AST6 Airline Online Service

**Team 6**

Kunda Wu

Thong Le

John McGinley

**Instructor**

Dr. Mike Wu

# **Table Content**

[**Table Content** 2](#_Toc26654552)

[**Project Overview** 5](#_Toc26654553)

[**System Environment** 7](#_Toc26654554)

[Figure 1: 3-tier Architecture diagram 7](#_Toc26654555)

[Functional Requirements 8](#_Toc26654556)

[Non-functional Issues 9](#_Toc26654557)

[**E/R Diagram** 12](#_Toc26654558)

[**Relation Schemas with description** 13](#_Toc26654559)

[Implementation 23](#_Toc26654560)

[**Overview** 23](#_Toc26654561)

[**Sign In** 24](#_Toc26654562)

[**Steps** 24](#_Toc26654563)

[1. **Retrieve user information from user table** 24](#_Toc26654564)

[2. **Retrieve user preference from preference table via user\_has\_preference relationship** 24](#_Toc26654565)

[3. **Store user and preference in express session** 25](#_Toc26654566)

[**Results** 25](#_Toc26654567)

[1. **Failed authentication** 25](#_Toc26654568)

[2. **Successfully authenticated** 25](#_Toc26654569)

[**Sign Out** 25](#_Toc26654570)

[**Steps** 25](#_Toc26654571)

[1. **Destroy server session** 25](#_Toc26654572)

[2. **Redirect user to Welcome page** 25](#_Toc26654573)

[**Sign Up** 26](#_Toc26654574)

[**Steps** 26](#_Toc26654575)

[1. **Insert user information into user table** 26](#_Toc26654576)

[2. **Insert default values into preference** 26](#_Toc26654577)

[3. **Insert user\_id / sha1(email) and preference\_id into user\_has\_preference** 26](#_Toc26654578)

[4. **Send Welcome email** 26](#_Toc26654579)

[**Results** 26](#_Toc26654580)

[1. **Failed validation** 26](#_Toc26654581)

[2. **Successfully signed up. System will send email (if not seeing, check Spam) to welcome new user** 27](#_Toc26654582)

[**Forgot Password** 28](#_Toc26654583)

[**Steps** 28](#_Toc26654584)

[1. **Check if given email existing in the system** 28](#_Toc26654585)

[2. **Add (update) recovery\_link, recovery\_exp into user table** 28](#_Toc26654586)

[3. **Send Recovery email** 28](#_Toc26654587)

[**Results** 28](#_Toc26654588)

[1. **Failed validation** 28](#_Toc26654589)

[2. **Successfully requested for recovery link** 29](#_Toc26654590)

[3. **Click on recovery link; then, update password (see User Password)** 29](#_Toc26654591)

[**Dashboard** 29](#_Toc26654592)

[**Steps** 29](#_Toc26654594)

[1. **Get flights associated with authenticated user\_id** 29](#_Toc26654595)

[2. **Booking number link in Flight table is to redirect to Booking Detail (see Booking Detail)** 30](#_Toc26654596)

[**Results** 30](#_Toc26654597)

[1. **Retrieve flights associated with authenticated user** 30](#_Toc26654598)

[**Booking Details** 31](#_Toc26654599)

[**Steps** 31](#_Toc26654600)

[1. **Retrieve booking information with given booking number** 31](#_Toc26654601)

[2. **Cancel booking by selecting one or more passengers** 31](#_Toc26654602)

[b. **Update booking\_status (from Normal to Cancelled) in booking\_passenger** 31](#_Toc26654603)

[c. **Retrieve flight\_id from user\_book\_flight with given booking\_number** 31](#_Toc26654604)

[d. **Update seats for flight** 31](#_Toc26654605)

[**Results** 32](#_Toc26654606)

[**User Profile** 32](#_Toc26654607)

[**Steps** 32](#_Toc26654608)

[**Results** 32](#_Toc26654609)

[**User Basic Information** 32](#_Toc26654610)

[**User Password** 33](#_Toc26654611)

[**Steps** 33](#_Toc26654612)

[1. **If update password from recovery link, looking up for user with corresponding recovery\_link and check if link is expired with recovery\_exp** 33](#_Toc26654613)

[2. **Update password with given user\_id and given current password** 33](#_Toc26654614)

[**Results** 33](#_Toc26654615)

[1. **Failed validation: password is too short or too simple** 33](#_Toc26654616)

[2. **Successfully updated password and stay at the same page** 33](#_Toc26654617)

[**User Preference** 34](#_Toc26654618)

[**Steps** 34](#_Toc26654619)

[1. **Update all attributes in user preference with session-stored preference\_id** 34](#_Toc26654620)

[**Results** 34](#_Toc26654621)

[1. **Successfully updated (alert appears from top)** 34](#_Toc26654622)

[**Booking (Search)** 35](#_Toc26654623)

[**Steps** 35](#_Toc26654624)

[1. **Fetch airport data** 35](#_Toc26654625)

[2. **Get list of airports for autocomplete input fields (Flight From, Flight To)** 35](#_Toc26654626)

[3. **Trip type (Roundtrip, One-way)** 36](#_Toc26654627)

[a. **Roundtrip** 36](#_Toc26654628)

[b. **One-way** 37](#_Toc26654629)

[4. **Passengers (number of passengers)** 37](#_Toc26654630)

[5. **Advanced Options** 37](#_Toc26654631)

[a. **Flight class (all, economy, business)** 37](#_Toc26654632)

[b. **Max price** 37](#_Toc26654633)

[c. **Sort by** 37](#_Toc26654634)

[6. **Click Search** 38](#_Toc26654635)

[7. **Function to process and build request call on front-end side** 38](#_Toc26654636)

[8. **Function to process request call and build query on back-end side** 38](#_Toc26654637)

[**Booking (Search Results)** 39](#_Toc26654638)

[**Steps** 39](#_Toc26654639)

[1. **User can refine search by clicking on Modify link** 39](#_Toc26654640)

[2. **User can select different Sort By (Depart Time, Arrive Time, Price)** 39](#_Toc26654641)

[3. **Select a flight by clicking on Economy or Business button** 39](#_Toc26654642)

[4. **Complete selecting flight, Continue to Reservation button will appear** 39](#_Toc26654643)

[**Results** 40](#_Toc26654644)

[**Booking (Reservation)** 40](#_Toc26654645)

[**Steps** 40](#_Toc26654646)

[1. **Click on Flight Summary to view summary of selected flight(s)** 40](#_Toc26654647)

[2. **Click Add Passenger button to add one more passenger to book** 41](#_Toc26654648)

[3. **Click on Send confirmation (optional) to receive Confirmation Email** 41](#_Toc26654649)

[4. **After complete filling out inputs for passenger, click Finish button to complete Reservation (skip payment process, assume all payment is valid)** 41](#_Toc26654650)

[**Results** 41](#_Toc26654651)

[1. **Failed validation** 41](#_Toc26654652)

[2. **Successfully booked the flight(s) (alert appears from top).** 42](#_Toc26654653)

[**Setup procedures** 42](#_Toc26654654)

[**Overview** 42](#_Toc26654655)

[**Requirements** 42](#_Toc26654656)

[**Steps** 42](#_Toc26654657)

[CONCLUSION 43](#_Toc26654658)

[**Lessons learned** 43](#_Toc26654659)

[**Future Improvement** 43](#_Toc26654660)

# **Project Overview**

Our objective is to develop a web-based application for stakeholders who have recently founded an airline named Airline Services Team-6 (or AST6) and are in need of a website that can provide a booking service to their customers. Because they are new in the market, they want their website to be released as fast as possible, and they are giving us 3 months to accomplish their expectations. Needless to say, they consider efficiency, security, reliability and scalability of the utmost importance. A small list of the features that they find absolutely essential:

* Client:

1. UI is responsive, simple but modern.
2. Console interface for users to manage their flights and flight history
3. Search functionality that allows users to sort their options by price/class/etc

* Server:

1. Fast and accurate response and handling logic operation based on user’s input.
2. Connect client-server workflow to database management system

* Database:

1. Scalable: not specific
2. Secure: user’s information
3. Reliable: data should be available all time
4. Consists of a collection of data related to flights (price, number of seats, etc)

## 

# **System Environment**

Client

(ReactJS)

HTTP request

query

Home server

return data

HTTP response

Business layer

(express.js)

Data access layer (mysql

-connector)

Database

(MySQL)

Server (2-layer)

### Figure 1: 3-tier Architecture diagram

* Hardware & Software:

1. Apache2
2. Visual Studio Code (latest)
3. OpenVPN (future)

* RDBMS: MySQL community server (v8.0.17)
* Host: Home server (future)
* Frameworks & its core dependencies

1. Node.js (v11.10.1), npm (v6.7.0)
2. Front-end: React (v3.0.1), Redux (latest)
3. Back-end: express.js (v4.16.1) and mysql (connector, v2.17.1)

# Functional Requirements

1. **Users and how users access your system**

There is only one type of user which is customer/ client to our airline services. All the flight information and processes will be pre-set and simulated.

\*\* Users must sign up to start using our services.

* 1. Sign-Up

Users will be prompted to enter some information:

* + - First name, middle initial and last name
    - Email. This is important because users will have to use email to sign in our services, to get email confirmation (see 2-factor authentication, page 5), or to reset their password
    - Password (at least 6 characters, at least 1 special character)
    - User\_id (not shown to user): generated by sha1(email)
  1. Sign-In
  + Users can sign in using registered email and password
  1. Reset Password
* The function invokes when users click the ‘Forgot password’ button.
* It will then ask the user to provide the email to receive reset-password link.

1. **Functionality/features, functional processes and I/O(s)**

User Dashboard: When users sign in, they will have access to dashboard where they can search for flights, reserve flights, change their profile and as well as access to flight management

* + - 1. Search for flights:
* Users can pick departure and destination
* Users can refine search options

i. Based on Roundtrip or one-way

ii. Based on depart/ arrive time and number of passenger

iii. Based on price range (Allow users to set max price)

iv. Based on price sort (lowest to highest)

v. Based on flight classes (First Class, Economy Plus, or Economy)

* + - 1. Book flight: After users complete flight search and ready to pick up flight line
* Users can start to reserve it by entering more personal information such as passport/ DL number, and assume all the payment is valid. Then, users will be prompted to confirm or cancel the reservation

I. Confirm: Users will receive a confirmation email for reservation if email is entered

II. Cancel booking by click on Modify (in Flight Summary)

* Finally, users will be redirected back to state of searching for a flight
  + - 1. Profile edit:

Users can update their password and set their preferences:

* + - Preferred Flight Class: next time they search, their flight class will be set as their default, they can change it anytime
    - Preferred Depart/ arrive time
    - Preferred Max price
    - Preferred Sort by
      1. Booking detail: users can view information about their flights booked with our airline.
* Users can use this feature to cancel booking for one or more passengers

# Non-functional Issues

* **Graphical User Interface (GUI)**

We will be building a Graphical User Interface for this application using React, Redux, HTML and CSS3/Bootstrap. To use our application to search for flights, users will have to create an account with sign up form using an ‘Enroll now’ button that will be placed in the center of the page. If they are not signed in and try to access via direct url, they will be redirected to the Welcome page where users can sign in using the form with their registered email and password. Also in the Welcome page, users can reset their password with ‘Forgot password’ using their registered email address. This will allow us to send them a verification email and allow them to change their password.

Once users sign in, they can search for flights using our search bar which will be placed at the top of the page. When they search, they will fill in their departing location and desired destination, as well as departure and return dates. Users will then click search or press enter to start querying for their flights, then the page will display a list of flights matching the given criteria in our database. Users can open up advanced search to refine some search options as well as prioritization for time in day and flight classes. Moreover, users will be able to sort the returned flights by total duration/number of stops and price, they will also be able to set the min and max price when searching so they do not have to view tickets outside of their budget. Having these search options will allow users to find a flight that they need in the most streamlined way. In addition, users can change their profile by clicking a button in the upper right part of user dashboard. This will bring users to profile page that allows them to change their password and change their preferences. There will be a menu on the left of the page that allows them to do things such as manage their flight information, change or upgrade their flight, check-in their flight, or cancel entirely.

* **Security and miscellaneous**
  1. Authorization:

The security of users will be protected by email address and password. Enrolled user​ ​account​ ​information​ ​will​ ​be​ ​securely​ ​stored​ ​in​ the database.​ ​​​In​ ​order​ ​to​ ​login​ ​to the​ ​system​, the ​user​ ​must​ ​provide​ ​an​ ​existing​ ​email​ ​account​ ​along​ ​with​ ​the associated​ ​password.​ Meanwhile, if a user happens to forget password, the user have to reset his password before signing in to the system. ​The​ database system ​will​ ​be​ ​implemented​ ​using​ ​local host ​providing​ connection​ ​to​ ​server.

* 1. Secure sign-in: two-factor authentication (a confirmation code will be sent to user’s registered email)
  2. User’s information is encrypted (simple encryption) before persisting into the database system
  3. Simulated flight information

Each flight will have a maximum number of seats and each person can not double-book the same flights. When the user enters a search query, a series of desired flights will be displayed, with the number of seats left. If the number of seats is zero, that flight will not be shown to the user, even if it meets their other criteria.

* 1. Maintainability and scalability

System will be designed so it will be easy to maintain and scale up when the business grows

Also, it depends on further requirements from the stakeholders

* 1. Reliability of booking/flight information

All the reservation/ flight information will be persisted and secured

* 1. Performance of this booking service (Times, overall experience, and etc)

In this AST6 project, we implement the concurrently popular technology Node.js and React to build the client side and server side.

## 

# **E/R Diagram**

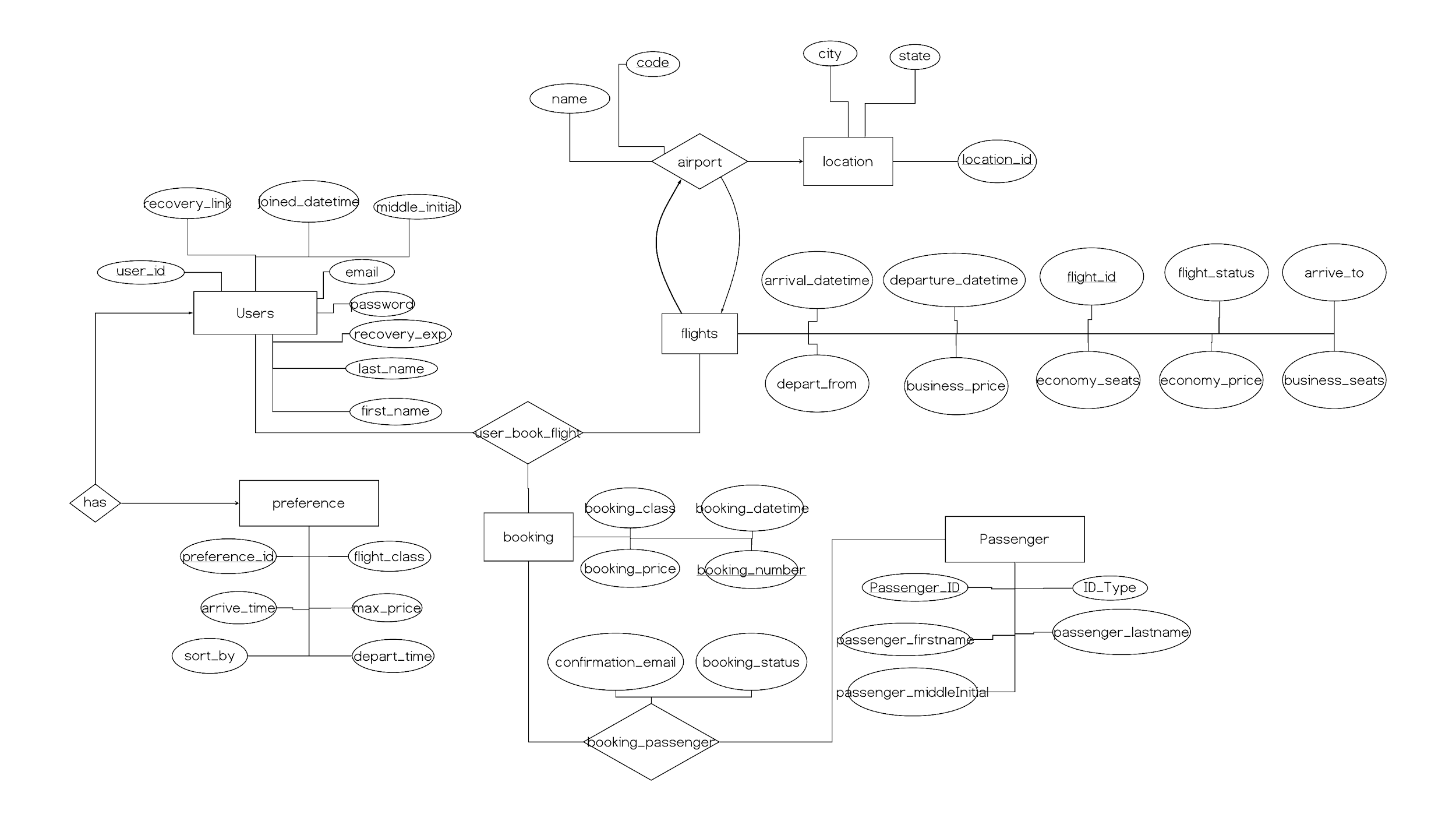
****

Figure 2: E/R diagram for AST6 Airline Online Services

# **Relation Schemas with description**

1. **user(user\_id, email, password, joined\_datetime, first\_name, last\_name, middle\_initial, recovery\_link, recovery\_exp):** Each user will have to provide email, first name, last name, middle name initial, join\_datetime, and has a unique user ID which is assigned by System. Also, each user has specific field for recovery\_link and recovery\_exp. The recovery\_link determine how long the recovery will be expirated. Each user can have exactly one preference at a time, but has a many to one relationship with special offers and tickets.

Non-trivial FDs:

User\_id → email

User\_id → password

User\_id → join\_datetime

User\_id → first\_name

User\_id → last\_name

User\_id → middle\_initial

User\_id → recovery-link

recovery\_link → recovery\_exp

Normalization: Prefect to BCNF:

Recovery\_link →  recovery\_exp violates BCNF

Because {recovery\_link}+ → {recovery\_link, recovery\_exp}

R = **{user\_id, email, password, joined\_datetime, first\_name, last\_name, middle\_initial, recovery\_link, recovery\_exp}**

**R1 = {recovery\_link}+**

**R2 = R - {recovery\_link}+ + {recovery\_link} = {user\_id, email, password, joined\_datetime, first\_name, last\_name, middle\_initial, recovery\_link}**

**R decompose to R1 + R2**

**CREATE TABLE `user` (**

**`user\_id` varchar(40) NOT NULL,**

**`email` varchar(45) NOT NULL,**

**`password` varchar(256) NOT NULL,**

**`joined\_datetime` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,**

**`first\_name` varchar(255) NOT NULL,**

**`last\_name` varchar(255) NOT NULL,**

**`middle\_initial` char(1) NOT NULL,**

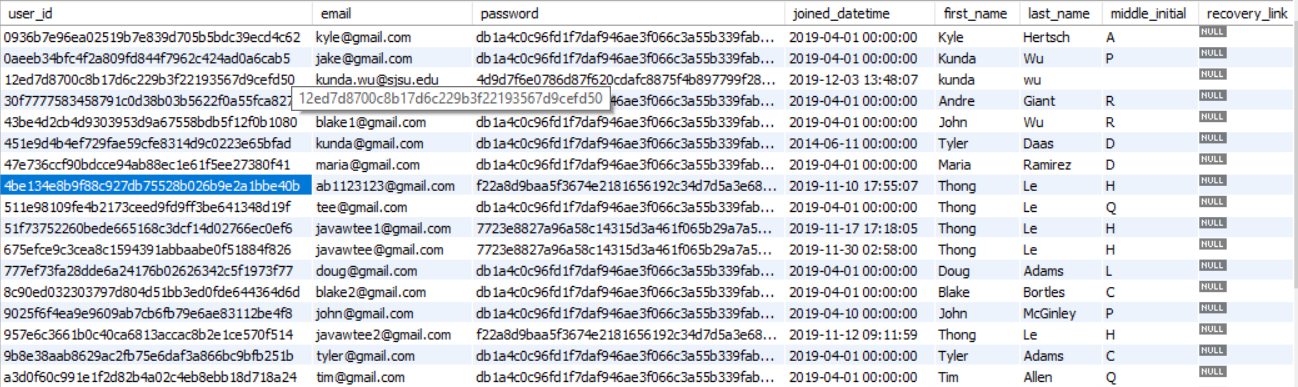
**`recovery\_link` varchar(256) DEFAULT NULL,**

**`recovery\_exp` bigint(20) DEFAULT NULL,**

**PRIMARY KEY (`user\_id`),**

**UNIQUE KEY `email\_UNIQUE` (`email`)**

**)**



1. **preference(preference\_id, flight class, arrive\_time, depart\_time, max\_price, sort\_by):** Each registered user has exactly one list of preferences, so it is a many-to-one relationship (each user can have exactly one preference, but a preference set may be had by multiple users) . A preference helps user to pre-select their options in flight search (flight time (day/night), ticket class (Business class/ Economic class). Alert is Yes/No to notify user when flight status changes. Max price is to set a limit for a ticket price

Non-trivial FDs:

Preference\_id → flight\_time

Preference\_id → sort\_by

Preference\_id → depart\_time

Prefenece\_id → ticekt\_class

Ticket\_class → max\_price

Normalization: Prefect to BCNF

**preference(preference\_id, flight\_time, ticket\_class, sort\_by, depart\_time, max\_price)**

Ticket\_class→ max\_price violates BCNF

Because {ticekt\_class}+ = {ticekt\_class, max\_price}.

R = {**preference\_id, flight\_time, ticket\_class, sort\_by, depart\_time, max\_price**}

R1 = {ticket\_class, max\_price}

R2 = R - R1 + {ticket\_class} = {**preference\_id, flight\_time, ticket\_class, sort\_by, depart\_time**}

R decompose to R1 and R2

**CREATE TABLE `preference` (**

**`preference\_id` int(11) NOT NULL AUTO\_INCREMENT,**

**`depart\_time` varchar(45) DEFAULT 'any',**

**`arrive\_time` varchar(45) DEFAULT 'any',**

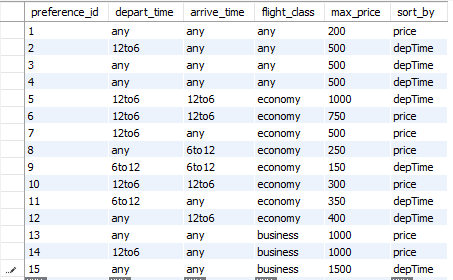
**`flight\_class` varchar(45) DEFAULT 'any',**

**`max\_price` double DEFAULT '500',**

**`sort\_by` varchar(45) DEFAULT 'depTime',**

**PRIMARY KEY (`preference\_id`)**

**)**



1. **user\_has\_preference [Relationship] (user\_id, preference\_id)**

user\_has\_preference is the relationship between the user and preference tables. It consists of unique preference\_id, user\_id. It requires these to link to two tables together, but needs no attributes of its own.

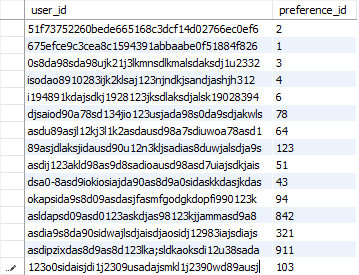
**CREATE TABLE `user\_has\_preference` (**

**`user\_id` varchar(40) NOT NULL,**

**`preference\_id` int(10) NOT NULL,**

**PRIMARY KEY (`user\_id`)**

**)**



Non-trivial FDs

User\_id → prefernce\_id

Normalization: Prefect to BCNF

We know that user\_id determine all other attributes.

{user\_id}+ = {passenger\_id, user\_id}

On the all the FDs,  we have superkey on the left hand side. Therefore, no violation occurs. The schema already is in BCNF.

1. **flight(flight\_id, departure\_datetime, depart\_from, arrival\_datetime, arrive\_to, flight\_status [onboarding|delay|postponed|ready|departed|arrived], economy\_seats, economy\_price, business\_seats, business\_price):** When a flight is generated by the system it is assigned a unique id. Each flight has one plane, but a plane can be assigned to flights until it stops working (one->many). Each flight has information of where it departs from and where it arrives to. Also, each flight has its own status, i.e: to let passengers know whether it is going to fly on-time or postponed, ready status is to determine when all passengers at the gate are onboarded.

Non-trivial FDs:

flight\_id → departure\_datetime

flight\_id → departure\_from

flight\_id → arrival\_datetime

flight\_id → arrival\_to

flight\_id → flight\_status

Economy\_price → economy\_seats

flight\_id → economy\_price

Business\_price → business\_seats

flight\_id → business\_price

**Normalization: Prefect to BCNF**

**R = {flight\_id, departure\_datetime, depart\_from, arrival\_datetime, arrive\_to, flight\_status [onboarding|delay|postponed|ready|departed|arrived], economy\_seats, economy\_price, business\_seats, business\_price}**

Economy\_price → economy\_seats violates BCNF

{Economy\_price}+ = {Economy\_price, economy\_seat}

R1 = {Economy\_price}+

R2= R - R1 + {Economy\_price}+ = {**flight\_id, departure\_datetime, depart\_from, arrival\_datetime, arrive\_to, flight\_status [onboarding|delay|postponed|ready|departed|arrived], economy\_price, business\_seats, business\_price**}

Business\_price → business\_seats violates BCNF (R2)

R21 = {Business\_price}+ = {Business\_price, Business\_seat}

R22 = R2 - R21 + {Business\_price} = {**flight\_id, departure\_datetime, depart\_from, arrival\_datetime, arrive\_to, flight\_status[onboarding|delay|postponed|ready|departed|arrived], economy\_price, business\_price}**

**R decompose to R21 and R22 and R1**

**CREATE TABLE `flight` (**

**`flight\_id` int(11) NOT NULL AUTO\_INCREMENT,**

**`departure\_datetime` timestamp NOT NULL,**

**`depart\_from` varchar(45) NOT NULL,**

**`arrival\_datetime` timestamp NOT NULL,**

**`arrive\_to` varchar(45) NOT NULL,**

**`flight\_status` varchar(45) NOT NULL,**

**`economy\_price` double NOT NULL DEFAULT '0',**

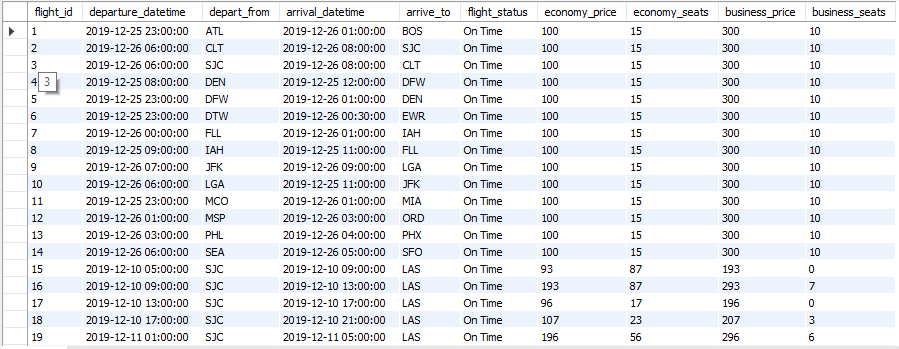
**`economy\_seats` int(10) NOT NULL DEFAULT '0',**

**`business\_price` double NOT NULL DEFAULT '0',**

**`business\_seats` int(10) NOT NULL DEFAULT '0',**

**PRIMARY KEY (`flight\_id`,`departure\_datetime`,`depart\_from`)**

**);**



1. **Booking(booking\_number, booking\_datetime, booking\_price, booking\_class):** Booking is generated by the system with an auto incremented id. Each booking can have one class (Business class/ Economic class) is assigned for a flight. Booking price can be updated by the system, i.e: regarding the time. Booking\_datetime is the time that the booking is for

Non-trivial FDs:

booking\_number → booking\_datetime

booking\_class → booking\_price

booking\_number→ booking\_class

Normalization: Prefect to BCNF

booking\_class → booking\_price violates BCNF

Because {booking\_class}+ = {booking\_class, booking\_price}

R = {**booking\_number, booking\_datetime, booking\_price, booking\_class**}

R1 = {booking\_class}+ = {booking\_class, booking\_price}

R2 = R - R1 + {booking\_class} = {**booking\_number, booking\_datetime,booking\_class**}

R decomposes to R1 and R2

**CREATE TABLE `booking` (**

**`booking\_number` varchar(255) NOT NULL,**

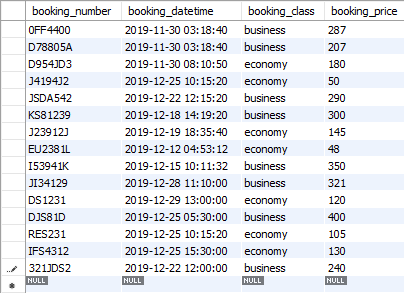
**`booking\_datetime` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,**

**`booking\_class` varchar(45) NOT NULL,**

**`booking\_price` double NOT NULL,**

**PRIMARY KEY (`booking\_number`)**

**);**



**6. Booking\_passenger** (passenger\_id, booking\_number, booking\_status, confirmation\_email)

Booking\_passenger is a relationship table that set the connection between the booking and passenger, each passenger can have zero or more bookings. Booking\_passenger consists of confirmation\_email, booking\_status and it bring the primary key (passenager\_id, booking\_status) from booking and passenger.

**CREATE TABLE `cs157a\_project`.`booking\_passenager` (**

**`booking\_number` INT NOT NULL,**

**`confirmation\_email` VARCHAR(45) NOT NULL,**

**`passenager\_id` INT NOT NULL,**

**`booking\_status` varchar(6) NOT NULL,**

**PRIMARY KEY (`booking\_number`, `passenager\_id`));**

Non-trivial FDs:

Passenager\_id, booking\_number → booking\_status

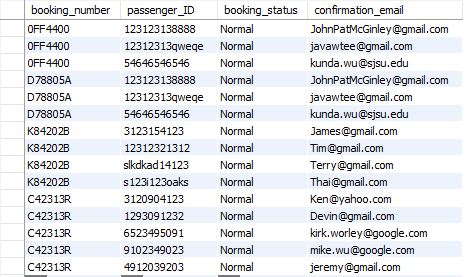
Passenager\_id, booking\_number → confirmation\_email

Normalization: Prefect to BCNF

We know that Passenager\_id, booking\_number determine all other attributes.

{Passenager\_id, booking\_number}+ = {passenger\_id, booking\_number, booking\_status, confirmation\_email}

On the all the FDs,  we have superkey on the left hand side. Therefore, no violation occurs. The schema already is in BCNF.



**7. user\_book\_flight**(user\_id, booking\_number, flight\_id, flight\_class)

User\_book\_flight describes the relationship between a user, booking, and a flight. It consists of three identities, flight\_id, booking\_number, users\_id.

CREATE TABLE `user\_book\_flight` (

  `user\_id` varchar(40) NOT NULL,

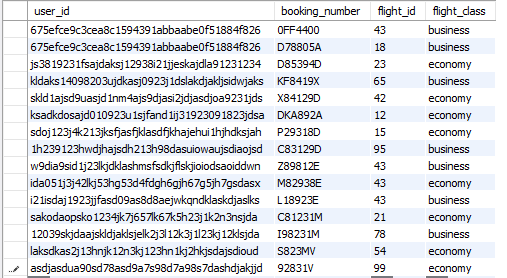
  `booking\_number` varchar(255) NOT NULL,

  `flight\_id` int(10) NOT NULL,

  `flight\_class` varchar(40) NOT NULL,

  PRIMARY KEY (`user\_id`)

);



**user\_book\_flight**(user\_id, booking\_number, flight\_id, flight\_class)

Non-trivial FDs:

User\_id → booking\_number

User\_id → flight\_id

booking \_number → flight\_class

Normalization: Prefect to BCNF:

Booking\_number → flight\_class violates BCNF

Because {booking\_number}+ = {booking\_number, flight\_class}

R = {user\_id, booking\_number, flight\_id, flight\_class}

R1 = {booking\_number}+

R2 = R - R1 + {booking\_number} = {user\_id, booking\_number, flight\_id}

R decompose to R1 and R2

**8. airport (code, name, location\_id)**: Airport is where flights take off and land. One city cannot have two airports with the same name. Airports have a from/to relationship with flights, and this is a many to many relationship and flights go to many airports and airports receive many flights.

Non-trivial FDs:

Code, location\_id → name

Normalization: Prefect to BCNF

We know that code, location\_id determine all other attributes.

{code, location\_id}+ = {**code, name, location\_id**l}

On all the FDs,  we have superkey on the left hand side. Therefore, no violation occurs. The schema already is in BCNF.

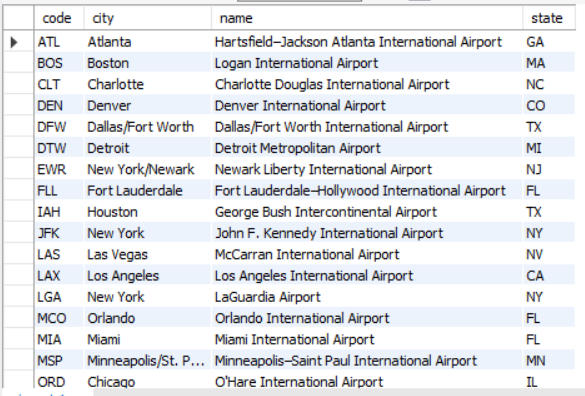
**CREATE TABLE `cs157a\_project`.`airport` (**

**`location\_id` INT NOT NULL,**

**`name` VARCHAR(45) NOT NULL,**

**`code` INT NOT NULL,**

**PRIMARY KEY (`code`, `location\_id`));**



9. **Location** (location\_id, city, state)

Location is the table that stores a specification information of a location. Each location is identified by its location\_id uniquely, and it also consists of attributes like city and state. It will associate with the airport, such that each airport on a unique location.

CREATE TABLE `location` (

  `location\_id` int(11) NOT NULL AUTO\_INCREMENT,

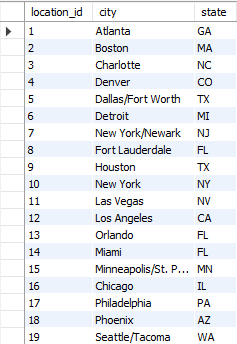
  `city` varchar(45) NOT NULL,

  `state` varchar(45) NOT NULL,

  PRIMARY KEY (`location\_id`),

  UNIQUE KEY `city\_UNIQUE` (`city`)

);



Non-Trivial FDs:

Location\_id → city

Location\_id → state

Normalization: Prefect to BCNF

We know that location\_id determine all other attributes.

{location\_id}+ = **{**location\_id, city, state**}**

On the all the FDs,  we have superkey left hand side. Therefore, no violation occurs. The schema already is in BCNF.

10. passenger**(passenger\_id, ID\_Type, first\_name, last\_name, middle\_initial):** Each passenger will have to provide unique passenger\_id along with their ID\_type,  first name, last name, middle name initial in order successfully abording. Each passenger can have exactly one preference\_id.

Non-trivial FDs:

passenger\_id → ID-Type

passenger\_id → first\_name

passenger\_id → last\_name

passenger\_id → middle\_initial

Normalization: Prefect to BCNF

We know that Passenager\_id determine all other attributes.

{Passenager\_id}+ = **(passenger\_id, ID\_Type, first\_name, last\_name, middle\_initial)**

On the all the FDs,  we have superkey left hand side. Therefore, no violation occurs. The schema already is in BCNF.

**CREATE TABLE `cs157a\_project`.`passenger` (**

**`passenger\_id` VARCHAR(32) NOT NULL,**

**`ID\_Type` VARCHAR(45) NULL,**

**`first\_name` VARCHAR(45) NOT NULL,**

**`last\_name` VARCHAR(45) NOT NULL,**

**`middle\_initial` CHAR(1) NOT NULL,**

**PRIMARY KEY (`passenger\_id`);**



# Implementation

## **Overview**

Our DB web-based application was implemented on React and Redux as front-end side, and ExpressJS and MySQL as back-end side.

React and Redux:

1. React is library for front-end development
2. React components are used to display view to users
3. React components are scalable and reusable as well as maintainable
4. Redux is used to manage React states
5. Redux is good for scalability (adding future features)

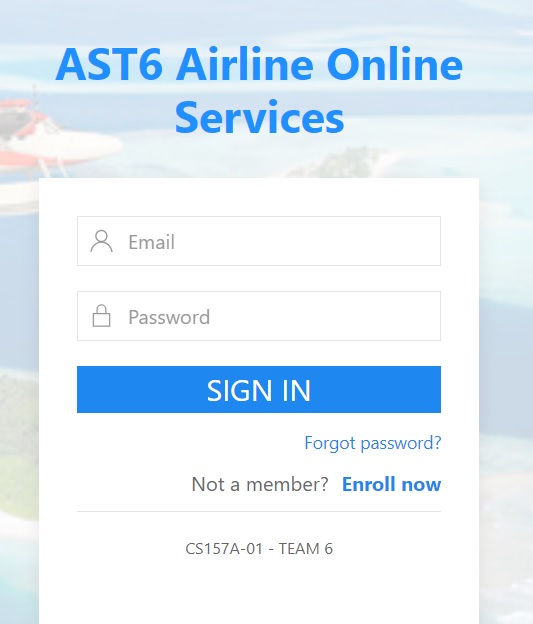
ExpressJS:

1. Framework that is built on nodejs and used for back-end development
2. Using mysql connector (package) to connect MySQL database
3. Using built-in/ server session to store user information  to authorize user when calling api and user’s preferences

MySQL:

1. Store user and system data
2. Return system data (flights and bookings)

## **Sign In**



React components: SignIn

Express route: POST /user/signin

Tables: user, user\_has\_preference, preference

### **Steps**

#### **Retrieve user information from user table**

select count(user\_id) as C, user\_id as ID, first\_name as firstName,

last\_name as lastName, middle\_initial as middleInitial, email, joined\_datetime as joinedDate

from user

where email=? and password=sha2(?,0);

#### **Retrieve user preference from preference table via user\_has\_preference relationship**

select p.preference\_id as preferenceId, p.depart\_time as departTimeId, p.arrive\_time as arriveTimeId, p.flight\_class as flightClassId, p.max\_price as maxPrice, p.sort\_by as sortById

from preference p, user\_has\_preference uhp

where uhp.preference\_id = p.preference\_id and uhp.user\_id=?;

#### **Store user and preference in express session**

### **Results**

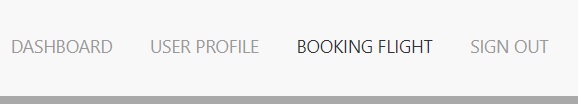
#### **Failed authentication**

* 1. Email or password is not correct (alert appears from top)
  2. Stay at the same page

#### **Successfully authenticated**

* 1. Go to Dashboard

## **Sign Out**



React components: NavBar, SignOut

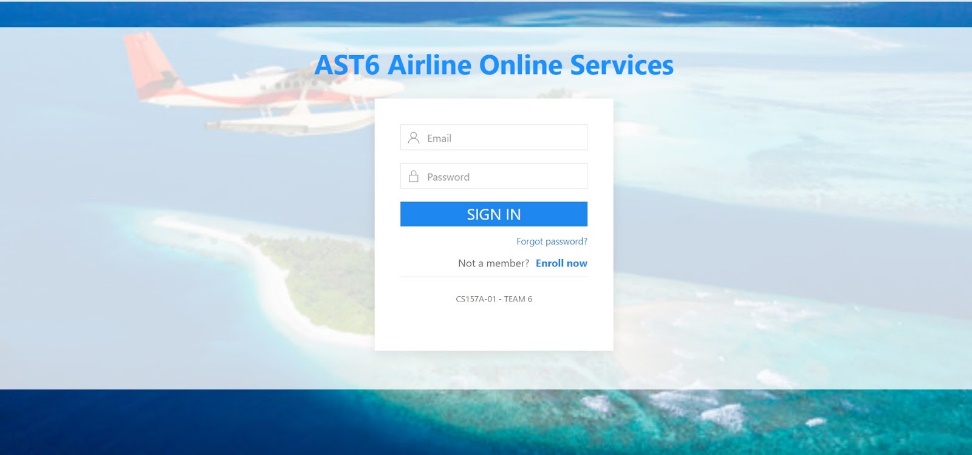
Express route: POST /user/signout

Tables: *Not applicable*

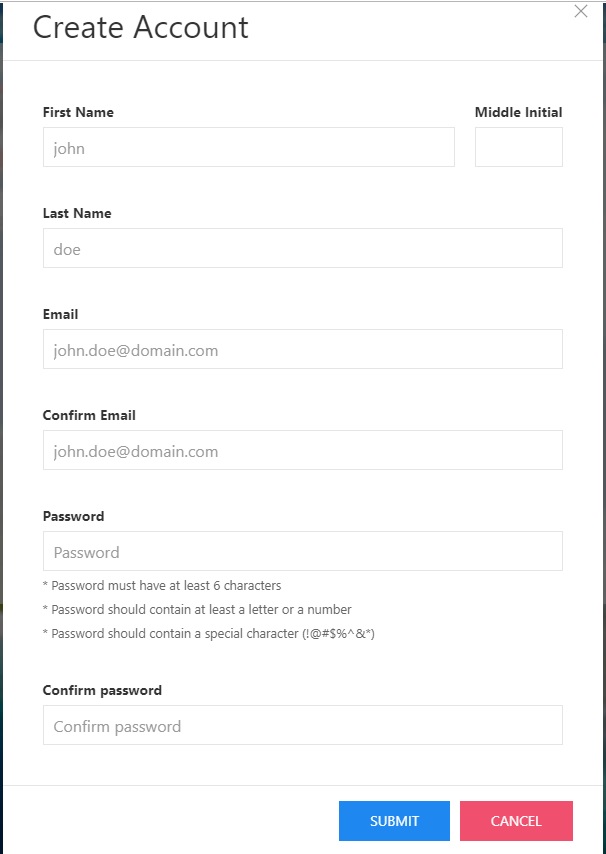
### **Steps**

#### **Destroy server session**

#### **Redirect user to Welcome page**



## **Sign Up**



React components: SignUp, PasswordField

Express route: /user/signup

Tables: user, user\_has\_preference, preference

### **Steps**

#### **Insert user information into user table**

insert into user (user\_id, email, password, first\_name, last\_name, middle\_initial) values (sha1(?), ?, sha2(?, 0), ?, ?, ?);

#### **Insert default values into preference**

insert into preference values()

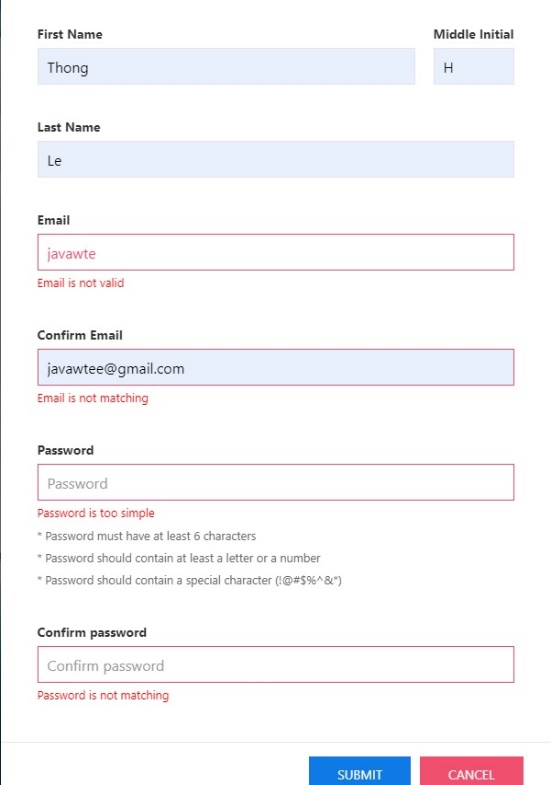
#### **Insert user\_id / sha1(email) and preference\_id into user\_has\_preference**

insert into user\_has\_preference values (sha1(?), ?);

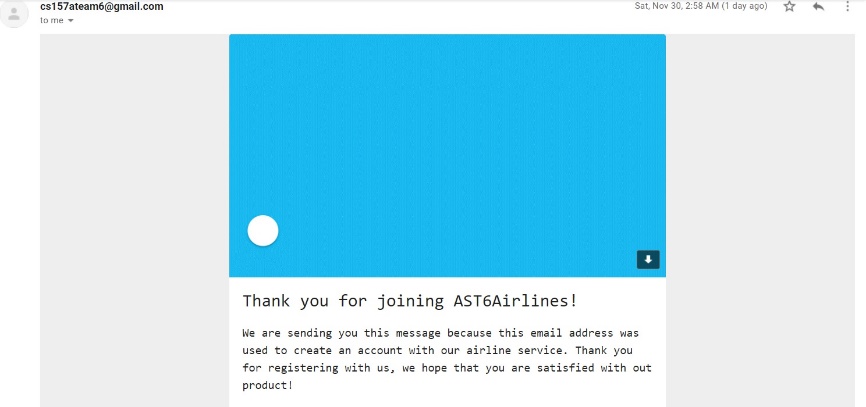
#### **Send Welcome email**

### **Results**

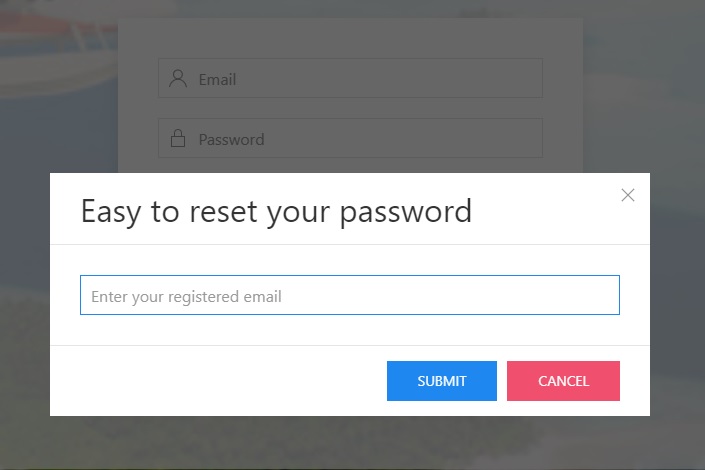
#### **Failed validation**



#### **Successfully signed up. System will send email (if not seeing, check Spam) to welcome new user**



## **Forgot Password**



React components: ForgotPassword, RecoveryPassword, PasswordField

Express routes:

* GET /user/recoverylink
* GET /user/confirmrecoverylink

Tables: user (column: recovery\_link, recovery\_exp)

### **Steps**

#### **Check if given email existing in the system**

select user\_id from user where email=?;

#### **Add (update) recovery\_link, recovery\_exp into user table**

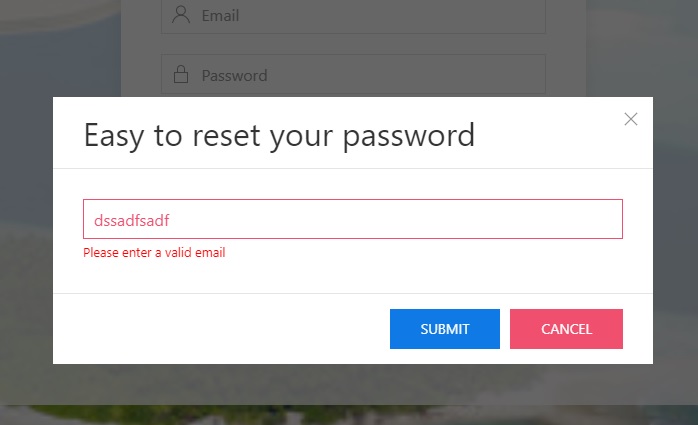
* 1. Recovery\_link = “/rl=” + uuid(email+time, “1b671a64-40d5-491e-99b0-da01ff1f3341”)
  2. Recovery\_exp (expiration time for link): time received request + 15 minutes

update user set recovery\_link=?, recovery\_exp=? where email=?;

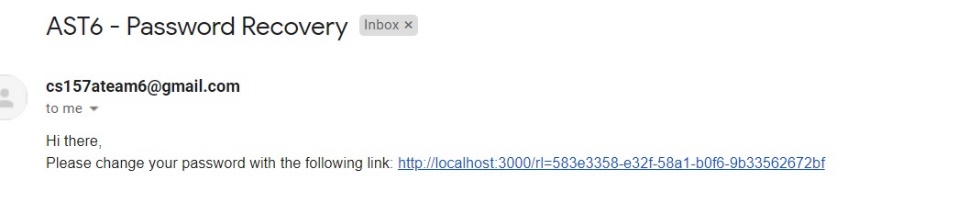
#### **Send Recovery email**

### **Results**

#### **Failed validation**



#### **Successfully requested for recovery link**



#### **Click on recovery link; then, update password (see** [**User Password**](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.w8o508mvx1fi)**)**

## **Dashboard**

### 

React components: Dashboard, FlightTable

Express routes: GET /user/getflights

Tables: user\_book\_flights, flight, flight join airport on depart\_from = code, flight join airport on arrive\_to = code

### **Steps**

#### **Get flights associated with authenticated user\_id**

select booking\_number as bookingNumber, departFrom, arriveTo, departure\_datetime as departTime, arrival\_datetime as arriveTime

from user\_book\_flight u,

flight f,

(select a.flight\_id, city as departFrom

  from ( select flight\_id, location\_id from flight join airport on depart\_from = code) a , location b where a.location\_id = b.location\_id) d,

(select a.flight\_id, city as arriveTo

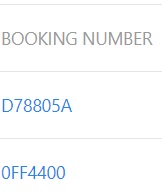
  from ( select flight\_id, location\_id from flight join airport on arrive\_to = code) a , location b where a.location\_id = b.location\_id) a

where

user\_id=? and u.flight\_id = f.flight\_id and

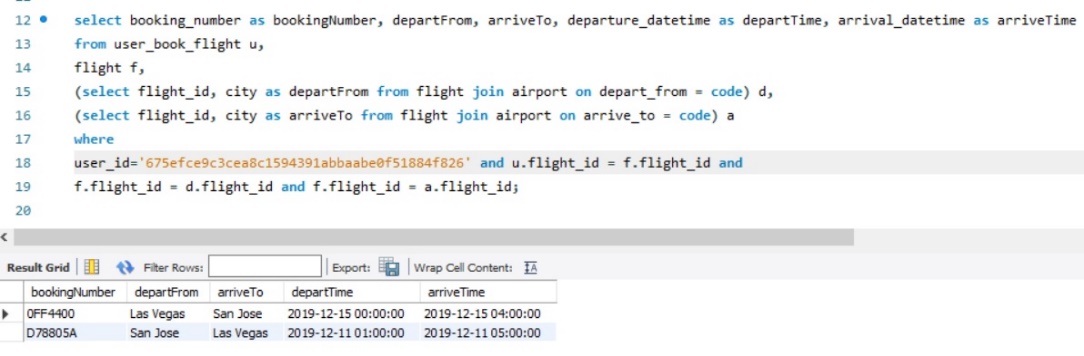
f.flight\_id = d.flight\_id and f.flight\_id = a.flight\_id;

#### **Booking number link in Flight table is to redirect to Booking Detail (see** [**Booking Detail**](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.z75oldxij3yb)**)**

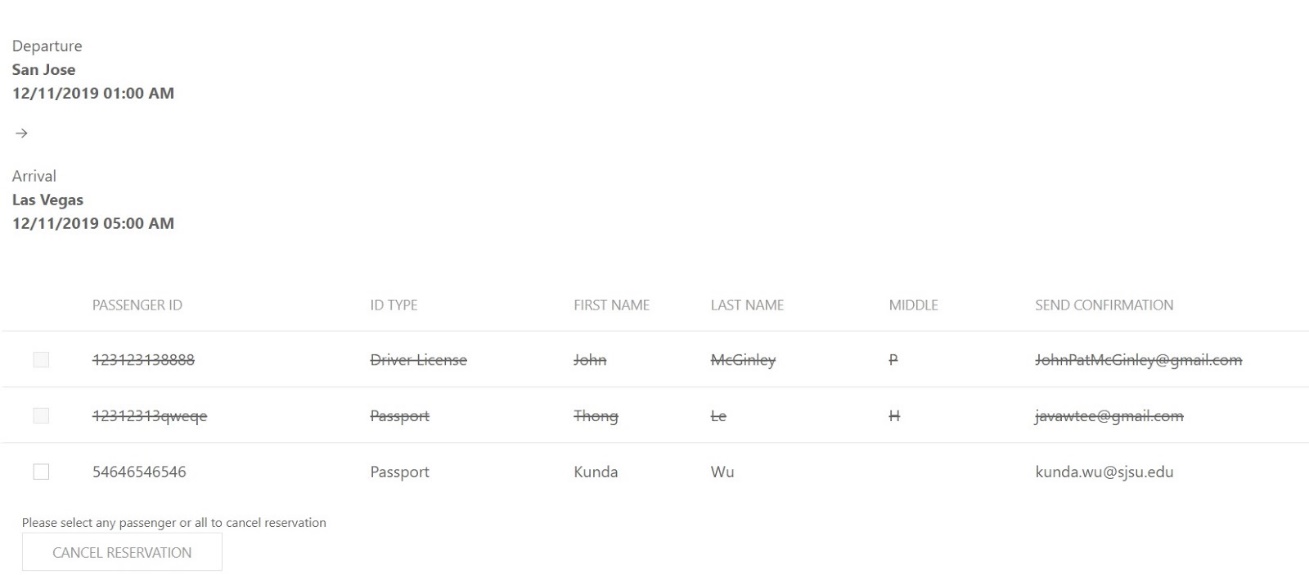


### **Results**

#### **Retrieve flights associated with authenticated user**



## **Booking Details**



React components: BookingDetail, BookingDetailPassengers, BookingDetailSummary

Express routes:

* GET /user/getbookingdetail
* POST /user/cancelbooking

Tables: booking\_passenger, passenger, user\_book\_flight, flight

### **Steps**

#### **Retrieve booking information with given booking number**

select passenger.passenger\_ID as passengerId, booking\_status as bookingStatus, confirmation\_email as confirmationEmail, ID\_Type as IDType, passenger\_firstname as firstName, passenger\_lastname as lastName, passenger\_middleinitial as middleInitial

from booking\_passenger, passenger

where booking\_number=? and

booking\_passenger.passenger\_ID = passenger.passenger\_ID;

#### **Cancel booking by selecting one or more passengers**

* 1. Line-through passenger(s)/ row(s): booking is cancelled for passenger

##### **Update booking\_status (from Normal to Cancelled) in booking\_passenger**

update booking\_passenger set booking\_status='Cancelled' where booking\_number=? ${isAll ? "" : "and passenger\_ID in (?)"};

* isAll: determine if all passengers is selected with booking number
* passenger\_ID in (<list of selected passengers>)

##### **Retrieve flight\_id from user\_book\_flight with given booking\_number**

select flight\_id, booking\_class

from user\_book\_flight, booking

where user\_id=? and booking.booking\_number=? and user\_book\_flight.booking\_number = booking.booking\_number;

##### **Update seats for flight**

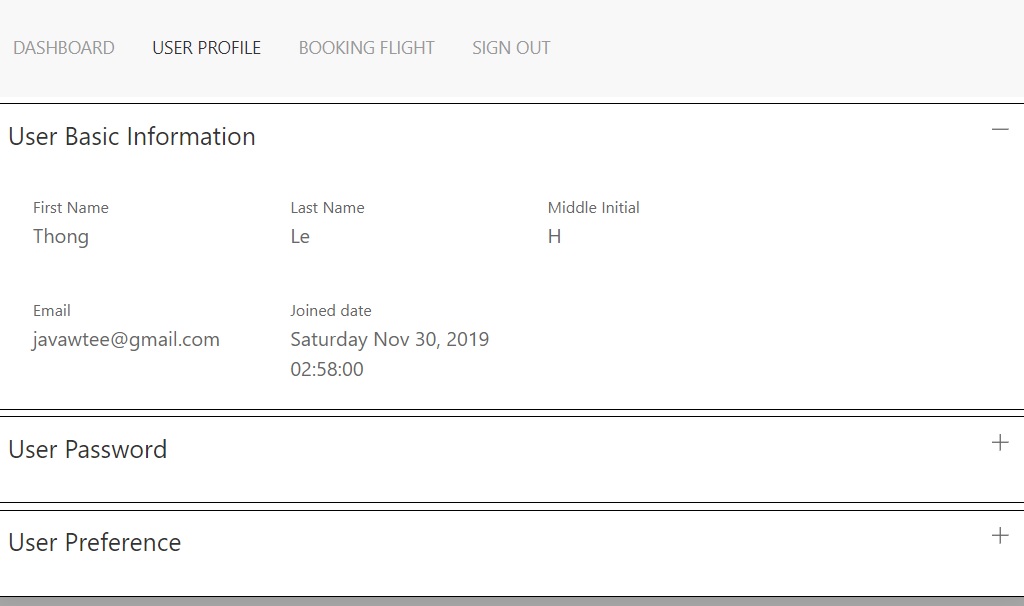
update flight set ${flightClass}\_seats=(${flightClass}\_seats + ${passengerIds.length}) where flight\_id=?;

* flightClass: determine which class of seats to update (add back to total seats)
* passengerIds.length: number of seats to add back

### **Results**

*<As shown after Booking Detail heading>*

## **User Profile**



React components: UserProfile, UserInfo, UpdatePassword, UserPreference

Express routes: see [SignIn](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.x8kr1a8wrtfi)

Tables: see [SignIn](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.x8kr1a8wrtfi)

### **Steps**

See [SignIn](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.x8kr1a8wrtfi)

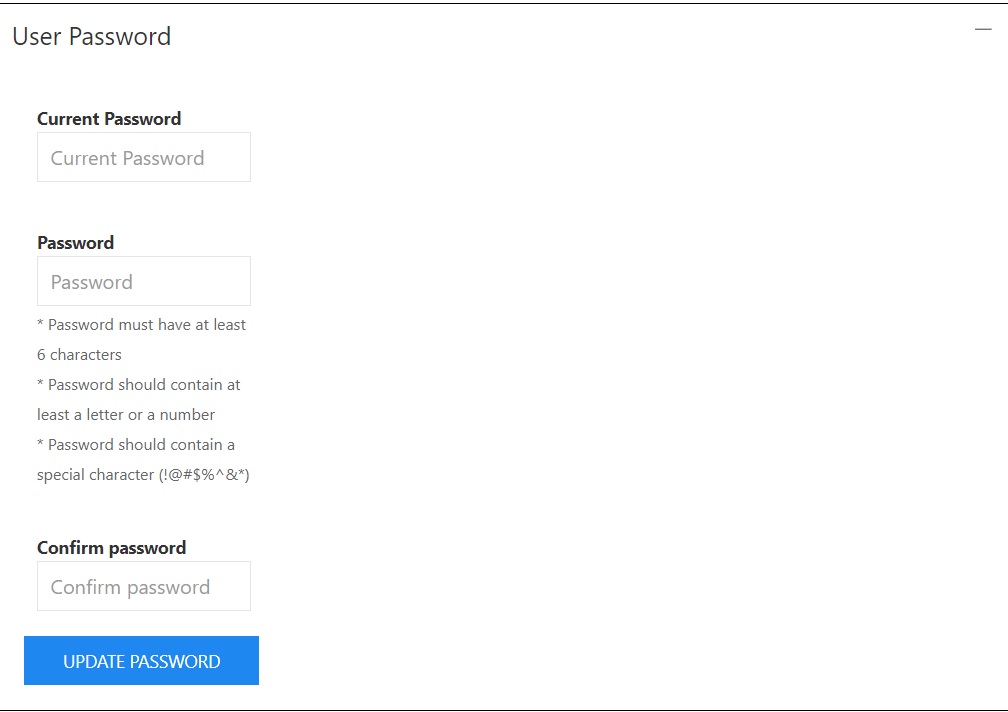
### **Results**

*<As shown after User Profile heading>*

## **User Basic Information**

See [User Profile](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.bbojrs62oe0y)

## **User Password**



React components: UpdatePassword, PasswordField

Express routes: POST /user/updatepassword

Tables: user

### **Steps**

#### **If update password from recovery link, looking up for user with corresponding recovery\_link and check if link is expired with recovery\_exp**

update user set password=sha2(?,0) where recovery\_link=?;

#### **Update password with given user\_id and given current password**

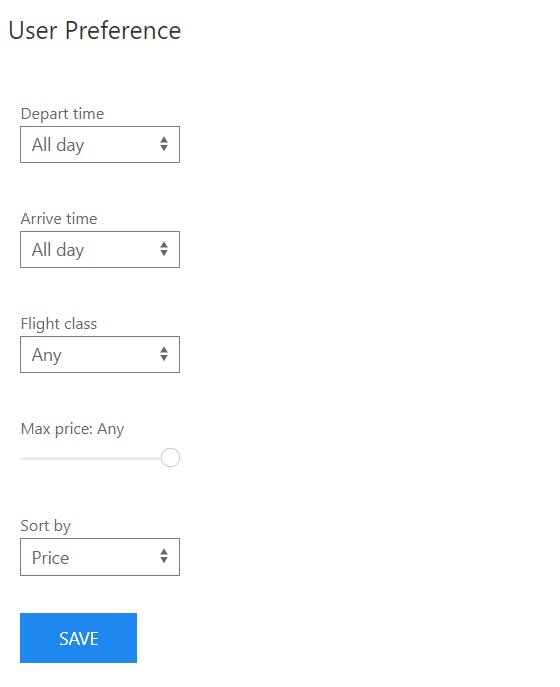
update user set password=sha2(?,0) where user\_id=? and password=sha2(?, 0)

#### **Results**

#### **Failed validation: password is too short or too simple**

#### **Successfully updated password and stay at the same page**

## **User Preference**



React components: User Preference

Express routes:

* See [SignIn](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.x8kr1a8wrtfi) for getting user preferences
* PUT /user/updatepreference

Tables: see [SignIn](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.x8kr1a8wrtfi)

### **Steps**

#### **Update all attributes in user preference with session-stored preference\_id**

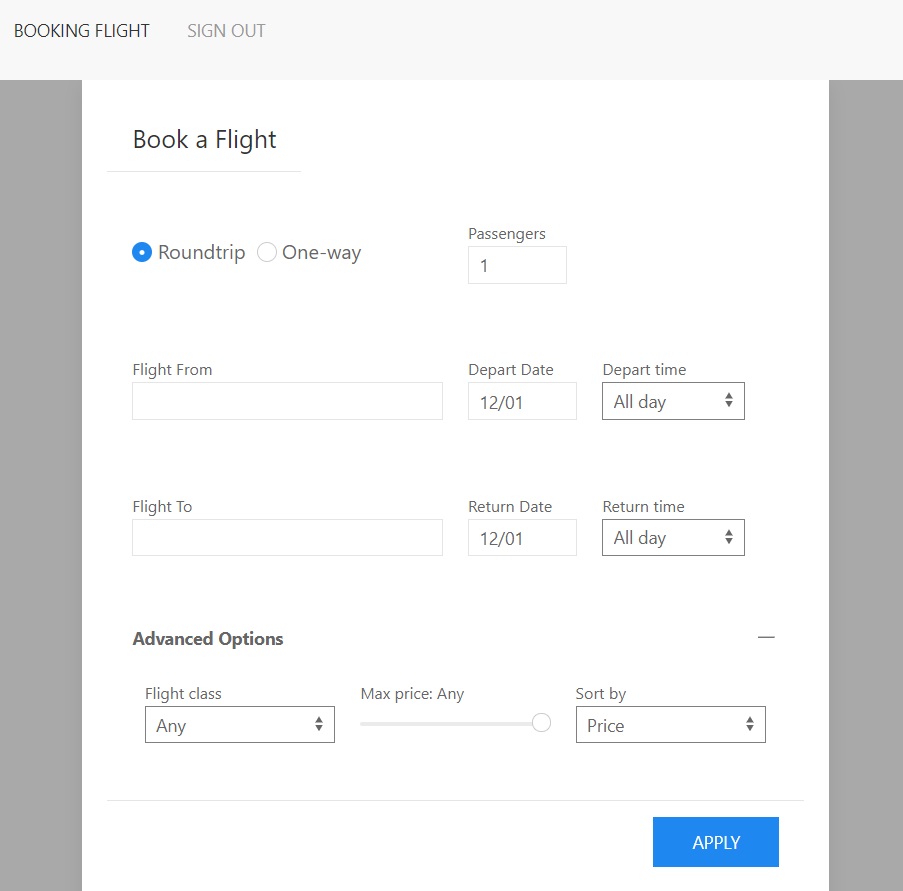
update preference set depart\_time=?, arrive\_time=?, flight\_class=?,

max\_price=?, sort\_by=? where preference\_id=?;

### **Results**

#### **Successfully updated (alert appears from top)**

## **Booking (Search)**



React components: Booking, Autocomplete, SearchField

Express routes: GET /airport

Tables: airport, preference from express session (see [SignIn](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.x8kr1a8wrtfi))

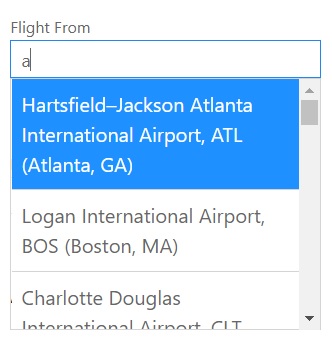
### **Steps**

#### **Fetch airport data**

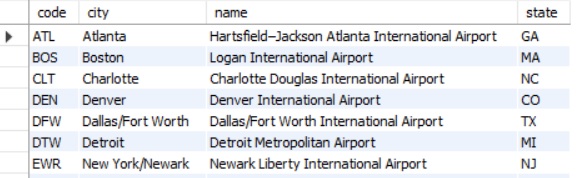
select code, name, city, state from airport, location where airport.location\_id = location.location\_id;

#### **Get list of airports for autocomplete input fields (Flight From, Flight To)**

* 1. Airport view format: name, code (city, state)



* 1. Airport table



* 1. Location table



#### **Trip type (Roundtrip, One-way)**

##### **Roundtrip**

* + 1. Departure flight
       1. Flight From: flight departs from
       2. Flight To: flight arrives to
       3. Depart Date: flight departs on date
       4. Depart Time: flight departs during time
    2. Return flight
       1. Flight departs from: Flight To
       2. Flight arrives to: Flight From
       3. Arrive Date and Time are similar to Depart Date and Time

##### **One-way**

* + 1. Disable Return Time and Return Date inputs
    2. Has no Return flight

#### **Passengers (number of passengers)**

* 1. Anticipated number of passengers to book
  2. Flight class (see below in Advanced options) and passengers determine number of seats required in SQL query

var seats = null

var maxPrice = null

*switch* (fclass) {

*case* "eco":

     seats = `and economy\_seats >= ${passengers}`

maxPrice = `and economy\_price <= ${max}`

*break*

*case* "bus":

      seats = `and business\_seats >= ${passengers}`

maxPrice = `and business\_price <= ${max}`

*break*

*default*: *// fclass = 'all'*

      seats = `and (economy\_seats >= ${passengers} or business\_seats >= ${passengers})`

maxPrice = ""

}

#### **Advanced Options**

##### **Flight class (all, economy, business)**

Determine query conditions for seats and price

##### **Max price**

Flight class and max price determine upper bound for price required in SQL query (see [Passengers](https://docs.google.com/document/d/1SnZdptshkC827HNgXN5mu1atal6rnKJ2Fsn2ol8xrso/edit#heading=h.tie56kfqoewp))

##### **Sort by**

Determine how to sort returned rows from query

var sortBy = null

*switch* (sort) {

*case* 'depTime':

     sortBy = 'time(departure\_datetime) ASC'

*break*

*case* 'arrTime':

     sortBy = 'time(arrival\_datetime) ASC'

*break*

*case* "price":

*if* (fclass === "eco") {

        sortBy = "economy\_price ASC"

      } *else* *if* (fclass === "bus") {

        sortBy = "business\_price ASC"

      } *else* {

        sortBy = "economy\_price ASC, business\_price ASC"

      }

*break*

*default*:

      sortBy = ""

}

#### **Click Search**

#### **Function to process and build request call on front-end side**

* 1. Example request call: */flight?roundtrip=y&depart=SJC|2019/12/10|any&arrive=LAS|2019/12/15|any&fclass=any&max=2000&passengers=1&sort=price*

#### **Function to process request call and build query on back-end side**

* 1. Example query:

*select flight\_id as flightId,*

*departure\_datetime as depTime,*

*arrival\_datetime as arrTime,*

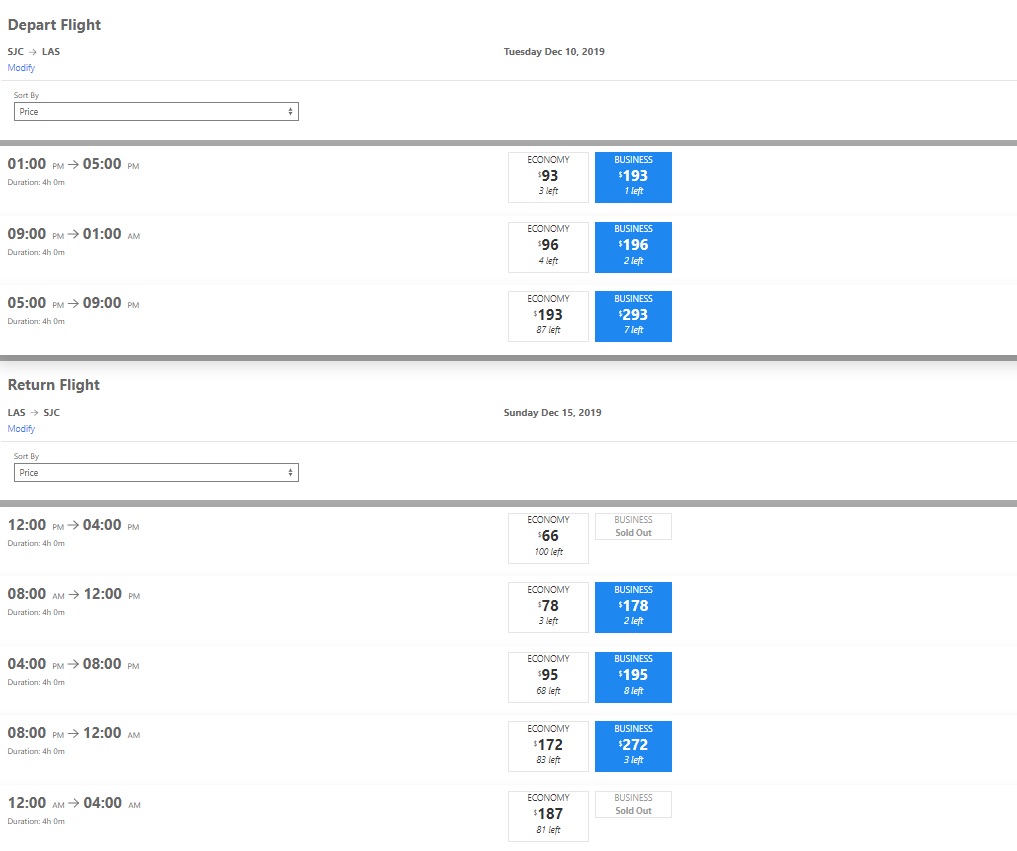
*economy\_price as ecoPrice, economy\_seats as ecoSeats,*

*business\_price as busPrice, business\_seats as busSeats*

*from flight*

*where depart\_from=? and date(departure\_datetime)=? and arrive\_to=?and (economy\_seats >= 1 or business\_seats >= 1)  order by economy\_price ASC, business\_price ASC;*

## **Booking (Search Results)**



React components: Booking, FlightSearchResults, ResultHeader, FlightSummary, ResultItem

Express routes: *Not applicable*

Tables: *Not applicable*

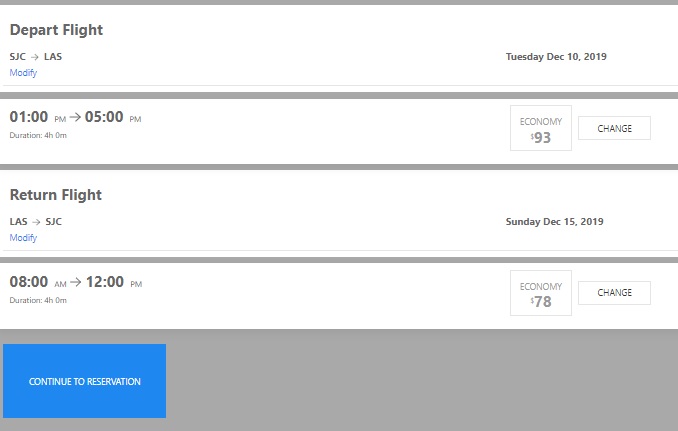
### **Steps**

#### **User can refine search by clicking on Modify link**

#### **User can select different Sort By (Depart Time, Arrive Time, Price)**

#### **Select a flight by clicking on Economy or Business button**

#### **Complete selecting flight, Continue to Reservation button will appear**



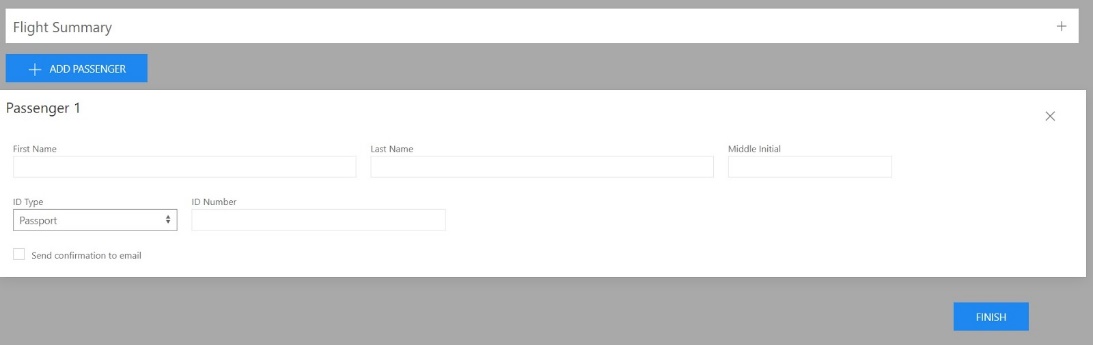
User can re-select flight by clicking on the Change button

### **Results**

1. If there is no flight found matching search inputs, see Step 1 to go back and refine Search
2. If Search for Roundtrip and there is no flight found for either Depart Flight or Return Flight, see Step 1 to back and refine Search.

Reason: If selecting flight process is not complete, Continue to Reservation will not appear to finish booking

## **Booking (Reservation)**



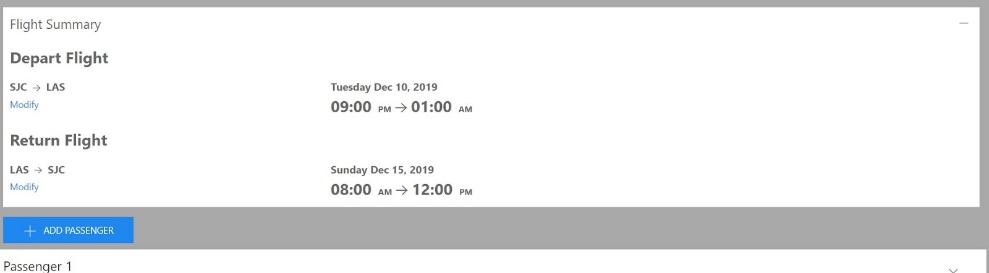
React components: Booking, FlightReservationMain, ResvPassenger

Express routes: *Not applicable*

Tables: *Not applicable*

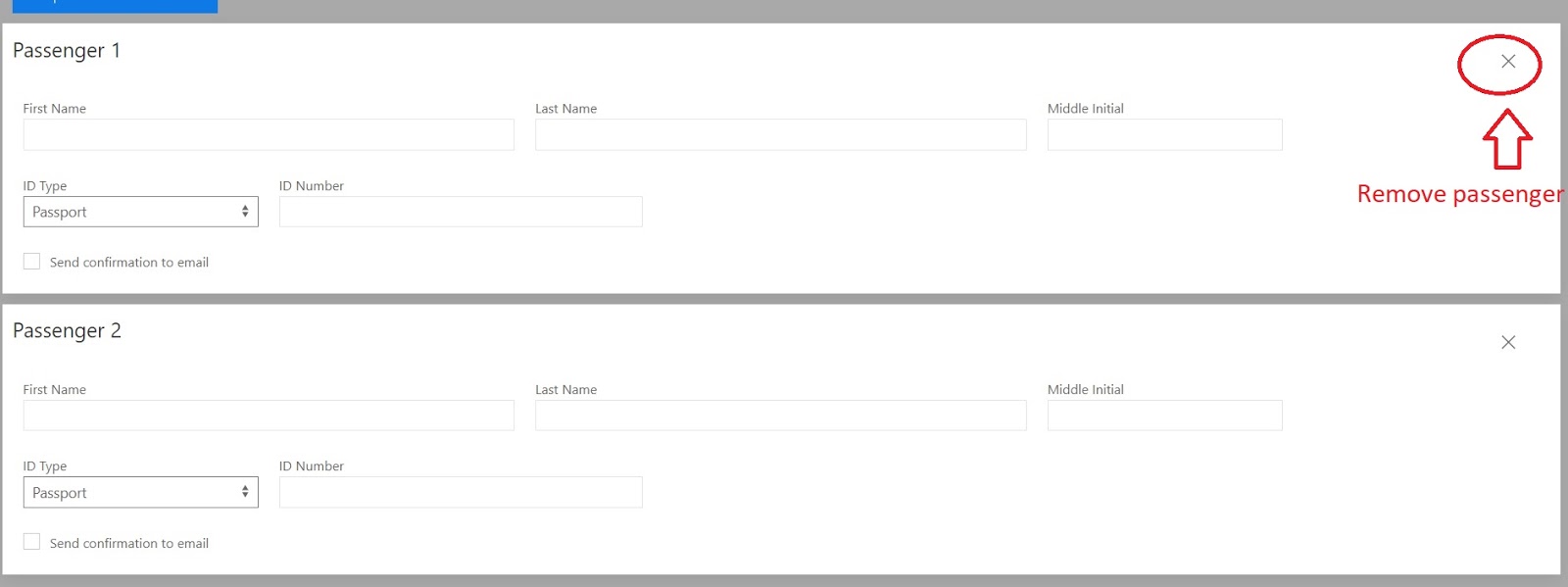
### **Steps**

#### **Click on Flight Summary to view summary of selected flight(s)**



#### **Click Add Passenger button to add one more passenger to book**

(Remove by following red arrow in the same screenshot)



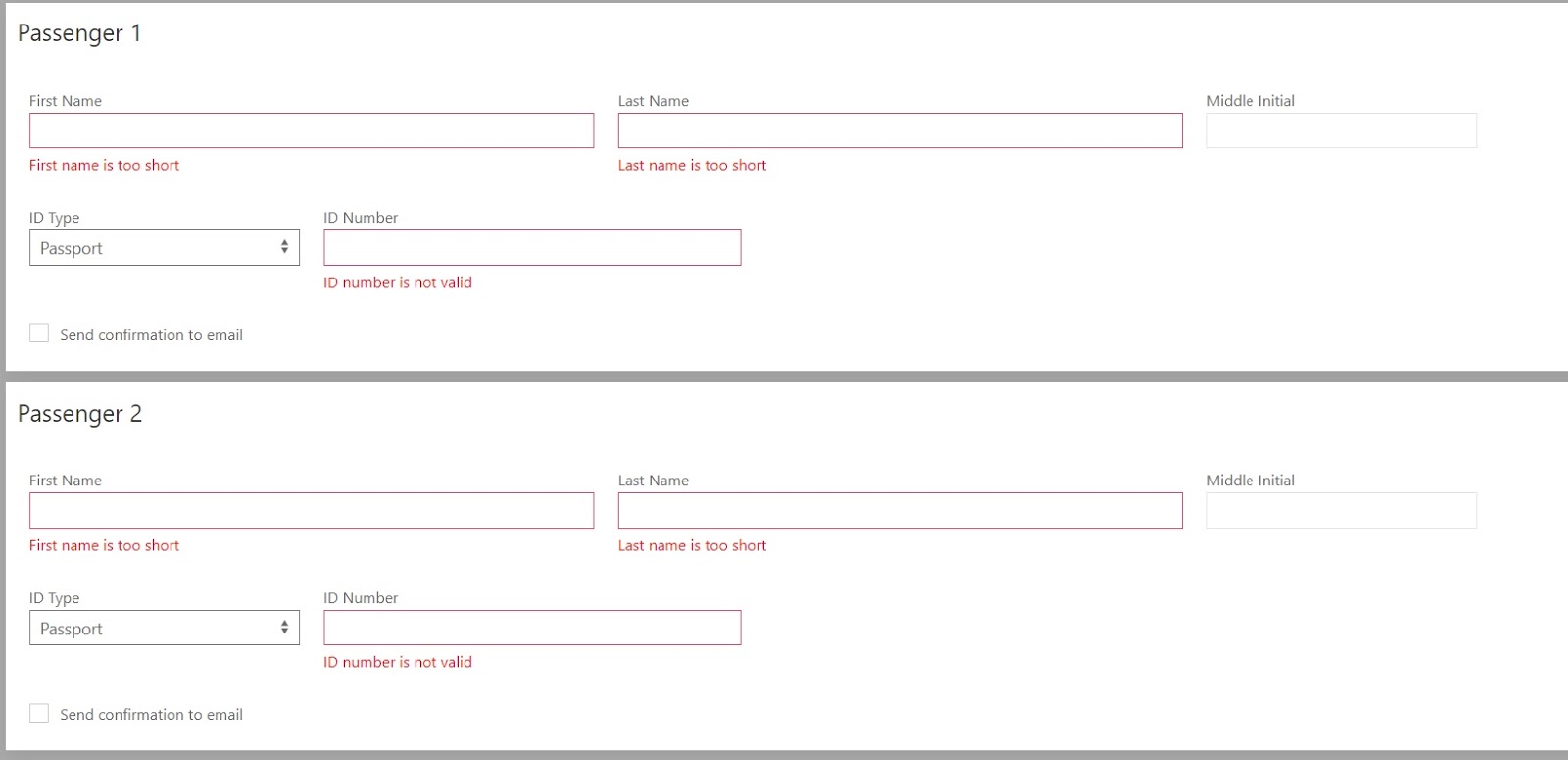
User can add the maximum of 5 passengers or the selected flight(s) have enough seats

#### **Click on Send confirmation (optional) to receive Confirmation Email**

#### **After complete filling out inputs for passenger, click Finish button to complete Reservation (skip payment process, assume all payment is valid)**

## **Results**

#### **Failed validation**



#### **Successfully booked the flight(s) (alert appears from top).**

Then, redirect to Dashboard

# **Setup procedures**

## **Overview**

This document is to show procedures to set up and run our DB web-based application with 3-tier architecture

## **Requirements**

* Download and install [Node.js](https://nodejs.org/en/)
* Download and install [Git](https://git-scm.com/downloads)
* Install and configure [Apache24](https://github.com/javawtee/CS157A-01-Team6/tree/master/Setup/apache24)
* Download and install [MySQL](https://dev.mysql.com/downloads/mysql/)
* Download and install [MySQL Workbench](https://dev.mysql.com/downloads/workbench/)
* Download [MySQL script](https://github.com/javawtee/CS157A-01-Team6/blob/master/Setup/mysql/complete_project_database.sql)

## **Steps**

1. Clone GitHub repository for our project

* Open cmd (for Windows) or terminal (for Linux-based OS)

git clone https://github.com/javawtee/CS157A-01-Team6.git <local\_repo\_name(optional)>

1. Set up database

* Open MySQL Workbench
* Create and open new connection
* Create new SQL query
* Add and run MySQL script in *<local\_repo\_name>/Setup/mysql/complete\_project\_database\_v2.sql* to create schema and tables with sample data

1. Configure mysql connector

* Go to <local\_repo\_name>/server/api/configs
* Open mysql\_config.js
* Change:
  + User: *<mysql server user>*
  + Password: *<mysql server user’s password>*
  + Database: 'cs157a\_project'

1. Set up Server

cd <local\_repo\_path>/server

npm install

1. Run Server

cd <local\_repo\_path>/server

npm start

1. Open on web browser: [http://localhost:5000](http://localhost)

# CONCLUSION

## **Lessons learned**

|  |  |
| --- | --- |
| Thong Le | * 3-tier architecture * Efficiency in database design will make latter works easier * How to manage and work with a small-sized team * Breakdown tasks into smaller ones and prioritize each * Mentoring team members on web technology |
| Kunda Wu | * How to user React handle front-end and express handle back-end * How to integrate the change from database to web server * How a 3-tier architecture can play a significant part in web server programming * How to conduct a sql statement to initialize and communicate between server and database |
| John McGinley | * I learned about React and how to use it on the front-end * I learned about axios and how to pass data from the back and front-ends of an application * Learned how to integrate a database into a project * Learned how to separate functionality properly from a backend a frontend * Learned a lot about available npm modules and how to properly hash user information * Learned how to use MySQL workbench and how to link it to a project |

## **Future Improvement**

There is still plenty of room to develop and improve our existing project.

* First of all, we planned to deploy and run it as professional project. However, due to other projects and limited time we can dedicate to this. We may still do so in the future
* We would like to advance user profiles, by allowing users to upload photos and save payment information
* We also like to dedicate significant time to creating a streamlined GUI with a more colorful palette to appeal to users
* We would like to thoroughly debug and test our project to ensure that it can handle all edge cases.

Ideally, our goal is simply to provide our users with streamlined airline booking system, and we feel that we have done a good job implementing one within the given timeframe.