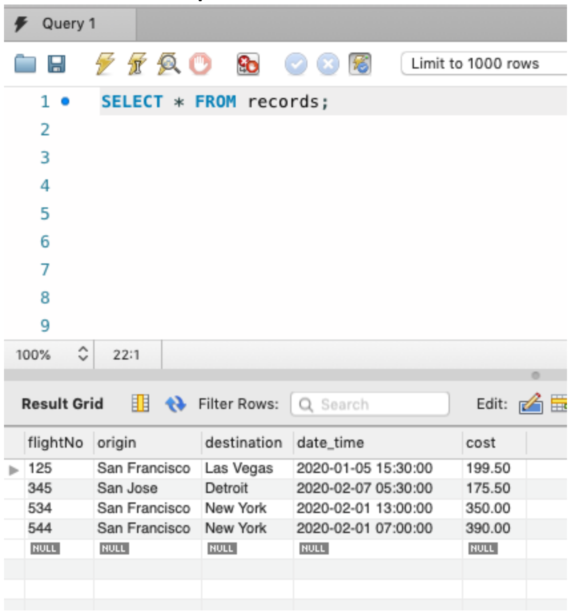
As we can see your flight has appeared in the records view:



flightNo	origin	destination	date_time	cost
125	San Francisco	Las Vegas	2020-01-05T23:30:00.000Z	199.5
345	San Jose	Detroit	2020-02-07T13:30:00.000Z	175.5
534	San Francisco	New York	2020-02-01T21:00:00.000Z	350
544	San Francisco	New York	2020-02-01T15:00:00.000Z	390

```
  Line 13:8:  'Axios' is defined but never used             no-unused-vars
  Line 14:10:  'useHistory' is defined but never used          no-unused-vars
  Line 41:10:  'isAuthenticating' is assigned a value but never used    no-unused-vars
  Line 41:28:  'setIsAuthenticating' is assigned a value but never used  no-unused-vars
./src/components/stats.js                                printWarnings — webpackHotDevClient.js:120
  Line 2:8:  'Button' is defined but never used  no-unused-vars
There were more warnings in other files.                 printWarnings — webpackHotDevClient.js:116
You can find a complete log in the terminal.
```

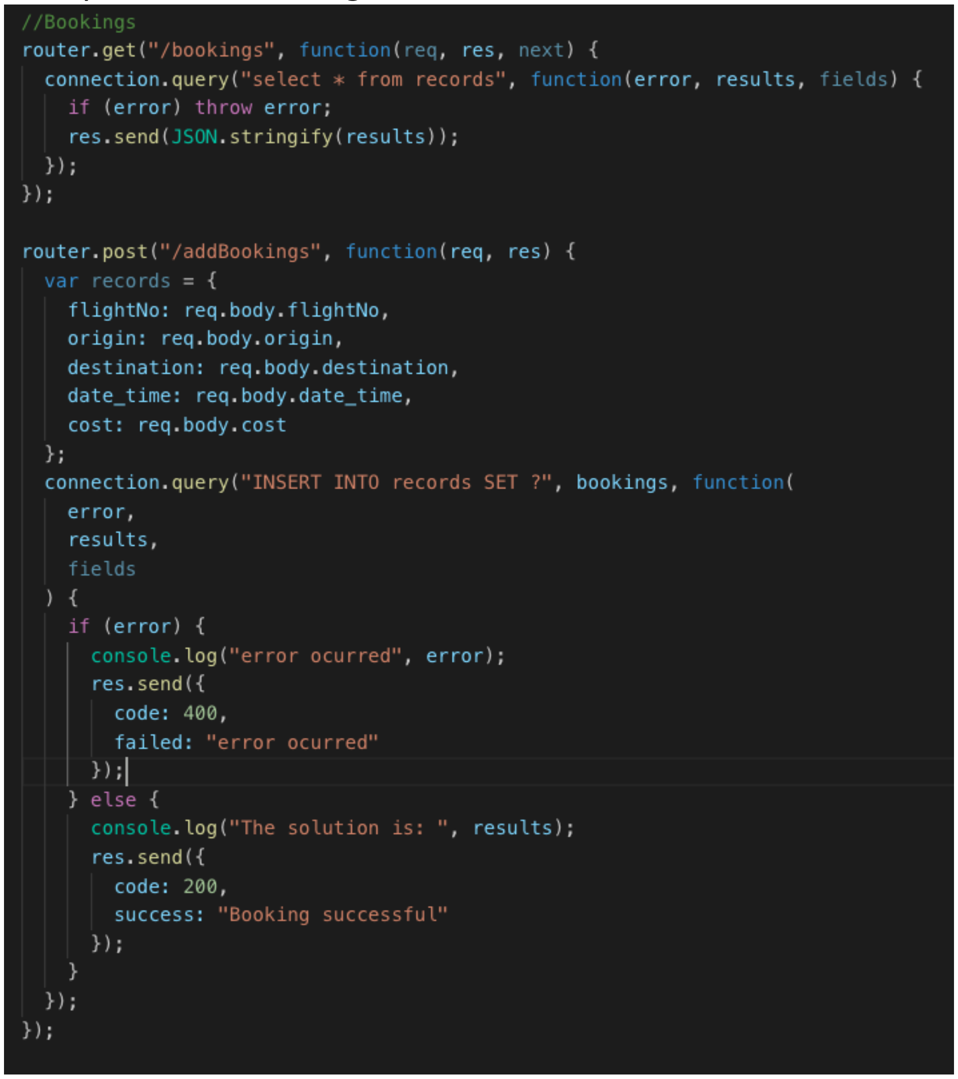
Here is the updated records table with that booked flight:



```
SELECT * FROM records;
```

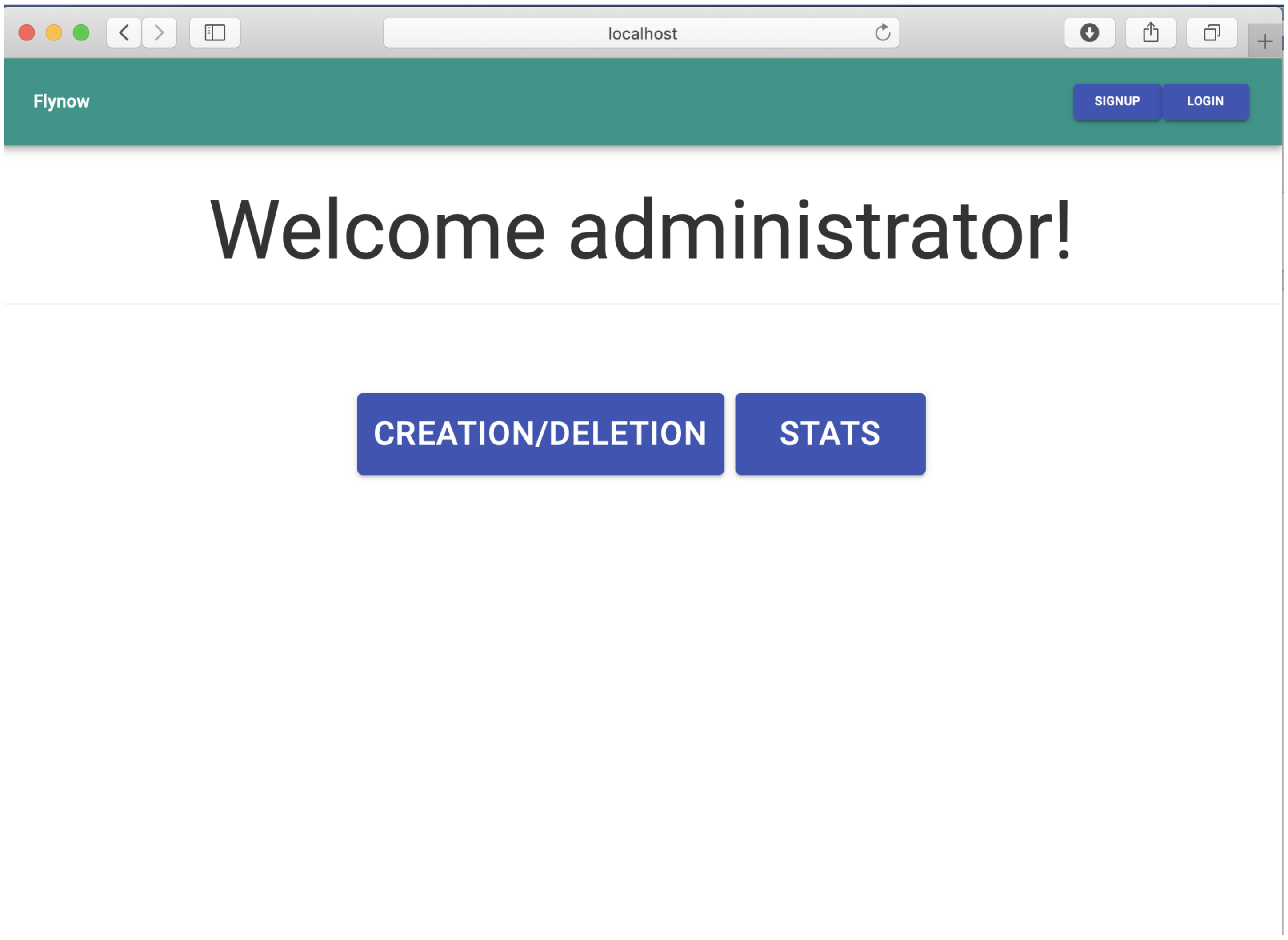
flightNo	origin	destination	date_time	cost
125	San Francisco	Las Vegas	2020-01-05 15:30:00	199.50
345	San Jose	Detroit	2020-02-07 05:30:00	175.50
534	San Francisco	New York	2020-02-01 13:00:00	350.00
544	San Francisco	New York	2020-02-01 07:00:00	390.00
NULL	NULL	NULL	NULL	NULL

Query code for bookings:



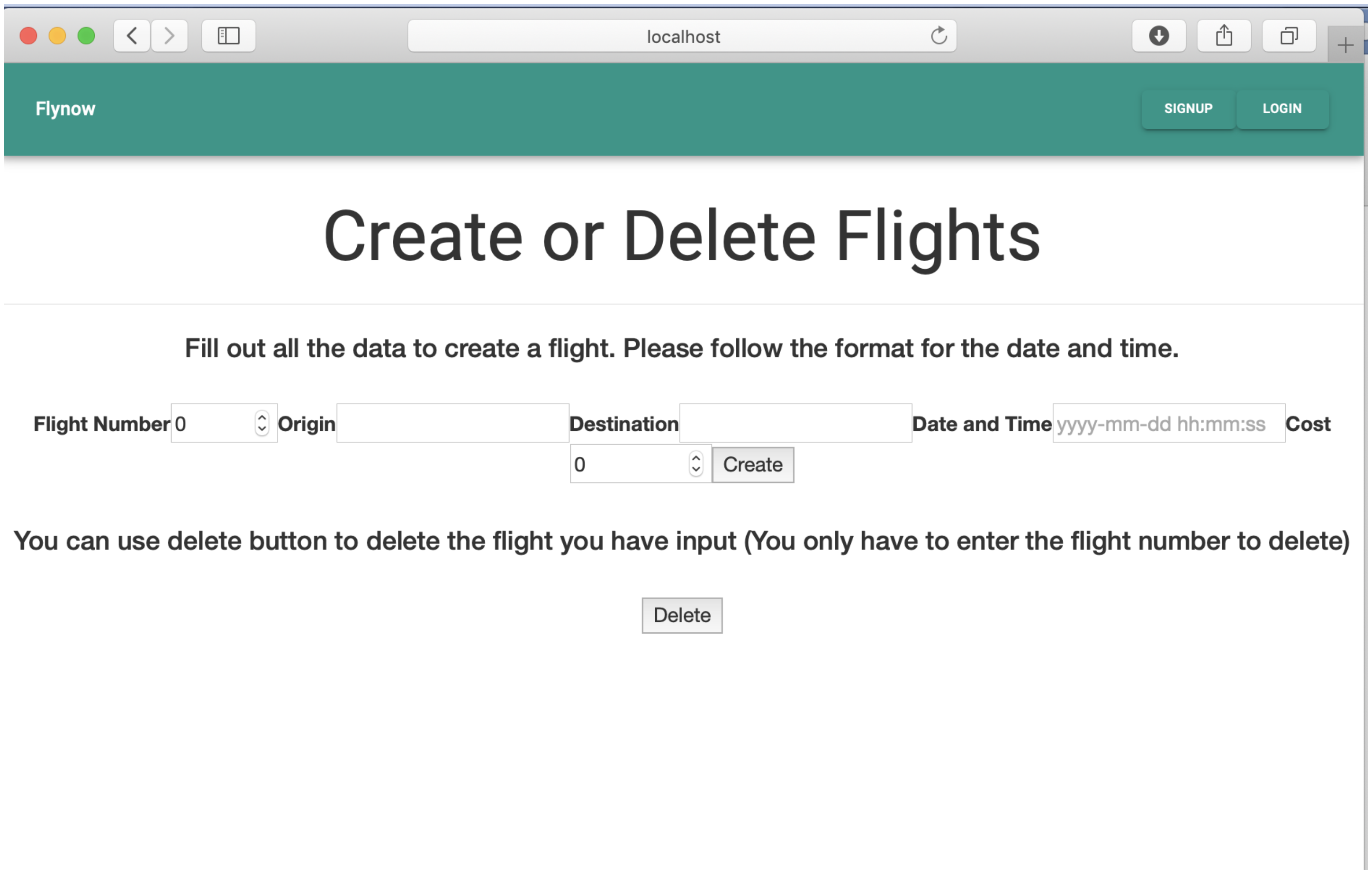
```
//Bookings
router.get("/bookings", function(req, res, next) {
  connection.query("select * from records", function(error, results, fields) {
    if (error) throw error;
    res.send(JSON.stringify(results));
  });
});

router.post("/addBookings", function(req, res) {
  var records = {
    flightNo: req.body.flightNo,
    origin: req.body.origin,
    destination: req.body.destination,
    date_time: req.body.date_time,
    cost: req.body.cost
  };
  connection.query("INSERT INTO records SET ?", bookings, function(
    error,
    results,
    fields
  ) {
    if (error) {
      console.log("error ocurred", error);
      res.send({
        code: 400,
        failed: "error ocurred"
      });
    } else {
      console.log("The solution is: ", results);
      res.send({
        code: 200,
        success: "Booking successful"
      });
    }
  });
});
```

15. When you sign as an administrator then this is your landing page.



Welcome administrator!

16. As an administrator you have the option to create or delete flights or view the capacity/availability of each flight. If you select CREATION/DELETION, then you will be brought to this landing page.



Create or Delete Flights

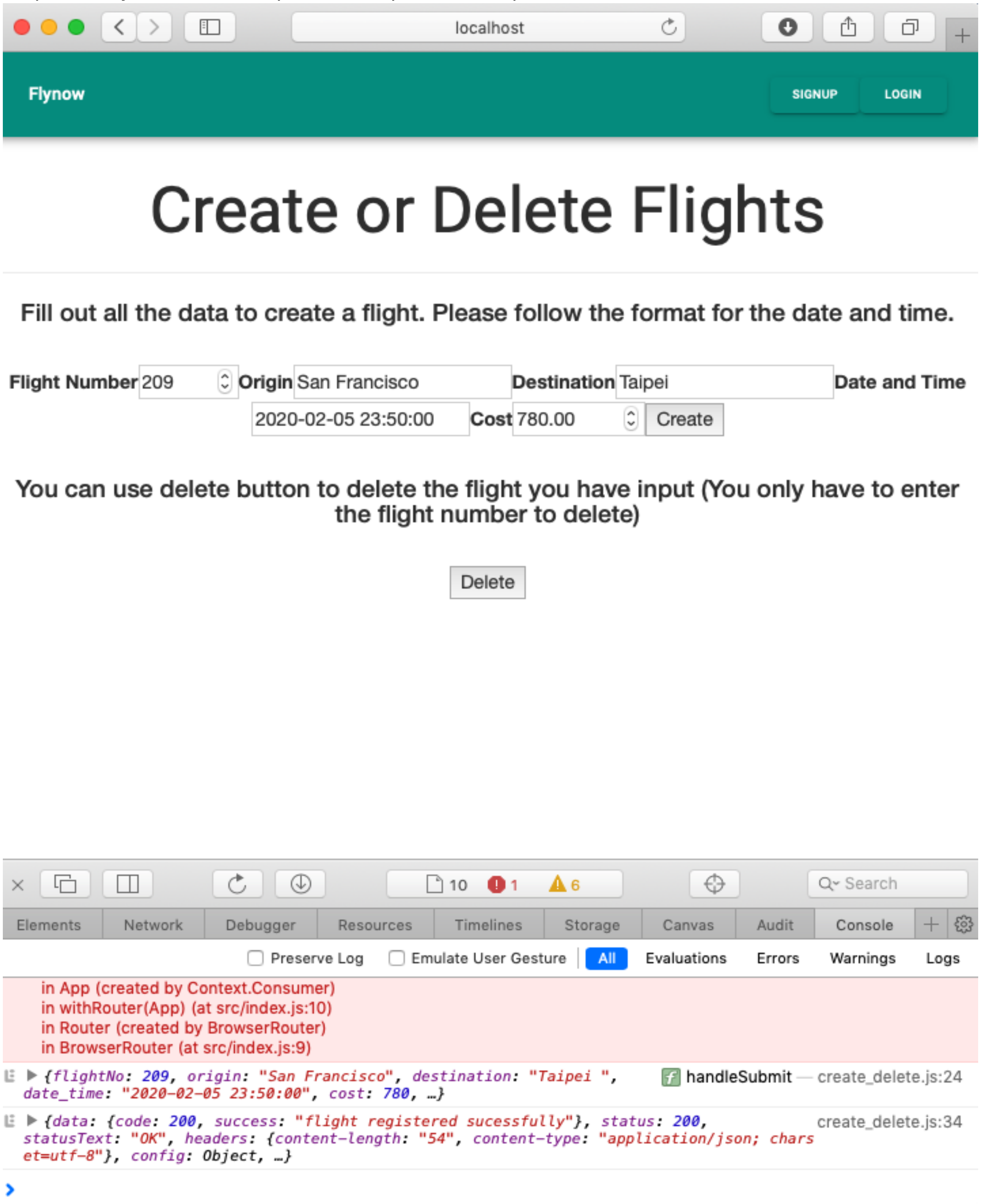
Fill out all the data to create a flight. Please follow the format for the date and time.

Flight Number 0 Origin Destination Date and Time yyyy-mm-dd hh:mm:ss Cost 0 Create

You can use delete button to delete the flight you have input (You only have to enter the flight number to delete)

Delete

17. From here you can create a flight let's say we want to create a flight from San Francisco to Taipei. We just need to input the requirements presented to us on the screen.



Create or Delete Flights

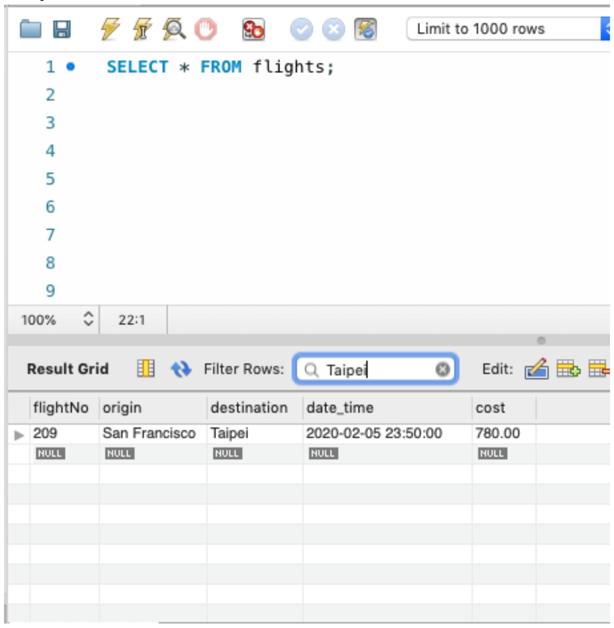
Fill out all the data to create a flight. Please follow the format for the date and time.

Flight Number: 209
Origin: San Francisco
Destination: Taipei
Date and Time: 2020-02-05 23:50:00
Cost: 780.00

You can use delete button to delete the flight you have input (You only have to enter the flight number to delete)

```
    in App (created by Context.Consumer)
    in withRouter(App) (at src/index.js:10)
    in Router (created by BrowserRouter)
    in BrowserRouter (at src/index.js:9)
▶ {flightNo: 209, origin: "San Francisco", destination: "Taipei ",        handleSubmit — create_delete.js:24
date_time: "2020-02-05 23:50:00", cost: 780, …}
▶ {data: {code: 200, success: "flight registered sucessfully"}, status: 200,         create_delete.js:34
statusText: "OK", headers: {content-length: "54", content-type: "application/json; chars
et=utf-8"}, config: Object, …}
>
```

MySQL workbench: As we can see it shows up in our flights table.



Limit to 1000 rows

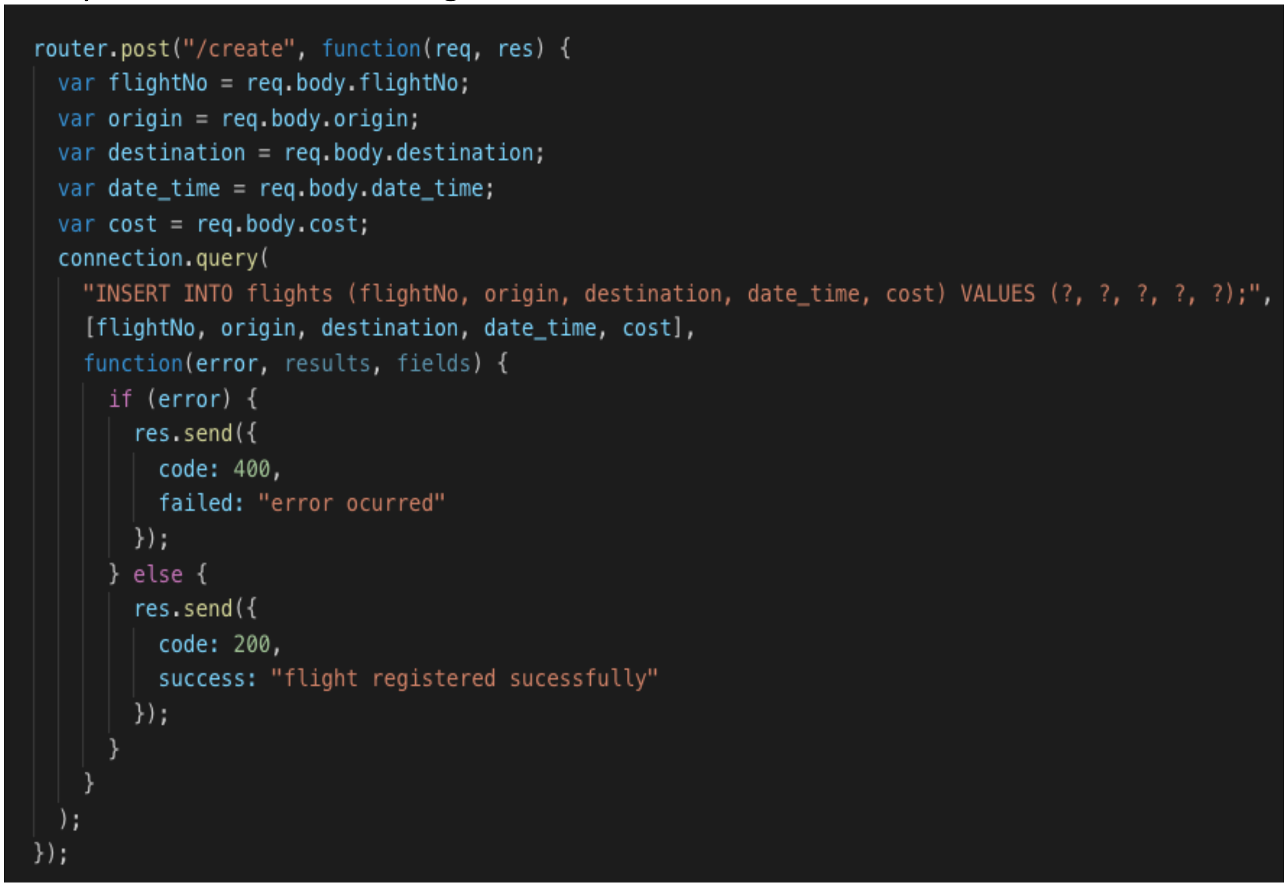
```sql
SELECT * FROM flights;
```

100% 22:1

Result Grid Filter Rows: Taipei Edit:

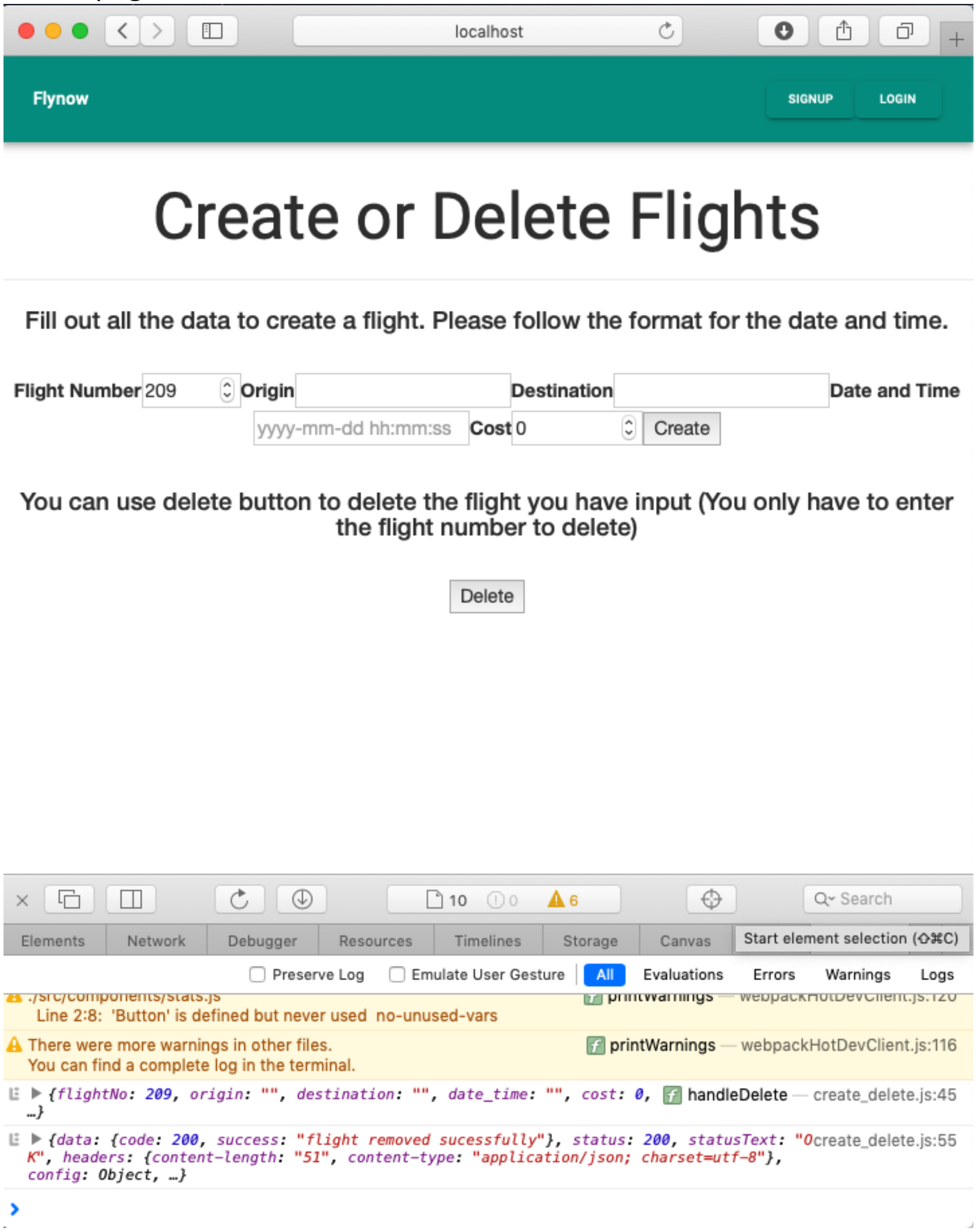
flightNo	origin	destination	date_time	cost
209	San Francisco	Taipei	2020-02-05 23:50:00	780.00
NULL	NULL	NULL	NULL	NULL

Query code for creation of flights:



```javascript
router.post("/create", function(req, res) {
  var flightNo = req.body.flightNo;
  var origin = req.body.origin;
  var destination = req.body.destination;
  var date_time = req.body.date_time;
  var cost = req.body.cost;
  connection.query(
    "INSERT INTO flights (flightNo, origin, destination, date_time, cost) VALUES (?, ?, ?, ?, ?);",
    [flightNo, origin, destination, date_time, cost],
    function(error, results, fields) {
      if (error) {
        res.send({
          code: 400,
          failed: "error ocurred"
        });
      } else {
        res.send({
          code: 200,
          success: "flight registered sucessfully"
        });
      }
    }
  );
});
```

18. To delete any flight only one entry is needed; the flight number. Let's say we want to delete our most recent flight we created. Then we just enter flight number 209 in our create or delete page and click on delete.


localhost
Flynow
SIGNUP
LOGIN

Create or Delete Flights

Fill out all the data to create a flight. Please follow the format for the date and time.

Flight Number 209 Origin Destination Date and Time yyyy-mm-dd hh:mm:ss Cost 0 Create

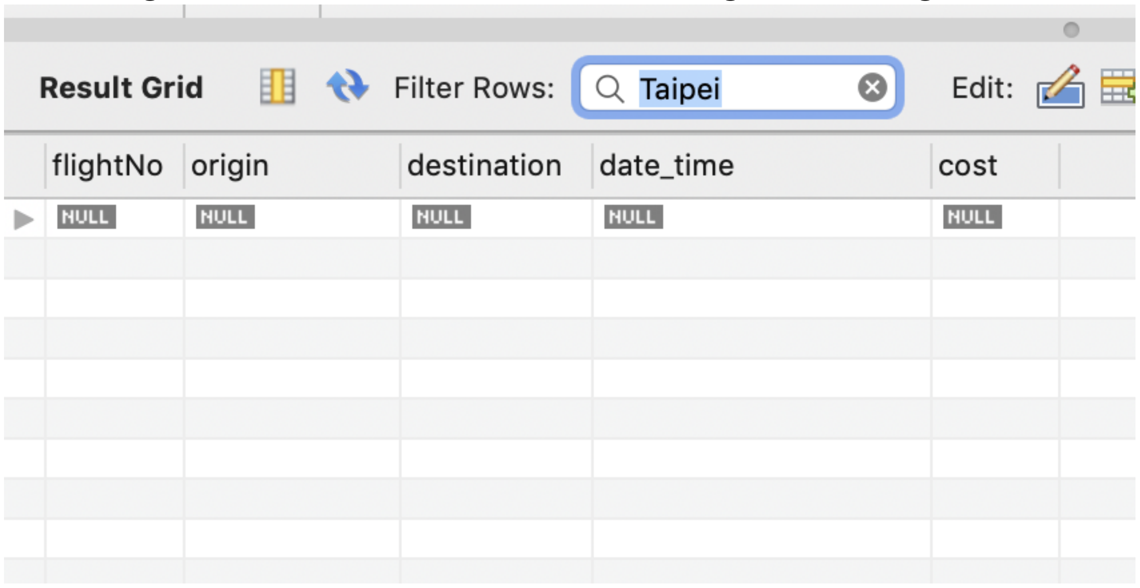
You can use delete button to delete the flight you have input (You only have to enter the flight number to delete)

Delete

10 0 6
Search
Elements Network Debugger Resources Timelines Storage Canvas
Start element selection (⇧⌘C)
Preserve Log
Emulate User Gesture
All Evaluations Errors Warnings Logs

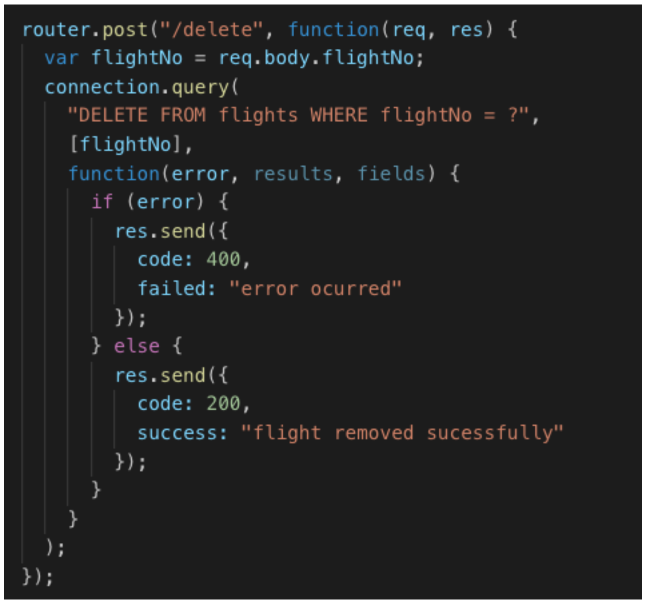
```
./src/components/stats.js                                        printWarnings — webpackHotDevClient.js:120
  Line 2:8:  'Button' is defined but never used  no-unused-vars
There were more warnings in other files.                         printWarnings — webpackHotDevClient.js:116
You can find a complete log in the terminal.
▶ {flightNo: 209, origin: "", destination: "", date_time: "", cost: 0,   handleDelete — create_delete.js:45
…}
▶ {data: {code: 200, success: "flight removed sucessfully"}, status: 200, statusText: "Ocreate_delete.js:55
K", headers: {content-length: "51", content-type: "application/json; charset=utf-8"},
config: Object, …}
```

Then we go back to our table to and see that is gone from flights table and it is gone.



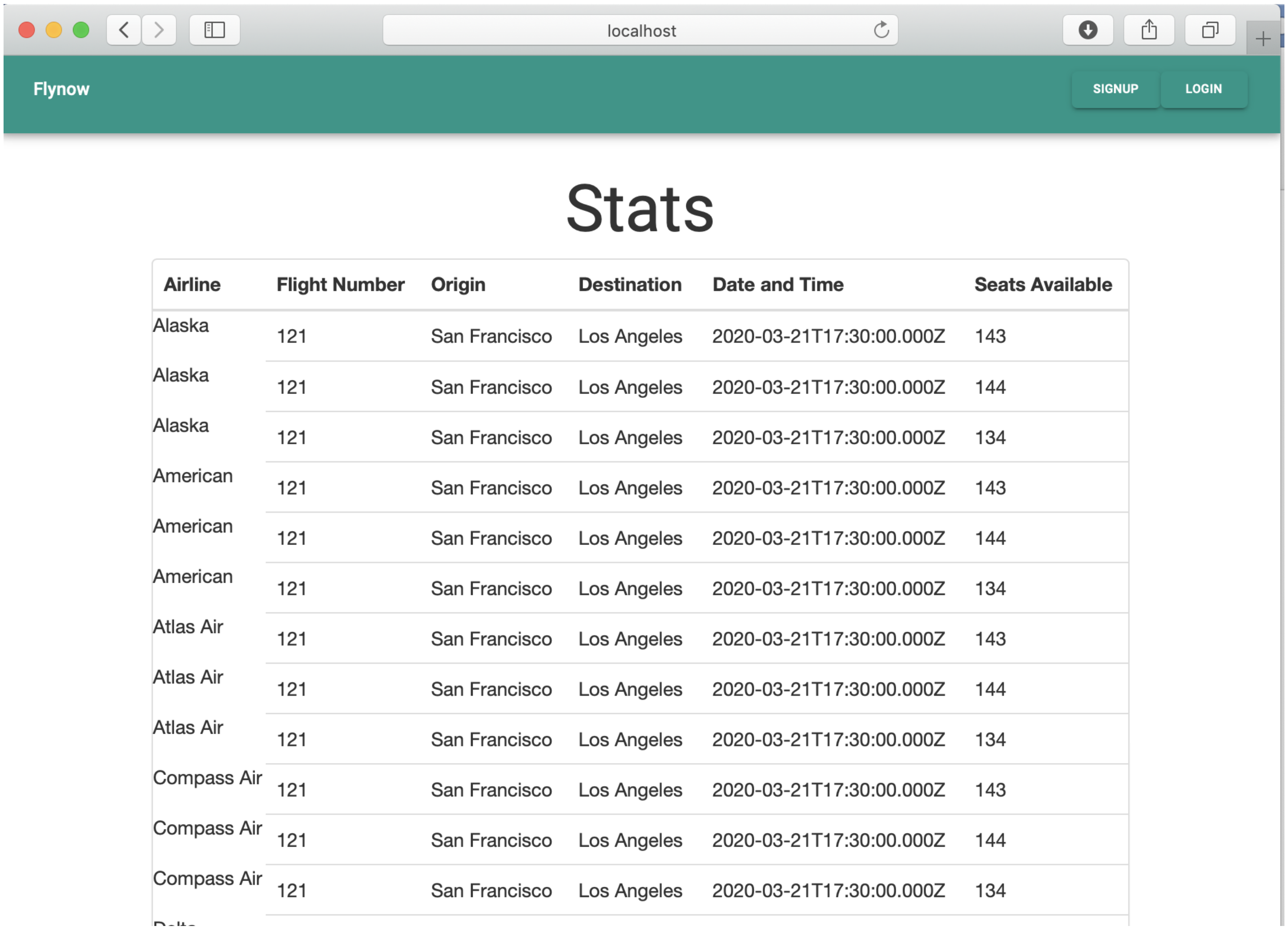
flightNo	origin	destination	date_time	cost
NULL	NULL	NULL	NULL	NULL

Query code for Deletion:



```
router.post("/delete", function(req, res) {
  var flightNo = req.body.flightNo;
  connection.query(
    "DELETE FROM flights WHERE flightNo = ?",
    [flightNo],
    function(error, results, fields) {
      if (error) {
        res.send({
          code: 400,
          failed: "error ocurred"
        });
      } else {
        res.send({
          code: 200,
          success: "flight removed sucessfully"
        });
      }
    }
  );
});
```

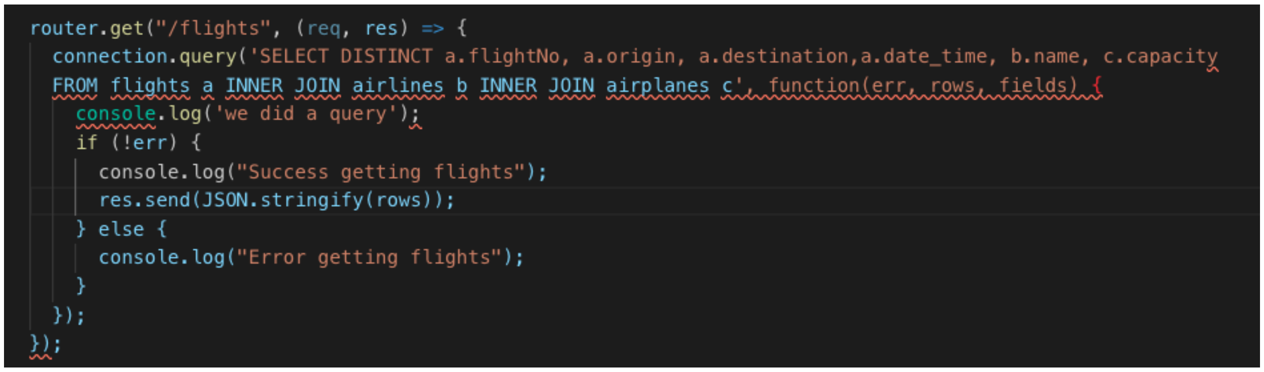
19. If you go back on the admin dashboard then you can see that you have stats page you can click on where it tells you all the flights availability.



Stats

Airline	Flight Number	Origin	Destination	Date and Time	Seats Available
Alaska	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	143
Alaska	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	144
Alaska	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	134
American	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	143
American	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	144
American	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	134
Atlas Air	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	143
Atlas Air	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	144
Atlas Air	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	134
Compass Air	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	143
Compass Air	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	144
Compass Air	121	San Francisco	Los Angeles	2020-03-21T17:30:00.000Z	134

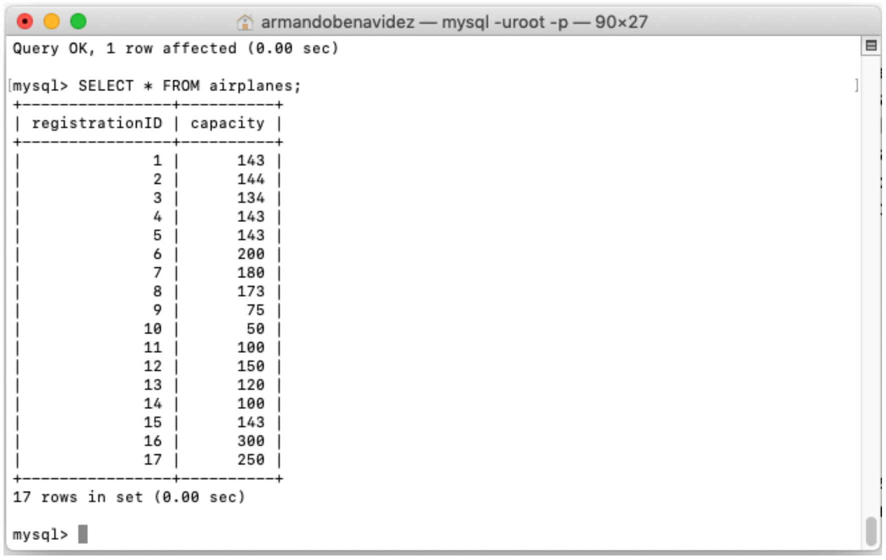
Query code for stats (moved down to fit into screenshot because it was long; that's why its underlined):



```
router.get("/flights", (req, res) => {
  connection.query('SELECT DISTINCT a.flightNo, a.origin, a.destination,a.date_time, b.name, c.capacity
  FROM flights a INNER JOIN airlines b INNER JOIN airplanes c', function(err, rows, fields) {
    console.log('we did a query');
    if (!err) {
      console.log("Success getting flights");
      res.send(JSON.stringify(rows));
    } else {
      console.log("Error getting flights");
    }
  });
});
```

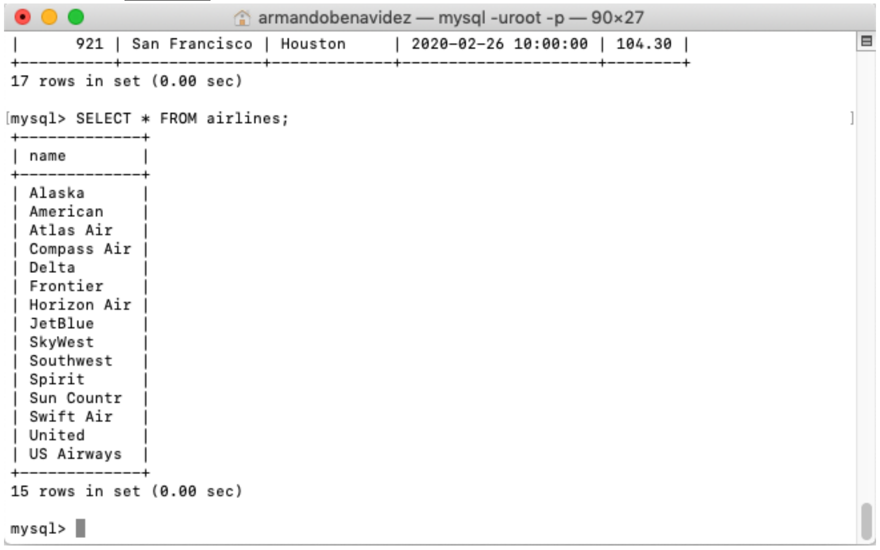
TABLES:

- airplanes (<u>registrationID</u>, capacity)



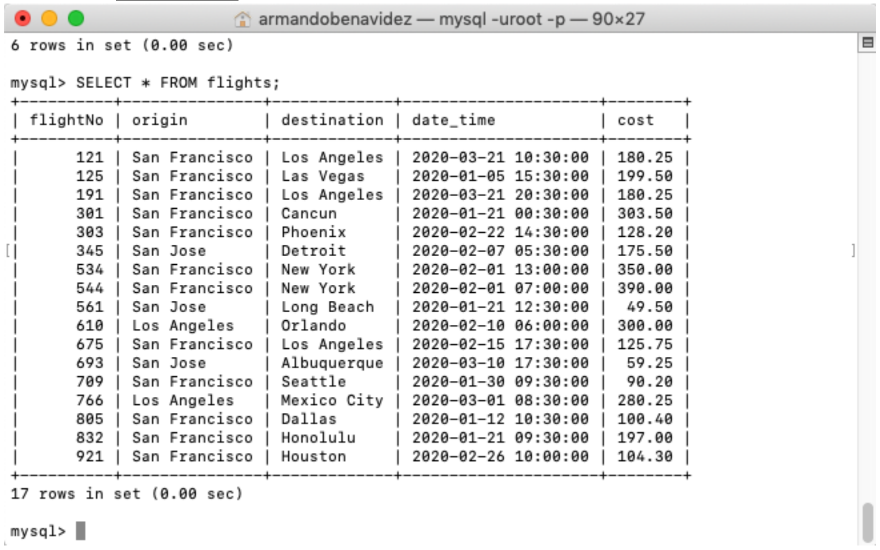
```
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM airplanes;
+----------------+----------+
| registrationID | capacity |
+----------------+----------+
|              1 |      143 |
|              2 |      144 |
|              3 |      134 |
|              4 |      143 |
|              5 |      143 |
|              6 |      200 |
|              7 |      180 |
|              8 |      173 |
|              9 |       75 |
|             10 |       50 |
|             11 |      100 |
|             12 |      150 |
|             13 |      120 |
|             14 |      100 |
|             15 |      143 |
|             16 |      300 |
|             17 |      250 |
+----------------+----------+
17 rows in set (0.00 sec)

mysql> 
```

- airlines(<u>name</u>);



```
|      921 | San Francisco | Houston     | 2020-02-26 10:00:00 | 104.30 |
+----------+---------------+-------------+---------------------+--------+
17 rows in set (0.00 sec)

mysql> SELECT * FROM airlines;
+-------------+
| name        |
+-------------+
| Alaska      |
| American    |
| Atlas Air   |
| Compass Air |
| Delta       |
| Frontier    |
| Horizon Air |
| JetBlue     |
| SkyWest     |
| Southwest   |
| Spirit      |
| Sun Countr  |
| Swift Air   |
| United      |
| US Airways  |
+-------------+
15 rows in set (0.00 sec)

mysql> 
```

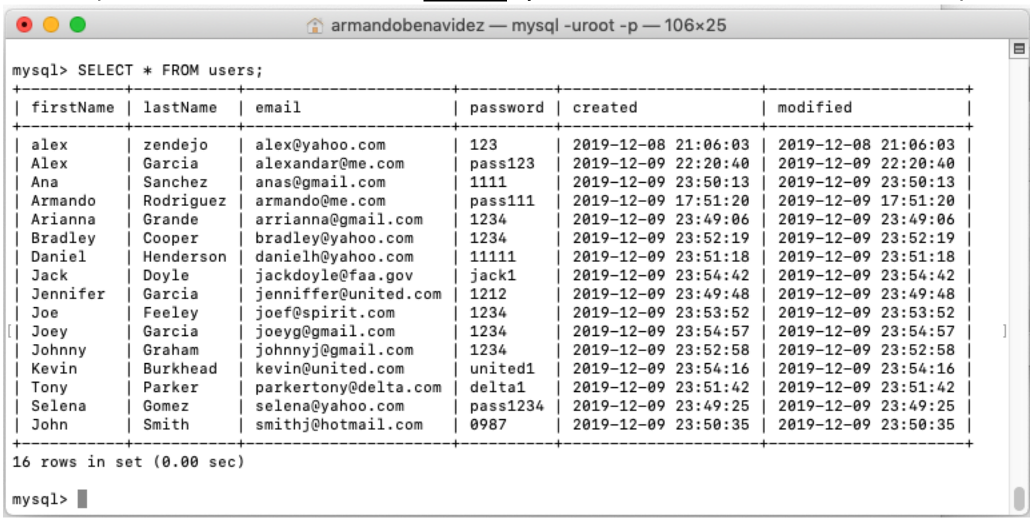
- flights (<u>flightNo</u>, origin, destination, date_time, cost)



```
6 rows in set (0.00 sec)

mysql> SELECT * FROM flights;
+----------+---------------+-------------+---------------------+--------+
| flightNo | origin        | destination | date_time           | cost   |
+----------+---------------+-------------+---------------------+--------+
|      121 | San Francisco | Los Angeles | 2020-03-21 10:30:00 | 180.25 |
|      125 | San Francisco | Las Vegas   | 2020-01-05 15:30:00 | 199.50 |
|      191 | San Francisco | Los Angeles | 2020-03-21 20:30:00 | 180.25 |
|      301 | San Francisco | Cancun      | 2020-01-21 00:30:00 | 303.50 |
|      303 | San Francisco | Phoenix     | 2020-02-22 14:30:00 | 128.20 |
|      345 | San Jose      | Detroit     | 2020-02-07 05:30:00 | 175.50 |
|      534 | San Francisco | New York    | 2020-02-01 13:00:00 | 350.00 |
|      544 | San Francisco | New York    | 2020-02-01 07:00:00 | 390.00 |
|      561 | San Jose      | Long Beach  | 2020-01-21 12:30:00 |  49.50 |
|      610 | Los Angeles   | Orlando     | 2020-02-10 06:00:00 | 300.00 |
|      675 | San Francisco | Los Angeles | 2020-02-15 17:30:00 | 125.75 |
|      693 | San Jose      | Albuquerque | 2020-03-10 17:30:00 |  59.25 |
|      709 | San Francisco | Seattle     | 2020-01-30 09:30:00 |  90.20 |
|      766 | Los Angeles   | Mexico City | 2020-03-01 08:30:00 | 280.25 |
|      805 | San Francisco | Dallas      | 2020-01-12 10:30:00 | 100.40 |
|      832 | San Francisco | Honolulu    | 2020-01-21 09:30:00 | 197.00 |
|      921 | San Francisco | Houston     | 2020-02-26 10:00:00 | 104.30 |
+----------+---------------+-------------+---------------------+--------+
17 rows in set (0.00 sec)

mysql> 
```

- records (<u>flightNo</u>, origin, destination, date_time, cost) //Created as per user

- users (firstname, lastname, email, password, created, modified)



```
mysql> SELECT * FROM users;
+-----------+-----------+----------------------+----------+---------------------+---------------------+
| firstName | lastName  | email                | password | created             | modified            |
+-----------+-----------+----------------------+----------+---------------------+---------------------+
| alex      | zendejo   | alex@yahoo.com       | 123      | 2019-12-08 21:06:03 | 2019-12-08 21:06:03 |
| Alex      | Garcia    | alexandar@me.com     | pass123  | 2019-12-09 22:20:40 | 2019-12-09 22:20:40 |
| Ana       | Sanchez   | anas@gmail.com       | 1111     | 2019-12-09 23:50:13 | 2019-12-09 23:50:13 |
| Armando   | Rodriguez | armando@me.com       | pass111  | 2019-12-09 17:51:20 | 2019-12-09 17:51:20 |
| Arianna   | Grande    | arrianna@gmail.com   | 1234     | 2019-12-09 23:49:06 | 2019-12-09 23:49:06 |
| Bradley   | Cooper    | bradley@yahoo.com    | 1234     | 2019-12-09 23:52:19 | 2019-12-09 23:52:19 |
| Daniel    | Henderson | danielh@yahoo.com    | 11111    | 2019-12-09 23:51:18 | 2019-12-09 23:51:18 |
| Jack      | Doyle     | jackdoyle@faa.gov    | jack1    | 2019-12-09 23:54:42 | 2019-12-09 23:54:42 |
| Jennifer  | Garcia    | jenniffer@united.com | 1212     | 2019-12-09 23:49:48 | 2019-12-09 23:49:48 |
| Joe       | Feeley    | joef@spirit.com      | 1234     | 2019-12-09 23:53:52 | 2019-12-09 23:53:52 |
| Joey      | Garcia    | joeyg@gmail.com      | 1234     | 2019-12-09 23:54:57 | 2019-12-09 23:54:57 |
| Johnny    | Graham    | johnnyj@gmail.com    | 1234     | 2019-12-09 23:52:58 | 2019-12-09 23:52:58 |
| Kevin     | Burkhead  | kevin@united.com     | united1  | 2019-12-09 23:54:16 | 2019-12-09 23:54:16 |
| Tony      | Parker    | parkertony@delta.com | delta1   | 2019-12-09 23:51:42 | 2019-12-09 23:51:42 |
| Selena    | Gomez     | selena@yahoo.com     | pass1234 | 2019-12-09 23:49:25 | 2019-12-09 23:49:25 |
| John      | Smith     | smithj@hotmail.com   | 0987     | 2019-12-09 23:50:35 | 2019-12-09 23:50:35 |
+-----------+-----------+----------------------+----------+---------------------+---------------------+
16 rows in set (0.00 sec)

mysql> 
```

-

PROJECT CONCLUSION

Armando: The objective of this project was simple which was to use MySQL with databases, queries, and tables implemented into a real application one would use. Compared to the beginning of the project and now at the end it's a huge difference. I went from not knowing anything about building an application with databases to I have a completely better understanding of how databases are created from the ground up and are used in real life applications such as this one we created. The lesson I learned is that databases are an important concept to know because they are used in everyday web browsing and when I browse or search for something on the internet, I will think to myself there's probably queries, and databases involved.

Juan Carlos: This project has taught us many things about databases, MySQL, web technologies, and design of a full application. Of course, we were able to have experience with real-world application of the database knowledge that we attained in the classroom, but we also learned much more than that. We learned about how to connect that knowledge with the front end and back end of the application, giving us some experience developing the full stack. This also goes along with the fact that I had very limited experience with web development, so I got a chance to become more familiar with web technologies like React, Node.js, Express.js, and putting them all together. The overall lesson learned was just how important database knowledge is for creating a complete application, especially in a world where data is always increasing and becoming more useful.

Nithil: We all started this project with no knowledge on web development, frameworks, SQL, and all the technologies that we have used. Moving forward with the project we understood these technologies are almost a necessity in this growing tech world. Technologies like React, Nodejs, SQL, Git are very practical and is used almost everywhere for development. We all worked as a team, learned all the technologies together and we were able to witness our growth and understanding of these topics over this course period. The struggles we had initially in using git to seamlessly using git after a point. Connecting frontend, backend, and database was much harder than we expected and almost lost our hopes over there. But we just kept on moving, tried our best every time we were not able to achieve something. We never gave up, and this project taught me a lot about team work too. Overall this was a very satisfying experience.

FUTURE IMPROVEMENTS

With more time and had we had more knowledge of using Node JS and React we could have implemented other features such as a way to accept a payment, included a better user interface with some pictures and auto completing as you type in search queries. Additionally we could have added security or more advanced authentication like using Passport.js system. Overall there is plenty of options for improvement.