

# **Road Trip Planner**

Authors: Zainab Siddiqui, Sean Boucher, Linghui Pan, Lamianoor Zinia

## Table of Contents

<b>Problem Statement</b>	<b>2</b>
<b>System Requirements</b>	<b>2</b>
<b>Conceptual Database Design - ER Diagram</b>	<b>3</b>
<b>Functional Requirements</b>	<b>3</b>
<b>Logical Database Design</b>	<b>4</b>
<b>Application Programs</b>	<b>5</b>
<b>User Interface Design</b>	<b>10</b>
<b>Implementation and Testing</b>	<b>16</b>
<b>Code Listing</b>	<b>16</b>
<b>Sample Output</b>	<b>16</b>

## **Problem Statement**

The application that we are proposing for our database project is a road trip planner written in Java and JavaFX. Our application is geared towards travel enthusiasts who like to stay on top of their plans or keep multiple travel itineraries. Users will be allowed to create and organize their itineraries with all of the pertinent information - hotel and car rental bookings, activities, etc - as well as view and update them as they wish. Integral to our road trip management system will be our MySQL database - it will allow us to store, retrieve, update, and remove road trip plans, as well as set up user authentication. We believe an application like this should remove a lot of the headache that comes with having to plan a vacation in your head and will instead make the whole process a lot more organized and exciting.

## **System Requirements**

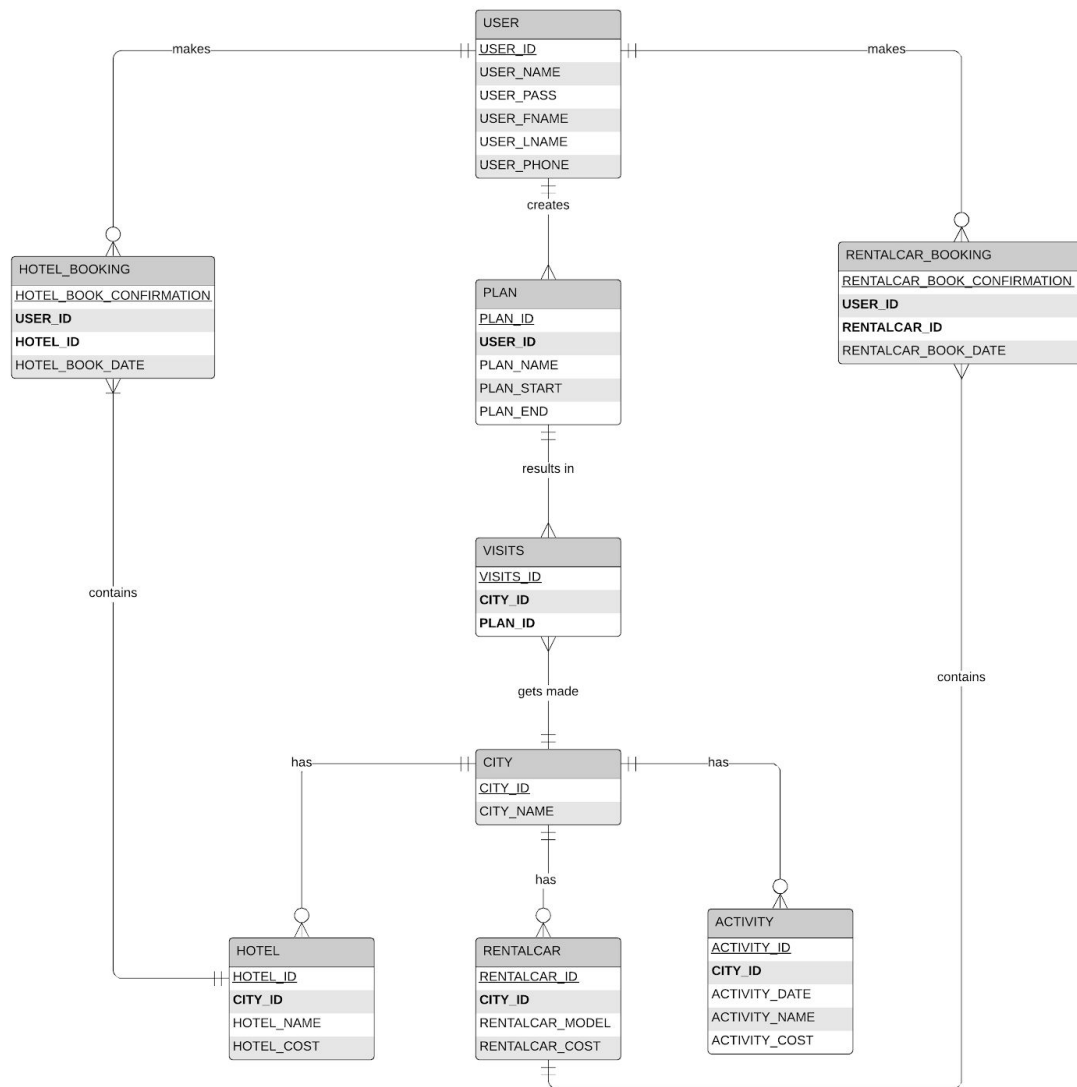
This road trip management system will:

- Include user authentication
- Feature an interface for users to be able to see their created road trip plans
- Provide options for users to create, modify, update, and delete one or more plans
- Allow users to book hotel accommodation and car rentals for the cities they plan to visit within each plan
- Allow users to select US cities from a dropdown
- Allow users to select activities they plan to do in each city

It will not:

- Suggest places of interest for the user
- Allow users to share their travel plans with others
- Notify users of upcoming plans

## Conceptual Database Design - ER Diagram



## Functional Requirements

Below is a breakdown of the tasks required for the features defined in system requirements.

- Allow user to sign up for an account (input needed: name (string), password (alphanumeric) - user validation operations will be conducted)
- Allow user to log in and log out (user authentication operations conducted)
- Allow user to change password (input needed: new alphanumeric, update operation conducted)
- Allow user to create a new road trip plan (inputs needed: name (string), start date, end date - create operation)

- Allow user to see all created plans on front page (read operation)
- Allow user to delete plan (delete operation)
- Allow user to modify plan - like name or range of dates (update operation)
- Allow user to upload, update, or remove a map image of their proposed road trip (input needed: image (not mandatory), create, update, and delete operations)
- Allow user to record cities they will be visiting (inputs needed: name (string), start date, end date (must ensure that this range of dates falls under the general plan's range of dates) - create operation conducted)
- For each city:
  - Allow user to add one or more hotels, as well as add their respective booking confirmation (inputs needed: hotel name (string), booking confirmation (alphanumeric) - create operation conducted)
  - Allow user to add one or more car rentals, as well as add their respective booking confirmation (inputs needed: hotel name (string), booking confirmation (alphanumeric) - create operation conducted)
  - Allow user to add zero or more activities for the chosen city (inputs needed: activity name (string), start time and end time (to ensure multiple activities aren't scheduled for the same time) - create operation conducted)
  - Allow user to view all of the above information they previously inputted (read operations)

## Logical Database Design

**USER** (USER\_ID INT(42), USER\_NAME CHAR(128), USER\_FNAME CHAR(128), USER\_LNAME CHAR(128), USER\_PHONE INT CHAR(10))  
 PRIMARY KEY: USER\_ID

**PLAN** (PLAN\_ID INT(42), USER\_ID INT(42), PLAN\_NAME CHAR(128), PLAN\_START DATE(), PLAN\_END DATE())  
 PRIMARY KEY: PLAN\_ID  
 FOREIGN KEY: USER\_ID REFERENCES USER

**VISITS** (VISITS\_ID INT(42), USER\_ID INT(42), PLAN\_ID (42))  
 PRIMARY KEY: VISITS\_ID  
 FOREIGN KEY: USER\_ID REFERENCES USER  
 FOREIGN KEY: PLAN\_ID REFERENCES PLAN

**CITY** (CITY\_ID INT(42), CITY\_NAME CHAR(128))  
 PRIMARY KEY: CITY\_ID

**HOTEL** (HOTEL\_ID INT(42), CITY\_ID INT(42), HOTEL\_NAME CHAR(128), HOTEL\_COST CHAR(10))

PRIMARY KEY: HOTEL\_ID  
FOREIGN KEY: CITY\_ID REFERENCES CITY

**RENTALCAR** (RENTALCAR\_ID INT(42), CITY\_ID INT(42), RENTALCAR\_MODEL CHAR(128), RENTALCAR\_COST CHAR(10))  
PRIMARY KEY: RENTALCAR\_ID  
FOREIGN KEY: CITY\_ID REFERENCES CITY

**ACTIVITY** (ACTIVITY\_ID INT(42), CITY\_ID INT(42), ACTIVITY\_DATE DATE(), ACTIVITY\_NAME CHAR(128), ACTIVITY\_COST CHAR(10))  
PRIMARY KEY: ACTIVITY\_ID  
FOREIGN KEY: CITY\_ID REFERENCES CITY

**HOTEL\_BOOKING** (HOTEL\_BOOK\_CONFIRMATION INT(42), HOTEL\_ID INT(42), USER\_ID INT(42), HOTEL\_BOOK\_DATE DATE())  
PRIMARY KEY: HOTEL\_BOOK\_CONFIRMATION  
FOREIGN KEY: HOTEL\_ID REFERENCES HOTEL  
FOREIGN KEY: USER\_ID REFERENCES USER

**RENTALCAR\_BOOKING** (RENTALCAR\_BOOK\_CONFIRMATION INT(42), RENTALCAR\_ID INT(42), USER\_ID INT(42), RENTALCAR\_BOOK\_DATE DATE())  
PRIMARY KEY: RENTALCAR\_BOOK\_CONFIRMATION  
FOREIGN KEY: RENTALCAR\_ID REFERENCES RENTALCAR  
FOREIGNKEY: USER\_ID REFERENCES USER

## Application Programs

Our application consists of two major folders. The first folder, application, contains all of our controllers and associated FXML files. The second is a package called dbconnection, which contains two files which handle configuring our database. This section will primarily deal with our application files.

For each screen/window of our application, there is a controller and a FXML file. We used SceneBuilder to create the GUI, which automatically generates the required FXML code. The logic and main code occurs in the controller files.

Main.java

- This class handles what happens on startup. For our application, it loads in our login page:

```

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        // Loads in the login page for the application at startup
        Parent root = FXMLLoader.load(getClass().getResource("Login.fxml"));
        primaryStage.setTitle("Road Trip Planner");
        primaryStage.setScene(new Scene(root, width: 800, height: 600));
        primaryStage.show();
        primaryStage.setResizable(false);
    }

    public static void main(String[] args) { launch(args); }
}

```

### LoginController.java

- This handles the login screen of our application. It has three functions:
  - Initialize "initializes" our database handler.

```

@Override
public void initialize(URL location, ResourceBundle resources) { handler = new DBHandler(); }

```

- loginAction(ActionEvent event) is a function that is fired when user clicks the login button. It checks our database to see if the given credentials are stored; if they are, it loads up the next screen, which is the Dashboard. If not, it sends an error message. Queries:
  - String loginQuery = "SELECT \* FROM user WHERE username=? AND pass=?";
- signUpAction(ActionEvent event) fires when user clicks the sign up button. All it does is change screens to the Sign Up page.

### SignUpController.java

- This controller handles the sign up screen of our application. It has three functions:
  - Initialize "initializes" our database handler.

```

@Override
public void initialize(URL location, ResourceBundle resources) { handler = new DBHandler(); }

```

- createAccount(ActionEvent event) is fired when the user clicks on the Create Account button. It checks first to see if all of the fields have been filled out. If not, it sends out an error message. Then, it checks to see if the passwords given match. If not, it sends out an error message. If everything is okay, it executes an insert query to our user table. Queries:
  - String insert = "INSERT INTO user(username, pass, fname, lname, phone) VALUES (?,?,?,?,?)";

- Note: question marks are later replaced with the help of a prepared statement.
- loginAction(ActionEvent event) is fired when the user clicks the back button. It redirects the user back to the login page without doing anything else.

#### DashboardController.java

- This handles the dashboard of our application, which the user sees when they log in. It displays their already created plans and allows them to create a new one. There are 4 main functions:
  - initialize(URL location, ResourceBundle resources) is fired immediately as this screen is loaded. It initializes our database handler as well as uses an insert query to load and display the plans the user has already created. These plans are in the form of clickable buttons that allow you to see more information. It also displays a count of road trip plans. Lastly, it displays the username of the user within the welcome message using the function setUsername(String user).  
Queries:
    - String getPlans = "SELECT \* FROM plan WHERE user\_id=?";
  - setUsername(String user) is called from within the initialize function. All it does is set the textfield at the top of the page with the user's username.
  - createNewPlan(ActionEvent event) is fired once the user clicks the Create New button. It just loads the PlanCreation.fxml window and transfers control over to PlanCreationController.
  - logOut(ActionEvent event) is fired once the user clicks the Log Out button. It returns the user back to the login screen.

#### PlanCreationController.java

- This class handles the form used to create a new plan for a given user. There are 3 main functions:
  - initialize(URL location, ResourceBundle resources) initializes our database handler. It also populates our dropdown options for the cities' fields using a SELECT query on our CITY table. Queries used:
    - String getCities = "select name from city";
  - addPlan(ActionEvent event) is fired once the user clicks on the Add Plan button. It inserts the information from text fields into our PLAN table. It also saves the ID of the plan just made in a global variable, as well as the three cities chosen. Lastly, it inserts entries into our VISITS table for each city within the plan.  
Queries:
    - String insertPlan = "INSERT INTO plan(name, start, end, user\_id) VALUES (?,?,?)";
    - String getplanID = "select id from plan where name = ? and user\_id =?";
    - String getCity1ID = "select id from city where name = ?";
    - String getCity2ID = "select id from city where name = ?";
    - String getCity3ID = "select id from city where name = ?";



- `String insertVisits = "INSERT INTO visits(plan_id, city_id) VALUES (?,?)";`
- `onBack(ActionEvent event)` is a function that fires when user clicks on the back button. It simply redirects the user back to the dashboard without doing anything else.

#### PlanViewController.java

- This class handles the screen and logic for when the user clicks on a specific plan to view more information. It displays the plan's name, start, and end date, as well as the cities associated with it, with options to view each city in more detail. It also includes a delete button so that the user may delete the plan entirely. There are 6 main functions:
  - `initialize(URL location, ResourceBundle resources)` initializes our DB handler and also executes queries to grab the current plan's name, start and end date, as well as associated cities. Queries:
    - `String planNameQuery = "select * from plan where name = ? and user_id = ?";`
    - `String grabCitiesQuery = "select city_id from visits where plan_id = ?";`
    - `String city1Query = "select name from city where id = ?";` (\*similar query executed for cities 2 and 3\*)
  - `deletePlan(ActionEvent event)` is fired when the user clicks the Delete Plan button. It executes a deletion query and returns the user back to the dashboard. Queries:
    - `String planDeletionQuery = "DELETE FROM plan where id = ?";`
  - `onBack(ActionEvent event)` is a function that fires when user clicks on the back button. It simply redirects the user back to the dashboard without doing anything else.
  - `viewCity1(ActionEvent event)` hides the current window, and loads in the details for city 1. A similar function is included for both cities 2 and 3.

#### CityViewController.java

- This is the controller class for the screen that the user sees once they click on a city to see more details for it. It includes hotel/car booking information, as well as selected activities. It has five main functions:
  - `initialize(URL location, ResourceBundle resources)` initializes our DB handler, and sets the city's name on the top of the page. It also grabs any hotel or car rental bookings that exist for the user and that given city and displays them. It also grabs any existing activities. Queries:
    - `String grabHotelBooking = "select * from hotel_booking where user_id = ? and hotel_id in (select id from hotel where city_id = ?);";`
      - If rows found, execute the following query:
        - `String grabHotelName = "select name from hotel where id = ?";`
    - `String grabCarBooking = "select * from rentalcar_booking where user_id = ? and rentalcar_id in (select id from rentalcar where city_id = ?);";`

- If rows found, execute the following query:
  - String grabCarName = "select model from rentalcar where id = ?";
    - String grabActivities = "select \* from activity where city\_id = ?";
  - bookHotel(ActionEvent event) is fired when the user clicks on the Add/Change Booking for hotels in the displayed city. It changes the screen to a list of available hotels.
  - bookCar(ActionEvent event) is fired when the user clicks on the Add/Change Booking for car rentals in the displayed city. It changes the screen to a list of available car rentals.
  - addActivity(ActionEvent event) is fired when the user clicks on the Add Activities button and changes the screen to the Activity Selection form.
  - onBack(ActionEvent event) is fired when the user clicks the back button and changes the screen back to the PlanView page.

#### HotelBookingController.java

- This class handles the listing of available hotels in a given city. It provides the user with the option to book them. There are six main functions:
  - initialize(URL location, ResourceBundle resources) initializes the DB handler and populates the page with a list of hotels available to book in the city. It also displays the associated costs of booking each hotel per night. Queries;
    - String grabHotelsQuery = "select \* from hotel where city\_id = ?";
  - bookHotel1(ActionEvent event) is fired when the user clicks on the button associated with booking the first hotel. It calls the function checkBookingExistsAndUpdate(int hotelID), where the hotelID is the ID of the first hotel. Similar functions exist for the second and third hotel.
  - checkBookingExistsAndUpdate(int hotelID) is fired anytime a user tries to book a hotel. It uses a select query to grab existing hotel bookings for the given user in the given city, if there are any. If any results are returned, it updates the hotel booking to what the user most recently selected. If there are no results, it inserts a new hotel booking into the HOTEL\_BOOKING table. Queries:
    - String updateHotelQuery = "UPDATE hotel\_booking SET confirmation = ?, date = ?, hotel\_id = ? WHERE confirmation = ?";
    - String insertBooking = "INSERT INTO hotel\_booking (confirmation, date, hotel\_id, user\_id) VALUES (?,?,?,?)";
  - randomInt(int digits) returns a random number given a number of digits. This function has been borrowed from <https://stackoverflow.com/a/37216930>. It is used to generate a confirmation number with 8 digits for hotel and rental car bookings.

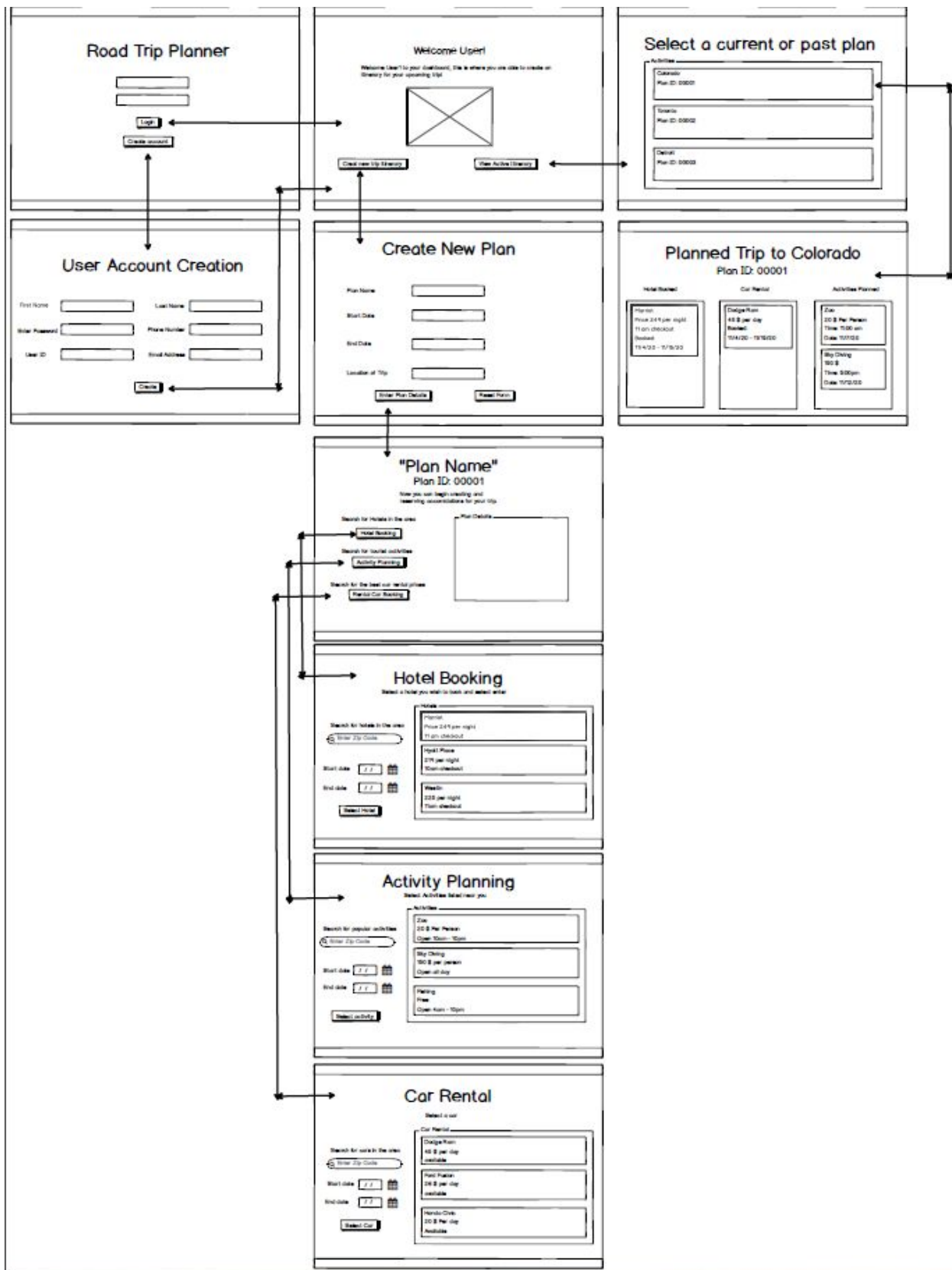
#### RentalCarBookingController.java

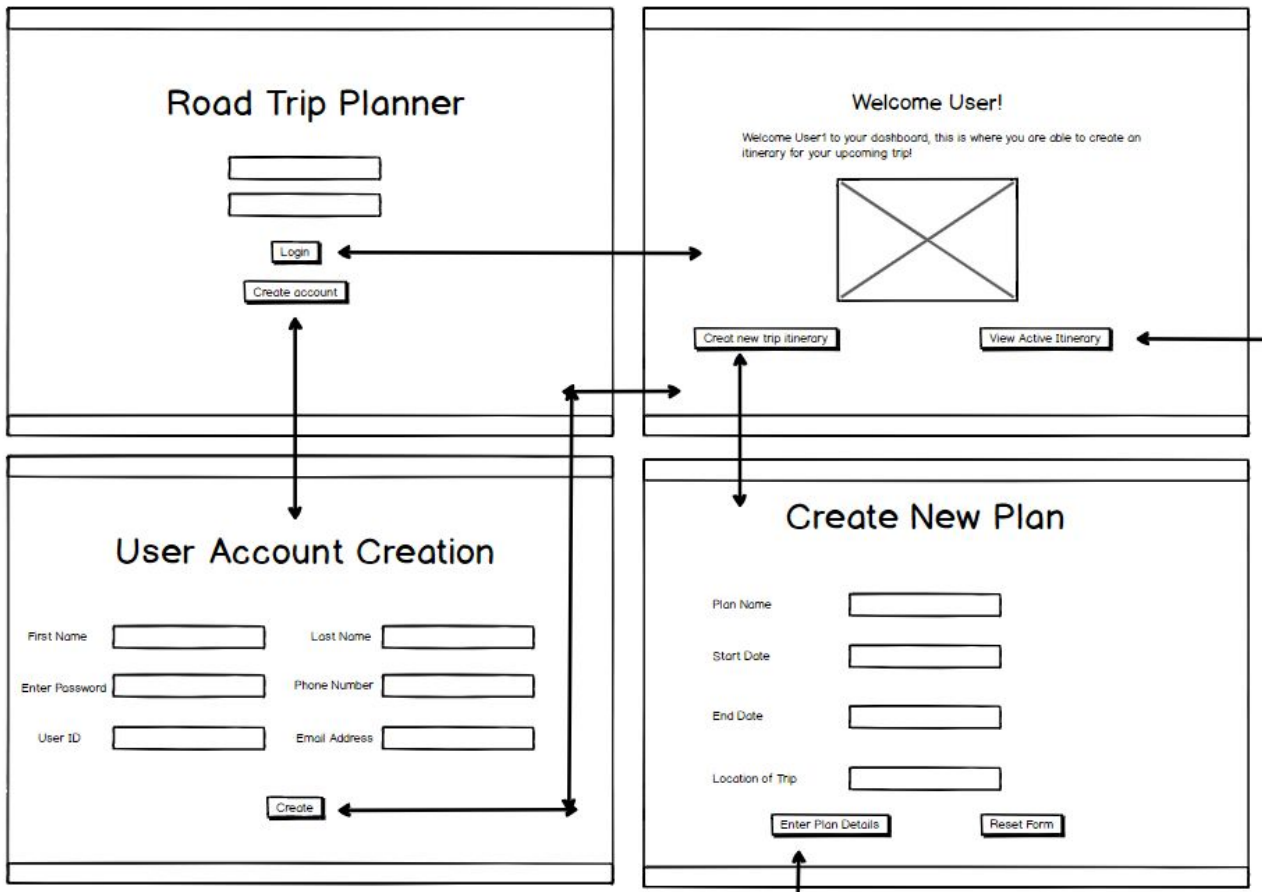
- This class has similar functions for booking rental cars in a given city. It follows the outline of HotelBookingController.java, just with car rental data rather than hotels.

### ActivitySelection.java

- This class handles the form used to add activities for a given city. User is prompted for activity name, cost, and date, and an insert query is used to insert this information into ACTIVITIES. There are four main functions:
  - initialize(URL location, ResourceBundle resources) initializes our DB handler and displays the city name at the top of the page.
  - addActivity(ActionEvent event) is fired when the user clicks the Add Activity button. Queries:
    - `String insertActivity = "INSERT INTO activity(date, name, cost, city_id) VALUES (?, ?, ?, ?)";`
  - addAnother(ActionEvent event) simply clears the text fields so the user may enter another activity of their choosing.
  - onBack(ActionEvent event) is fired when the user clicks the back button and returns the user back to the CityView without doing anything else.

### User Interface Design





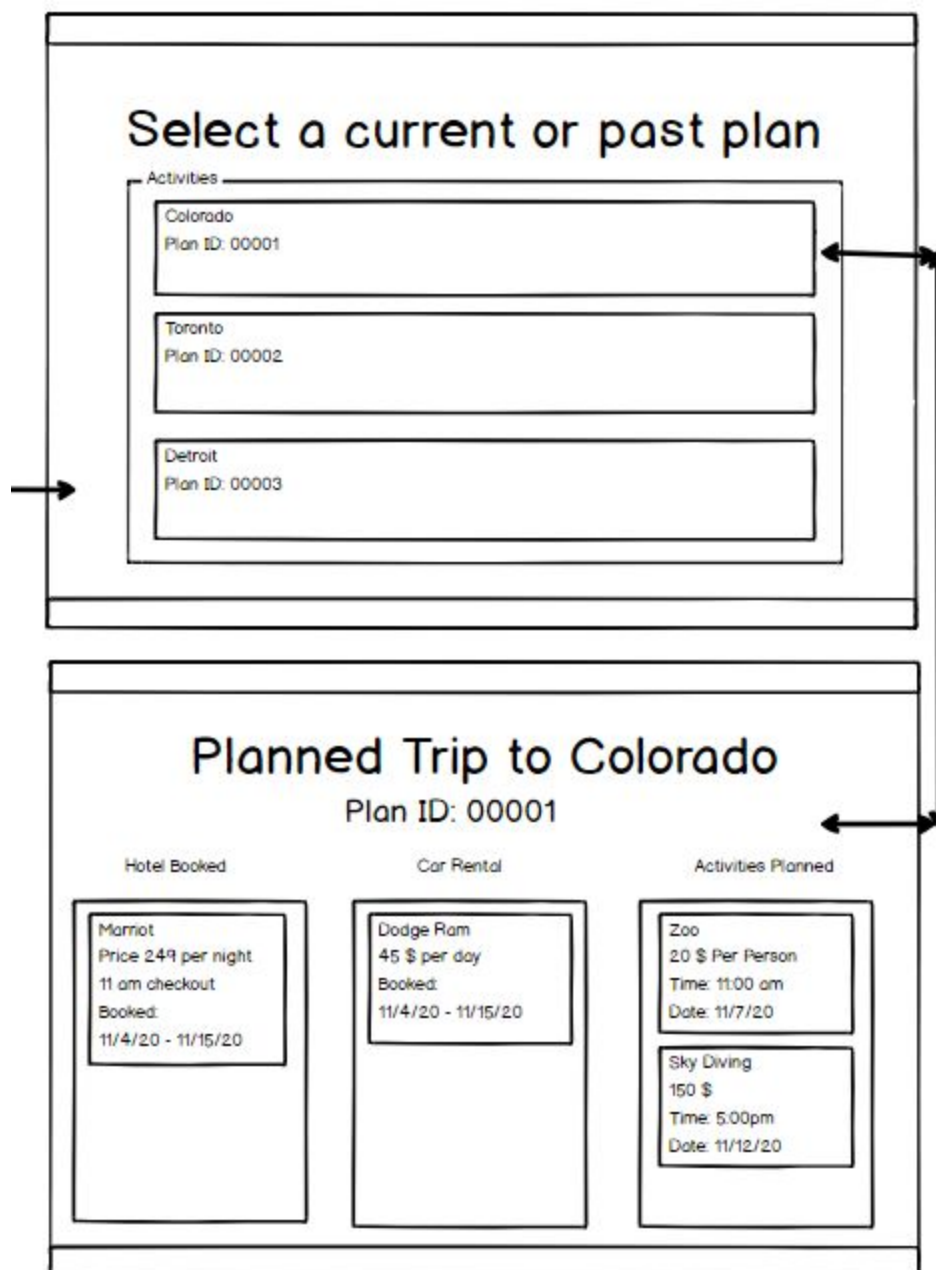
### User Account Creation Page

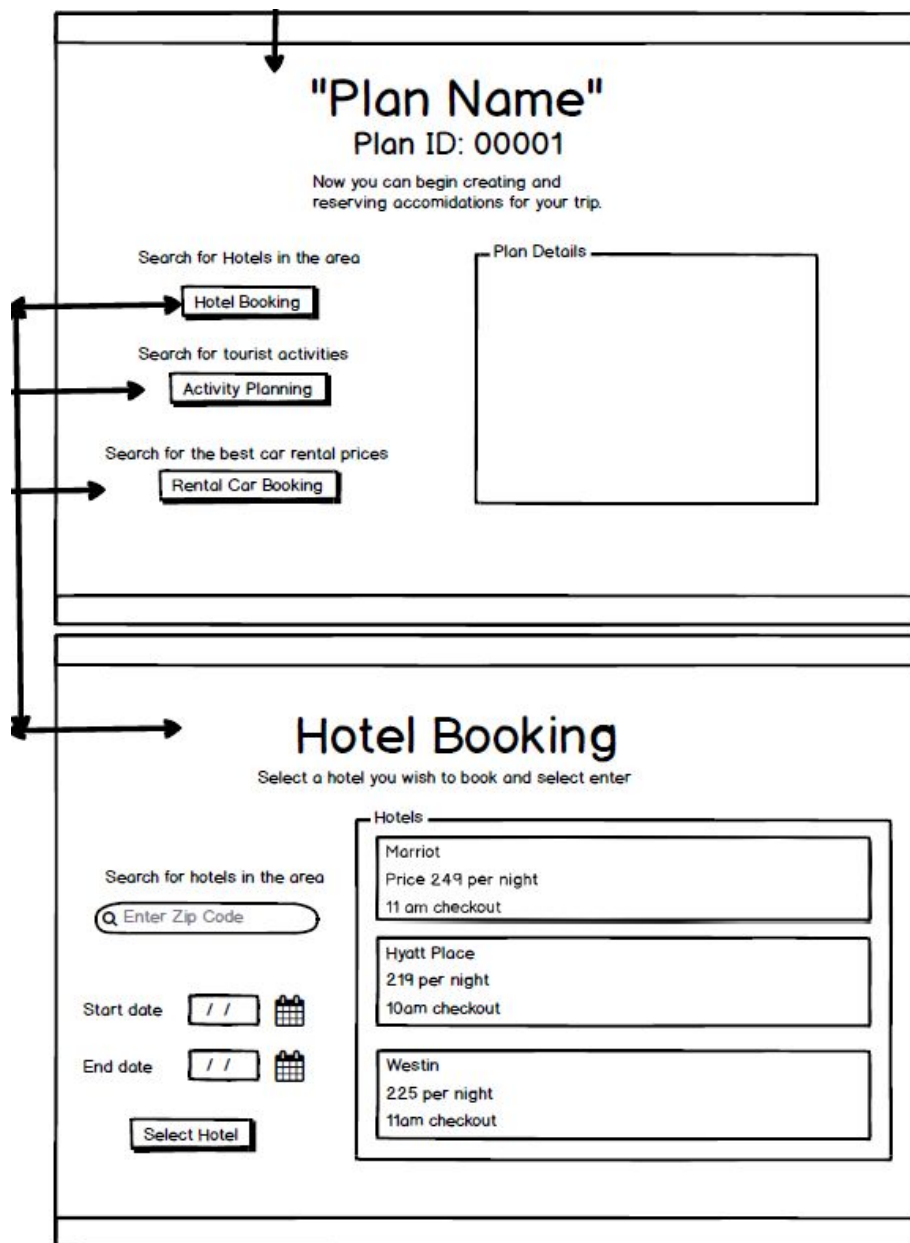
USER: USER\_ID, USER\_EMAIL, USER\_PASS, USER\_FNAME, USER\_LNAME

All of these attributes in the USER table are being modified on this page.

### Create New Plan

PLAN: PLAN\_ID is being created. PLAN\_NAME, PLAN\_START, PLAN\_END, PLAN\_MAP are all being modified on this page.





### Hotel Booking Page

HOTEL: HOTEL\_ID, HOTEL\_BOOK\_DATE, HOTEL\_BOOK\_CONFIRMATION will be modified on this page.

## Activity Planning

Select Activities listed near you

Search for popular activities

Q Enter Zip Code

Start time / /

End Time / /

Start date / /

End date / /

Select activity

Activities

Zoo

20 \$ Per Person

Open 10am - 10pm

Sky Diving

150 \$ per person

Open all day

Fishing

Free

Open 4am - 10pm

## Car Rental

Select a car

Search for cars in the area

Q Enter Zip Code

Start date / /

End date / /

Select Car

Car Rental

Dodge Ram

45 \$ per day

available

Ford Fusion

26 \$ per day

available

Honda Civic

20 \$ Per day

Available

### Activity Planning Page

ACTIVITY: ACTIVITY\_ID will be created. ACTIVITY\_STIME, ACTIVITY\_ETIME, ACTIVITY\_DATE, ACTIVITY\_NAME, ACTIVITY\_COST will all be modified on this page.

### Car Rental Page

RENTAL CAR: RENTALCAR\_MODEL, RENTALCAR\_COST will be modified on this page.  
 RENTAL CAR BOOKING: RENTALCAR\_ID will be created. RENTALCAR\_BOOK\_DATE and RENTALCAR\_BOOK\_CONFIRMATION will be modified on this page.



## Implementation and Testing

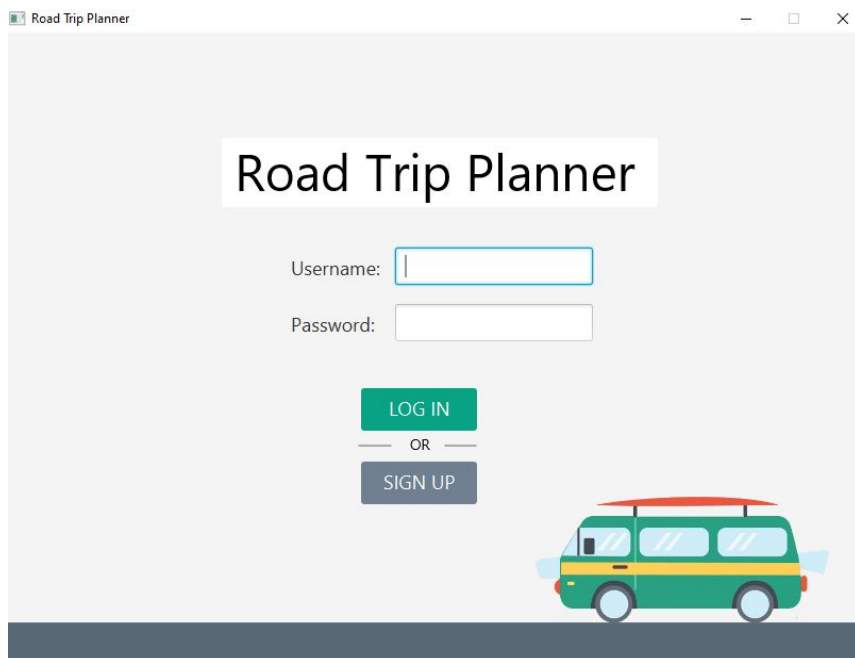
Implementation and Testing				
	Sean	Lamia	Linghui	Zainab
	Hours Spent			
Interface Design				
Home Page	1	1	1	1
Login Screen	1	1	1	1
Create User Page	2	2	2	2
Create Trip Page	2	2	2	2
View Plan Page	2	2	2	2
Plan Dashboard	3	2	3	2
Input Forms				
City Visits	2	2	2	2
Rental Car	2	2	2	2
Activity	2	2	2	2
Hotel	2	2	2	2
User Information	4	4	4	4
Database				
Querys	1	2	1	2
Design	1	2	1	3
Insert	1	1	1	1
Update	1	1	1	1
Delete	1	1	1	1
Select	1	1	1	1
Testing				
Interface Pages	3	3	3	3
Input Forms	3	3	3	3
Database querys	4	4	4	4
database design	5	5	5	5
Total Hours	44	45	44	46

## Code Listing

Our code is here: <https://github.com/zainabsiddiqui/road-trip-planner>.

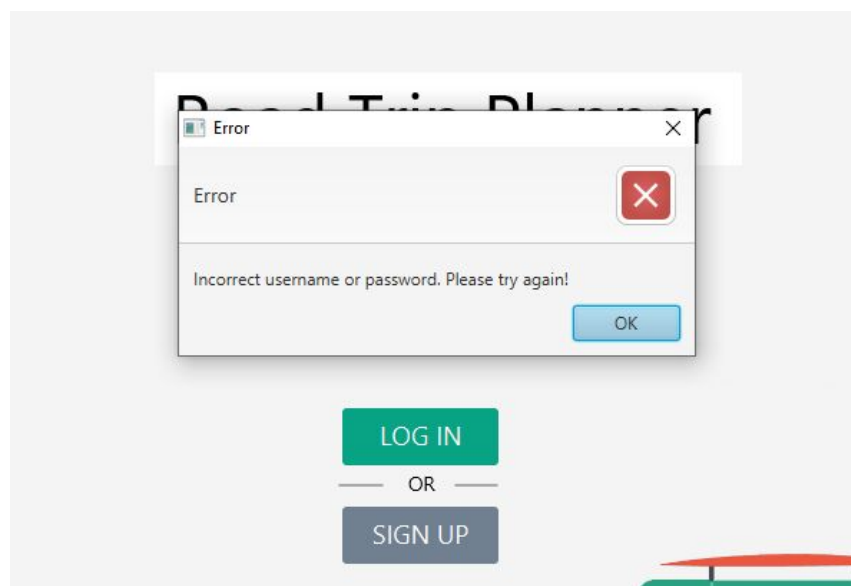
## Sample Output

On startup, the first screen the user is met with is the login screen, where they are prompted to enter their login credentials:

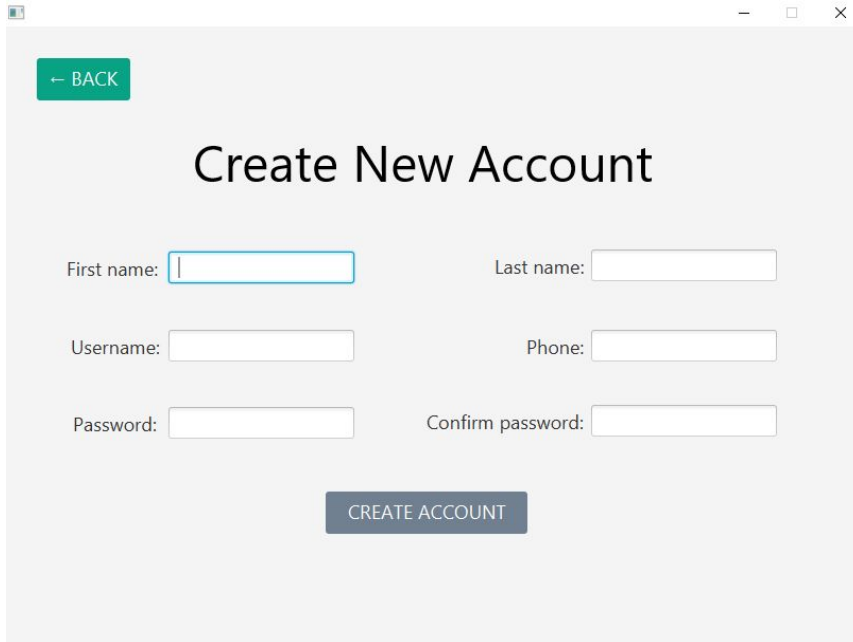


The image shows a web application window titled "Road Trip Planner". It features a login and sign up form. The form has two input fields: "Username:" and "Password:". Below these fields are two buttons: "LOG IN" (green) and "SIGN UP" (grey), separated by an "OR" label. To the right of the form is a cartoon illustration of a green and yellow van with a red roof rack. The background is a light grey gradient.

If the user enters a username and password combination not present within our USER table, our application sends the following error:

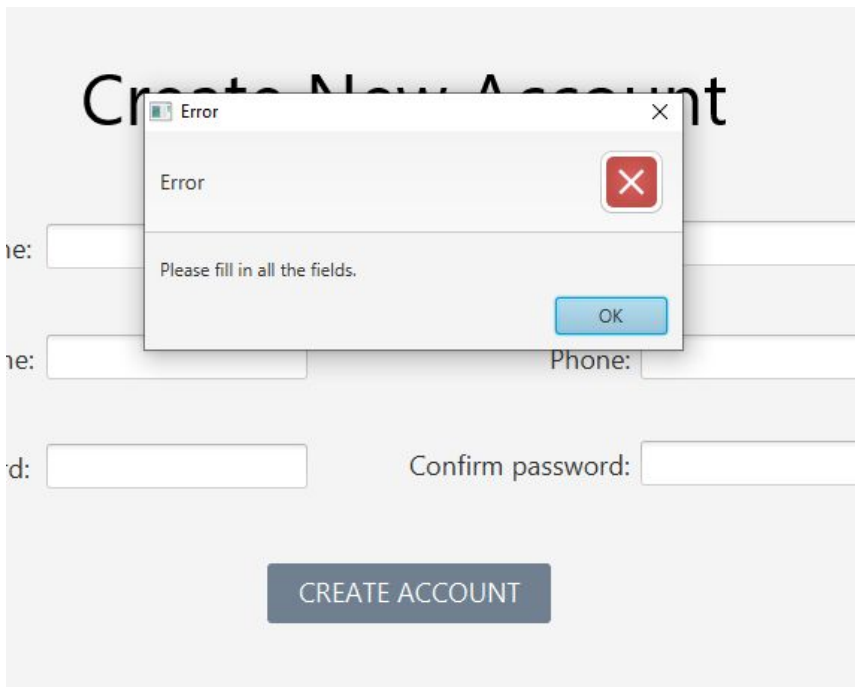


User can also choose to sign up or create a new account, the following is the associated form:



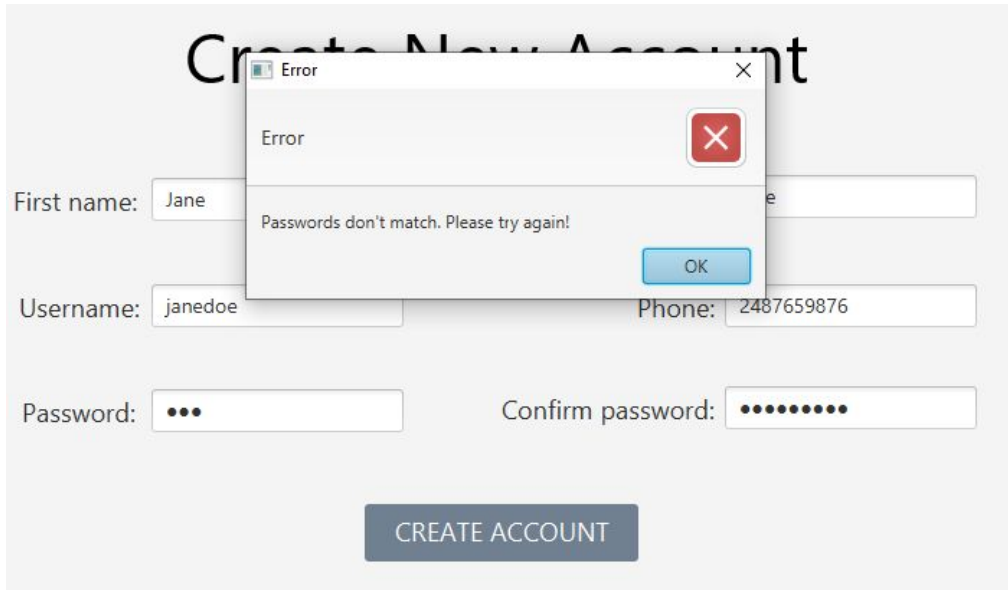
A screenshot of a web application window titled "Create New Account". In the top-left corner, there is a green button with a left-pointing arrow and the text "BACK". The main heading "Create New Account" is centered at the top. Below the heading, there are six input fields arranged in three rows: "First name:" and "Last name:" in the first row, "Username:" and "Phone:" in the second row, and "Password:" and "Confirm password:" in the third row. At the bottom center, there is a dark blue button with the text "CREATE ACCOUNT".

If the user does not fill in all the fields, an error is presented:



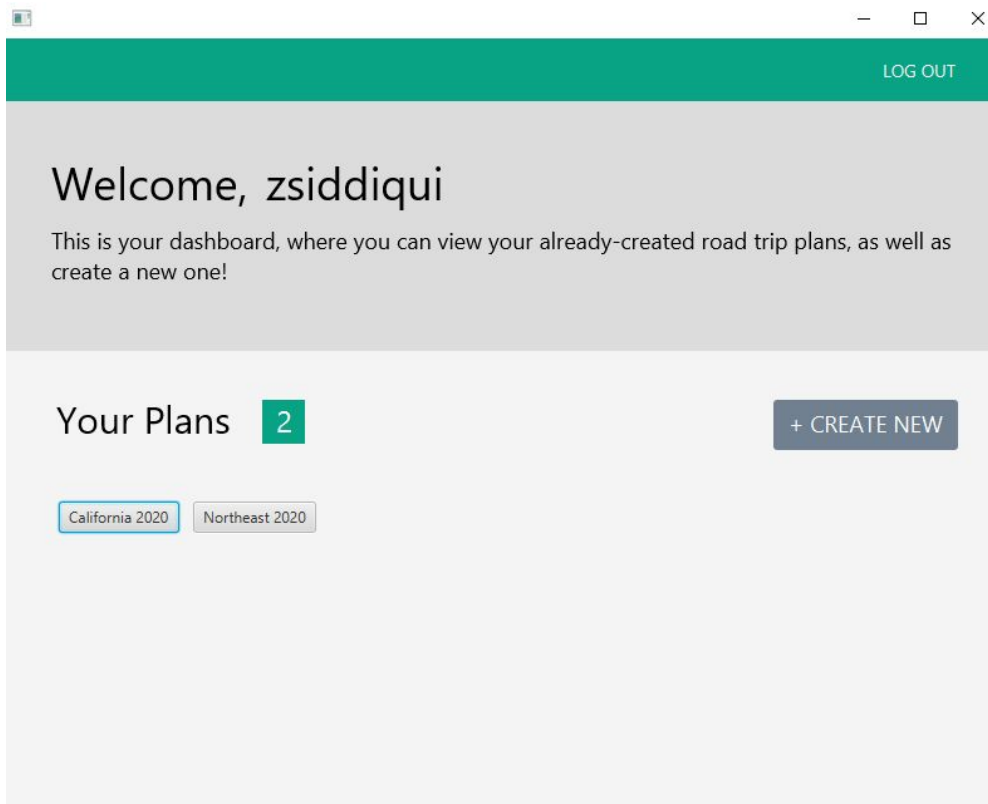
A screenshot of the same "Create New Account" form, but with an error dialog box overlaid in the center. The dialog box has a title bar that says "Error" and a red "X" icon in the top-right corner. The text inside the dialog box reads "Error" followed by "Please fill in all the fields." and an "OK" button at the bottom right. The background form is partially visible and slightly faded.

If the user enters two passwords that do not match, another error is presented:



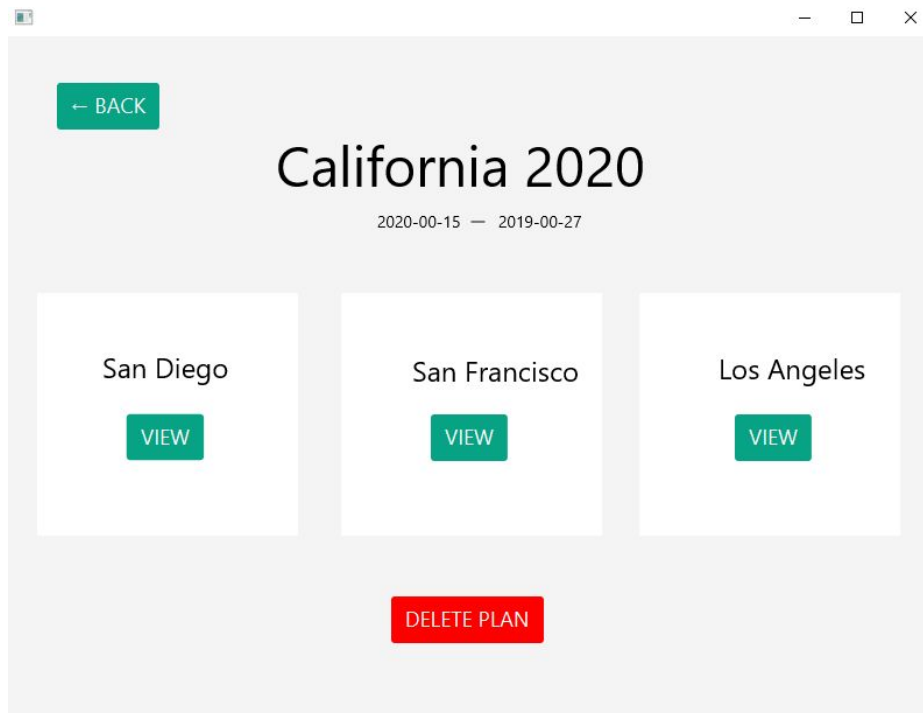
The screenshot shows a 'Create New Account' form. The form fields are: First name (Jane), Username (janedoe), Phone (2487659876), Password (masked with three dots), and Confirm password (masked with eight dots). A dark blue 'CREATE ACCOUNT' button is at the bottom. An error dialog box is open in the center, titled 'Error', with a red 'X' icon and the message 'Passwords don't match. Please try again!'. The dialog has an 'OK' button.

If the user successfully logs in, they are presented with the dashboard, which shows them their already created road trip plans:

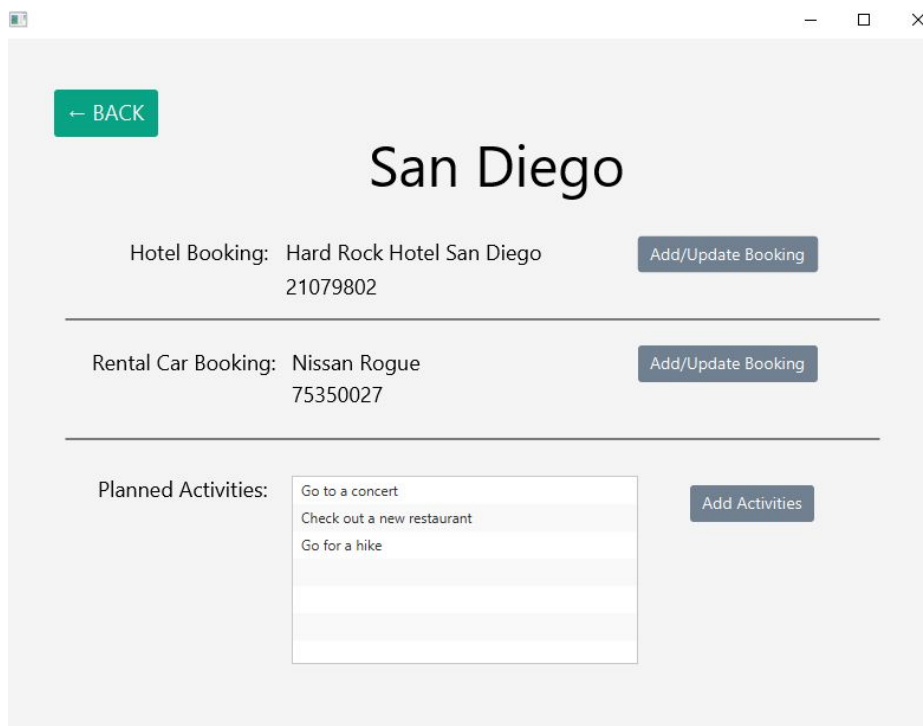


The screenshot shows a user dashboard for 'zsiddiqui'. At the top, there is a green header bar with a 'LOG OUT' link. Below the header, the user is greeted with 'Welcome, zsiddiqui' and a message: 'This is your dashboard, where you can view your already-created road trip plans, as well as create a new one!'. Under the heading 'Your Plans', there is a green badge with the number '2'. To the right is a '+ CREATE NEW' button. Below the plans, there are two buttons: 'California 2020' (highlighted with a blue border) and 'Northeast 2020'.

User can then click on any of the listed plans or create a new one. If we click on California 2020, this is what we see:

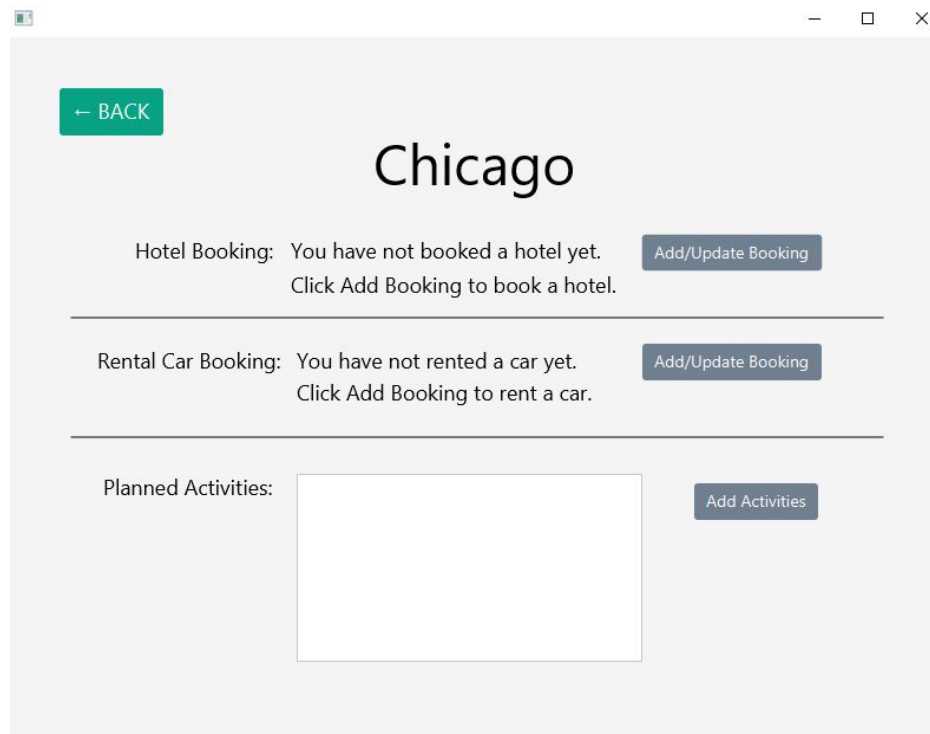


If we click on San Diego, which is one of the associated cities with this plan, we see the following:



As you can see, this city already has associated hotel bookings, car bookings, and planned activities. The confirmation numbers are listed below the bookings.

Let's view a city that does not have a booking yet:



← BACK

# Chicago

Hotel Booking: You have not booked a hotel yet.  
Click Add Booking to book a hotel.

Add/Update Booking

---

Rental Car Booking: You have not rented a car yet.  
Click Add Booking to rent a car.

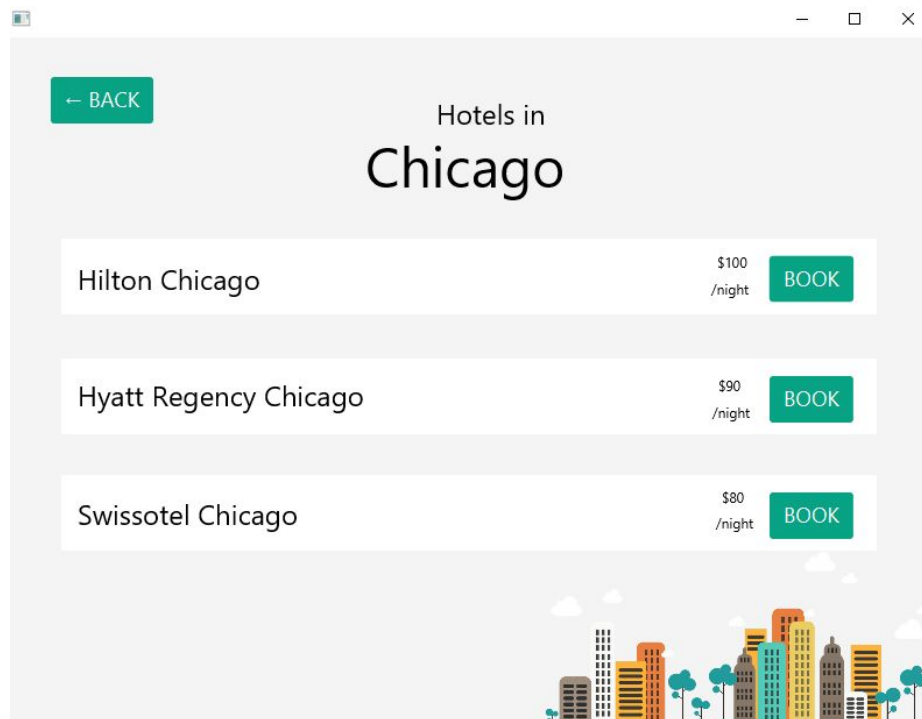
Add/Update Booking

---

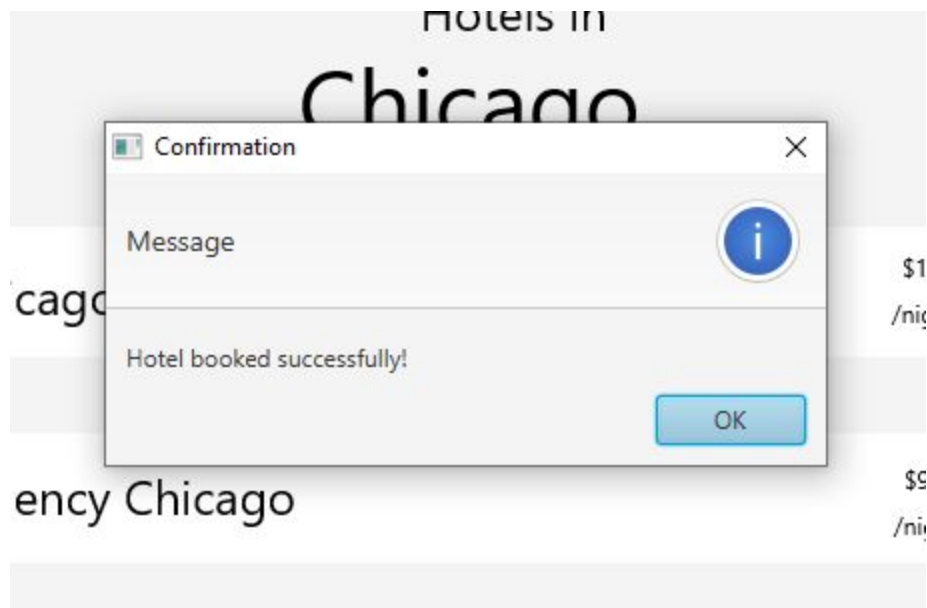
Planned Activities:

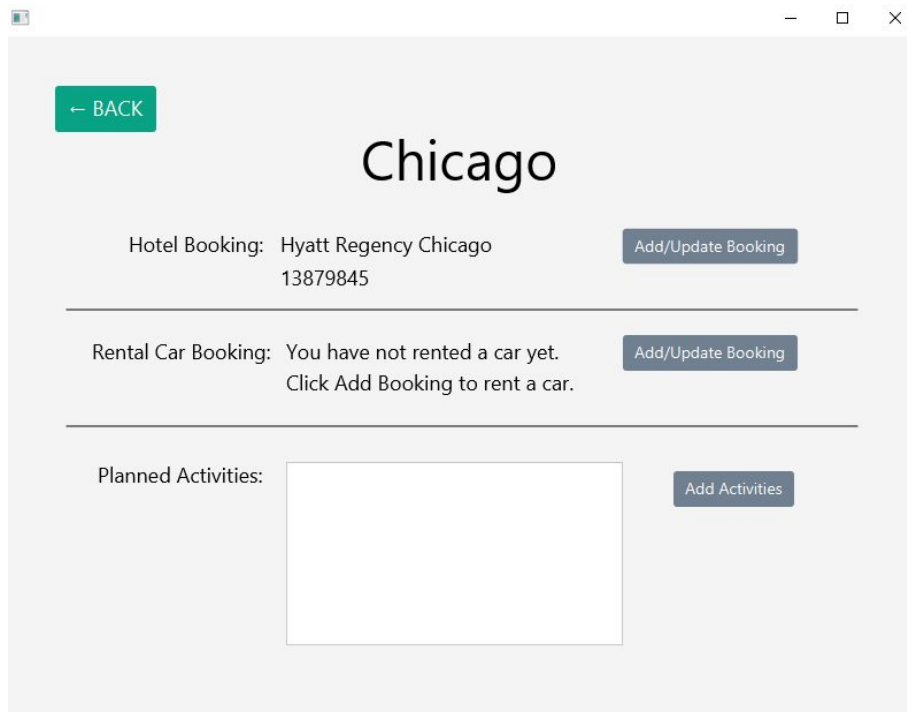
Add Activities

Now we can add bookings for Chicago. If we click on Add/Update Booking for a hotel, we get a page that lists the hotels available in Chicago:



If we book Hyatt Regency Chicago, we get a confirmation message and will now be able to see it on our city page:

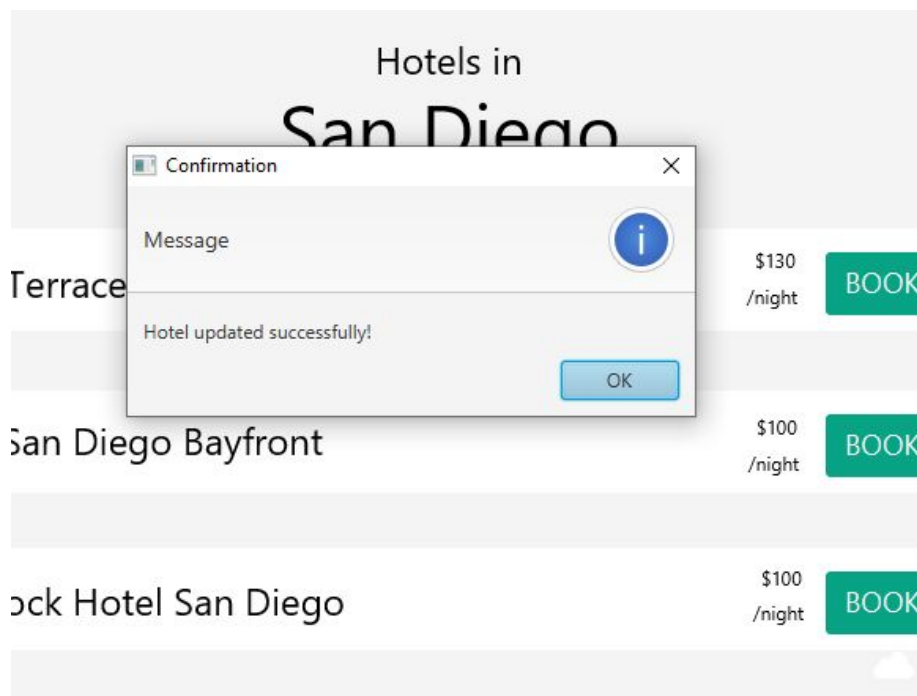




A screenshot of a web application interface for Chicago. At the top left is a green button with a left arrow and the text "BACK". The title "Chicago" is centered in a large, dark font. Below the title, there are three sections separated by horizontal lines. The first section is "Hotel Booking: Hyatt Regency Chicago 13879845" with a blue button "Add/Update Booking" to its right. The second section is "Rental Car Booking: You have not rented a car yet. Click Add Booking to rent a car." with a blue button "Add/Update Booking" to its right. The third section is "Planned Activities:" followed by a large empty rectangular box and a blue button "Add Activities" to its right.

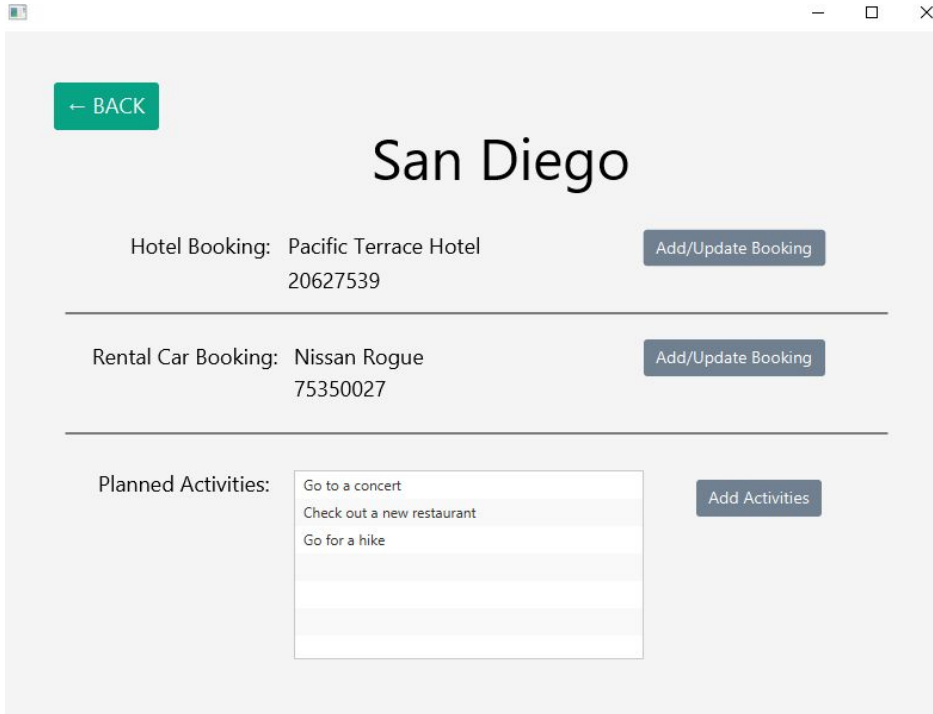
We can do the same thing with rental car bookings.

We can also update existing hotel and car rental bookings. If we go back to San Diego in our California 2020 plan, we can change our hotel to Pacific Terrace Hotel:



A screenshot of a web application interface for San Diego hotels. The title "Hotels in San Diego" is centered. Below it, there is a list of hotels. The first hotel is "Pacific Terrace Hotel" with a price of "\$130 /night" and a green button "BOOK". The second hotel is "San Diego Bayfront" with a price of "\$100 /night" and a green button "BOOK". The third hotel is "Rock Hotel San Diego" with a price of "\$100 /night" and a green button "BOOK". A "Confirmation" dialog box is overlaid on the screen, displaying the message "Hotel updated successfully!" and an "OK" button.





← BACK

## San Diego

Hotel Booking: Pacific Terrace Hotel  
20627539 [Add/Update Booking](#)

---

Rental Car Booking: Nissan Rogue  
75350027 [Add/Update Booking](#)

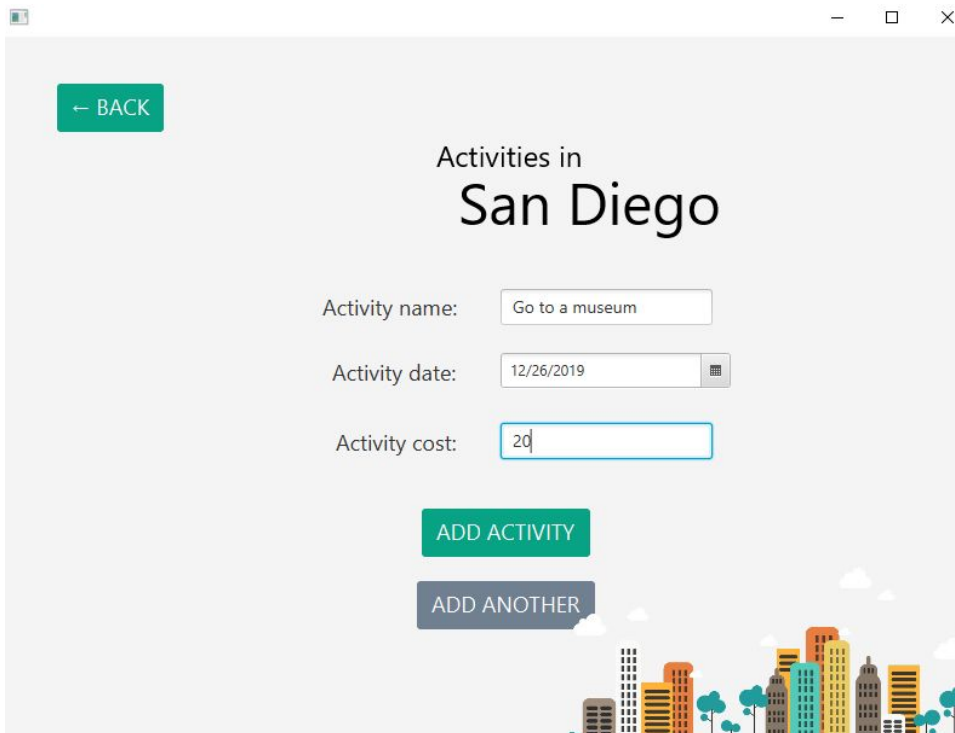
---

Planned Activities:

- Go to a concert
- Check out a new restaurant
- Go for a hike

[Add Activities](#)

We can also add activities for San Diego by clicking on Add Activities:



← BACK

## Activities in San Diego


Activity name:

Activity date:

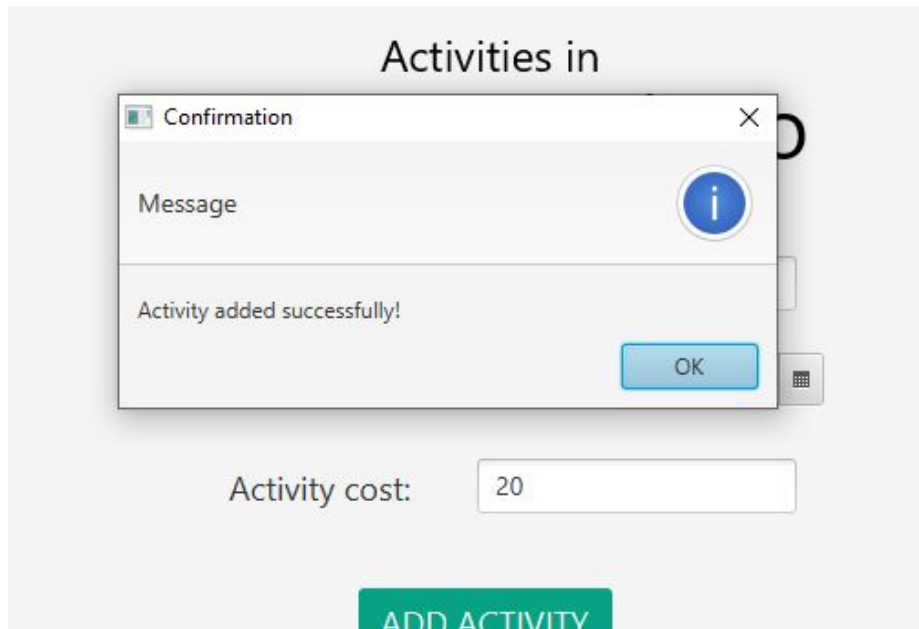
Activity cost:

[ADD ACTIVITY](#)

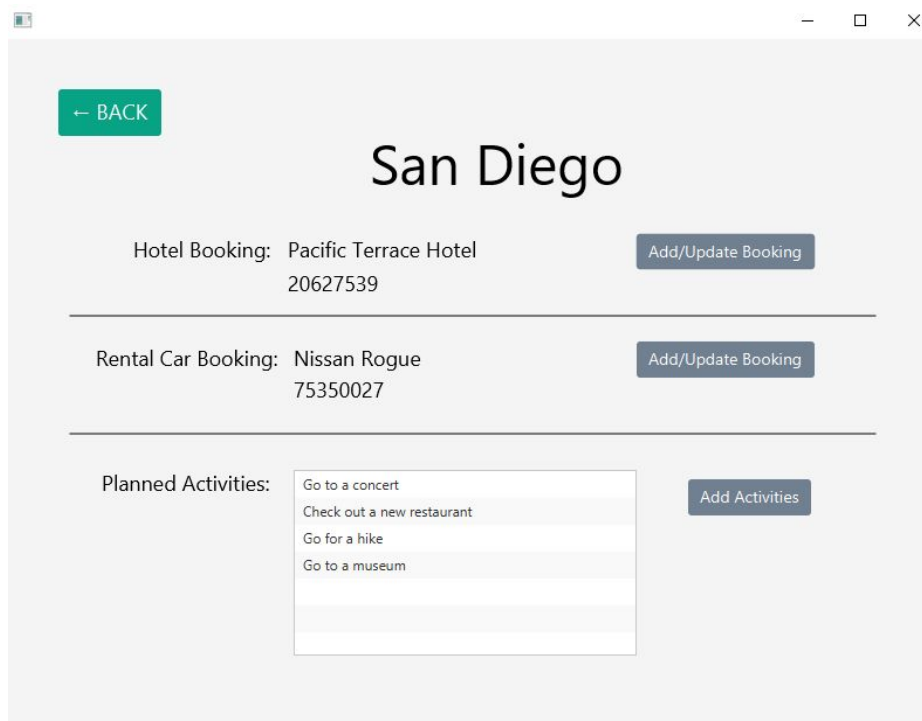
[ADD ANOTHER](#)



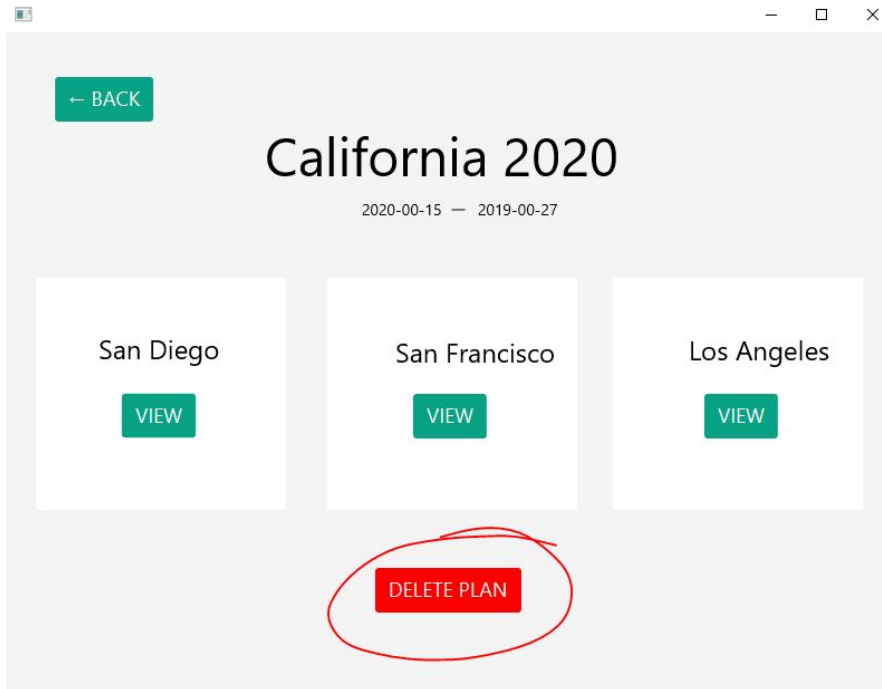
If we click Add Activity, we get a confirmation message:



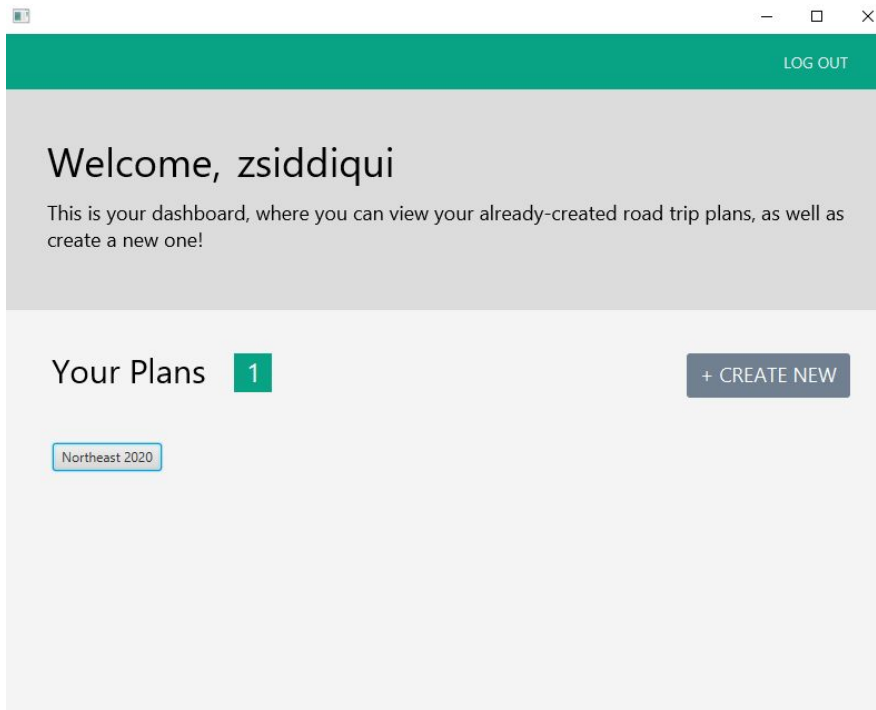
Now we see it reflected in the activities section of our page:



We can also delete entire road trip plans if we wanted to, using the delete button on the plan's page. Let's delete California 2020:



Once we click that button, we are returned to the Dashboard, where California 2020 is nowhere to be found:



Lastly, we can log out from the Dashboard. If we click the Log Out button, we are simply redirected to the Login page again.

**Estimate of Effort**

The estimated effort for this project will take us the full four weeks. Each member is expected to put in at least 6 hours of work towards the project in order to complete all of the requirements.

Estimate of Effort				
	Sean	Lamia	Linghui	Zainab
Tasks	Hours Spent			
Phase 1				
Problem Statement				
Create a description of the application	1	1	1	
System Requirements				
Define the scope of the system	1	1	1	1
How will the system be used	1	1	1	1
Conceptual Database Design				
Create a diagram of our database	2	2	2	2
Functional requirements				
Describe the database transactions required	1	1	1	1
List the inputs and outputs/database interactions	1	1	1	1
Reporting data and entry requirements	1	1	1	1
Estimate of effort				
Discuss the effort required	1	1	1	1
Create an implementation plan to complete	1	1	1	1
	10	10	10	10
Phase 2				
Logical database design				
Convert ER diagram into a set of relations	2	2	2	2
Create a table to show attributes	3	3	3	3
Clearly indicate all foreign/primary keys	0.5	0.5	0.5	0.5
Map out physical design of our database	2	2	2	2
Design of application programs				
List SQL queriese that are needed	3	3	3	3
Implementation of modules	1	1	1	1
Configure modules and complete testing	2	2	2	2
Document each model	1	1	1	1
Interface Design				
Planning user interface design	1.5	1.5	1.5	1.5
Begin sketches of the various screens	2	2	2	2
Create scenarios of how the application is used	1	1	1	1
Indicate DB entities and attributes being modified	2	2	2	2
Implementation and testing				
Create a plan for implementation	1	1	1	1
Create test for our database	1	1	1	1
Create a schedule	1	1	1	1
	24	24	24	24
Phase 3				
User manual				

Describe what users need to know about the system	1	1	1	1
Provide documentation on functionality	1	1	1	1
Implementation manual	3	3	3	3
Create a list of dependencies	1	1	1	1
Create GUI				
Home page	2	2	2	2
User account creation	2	2	2	2
Create new plan	2	2	2	2
See past plans	2	2	2	2
Plans page	2	2	2	2
Hotel page	2	2	2	2
Rental car booking	2	2	2	2
Activity page	2	2	2	2
DB implementation				
provide steps on how we did this	1	1	1	1
	23	23	23	23
Total estimate of project hours	57	57	57	57

# Road Trip Planner



Authors: Zainab Siddiqui, Sean Boucher, Linghui Pan, Lamianoor Zinia

# Problem Statement

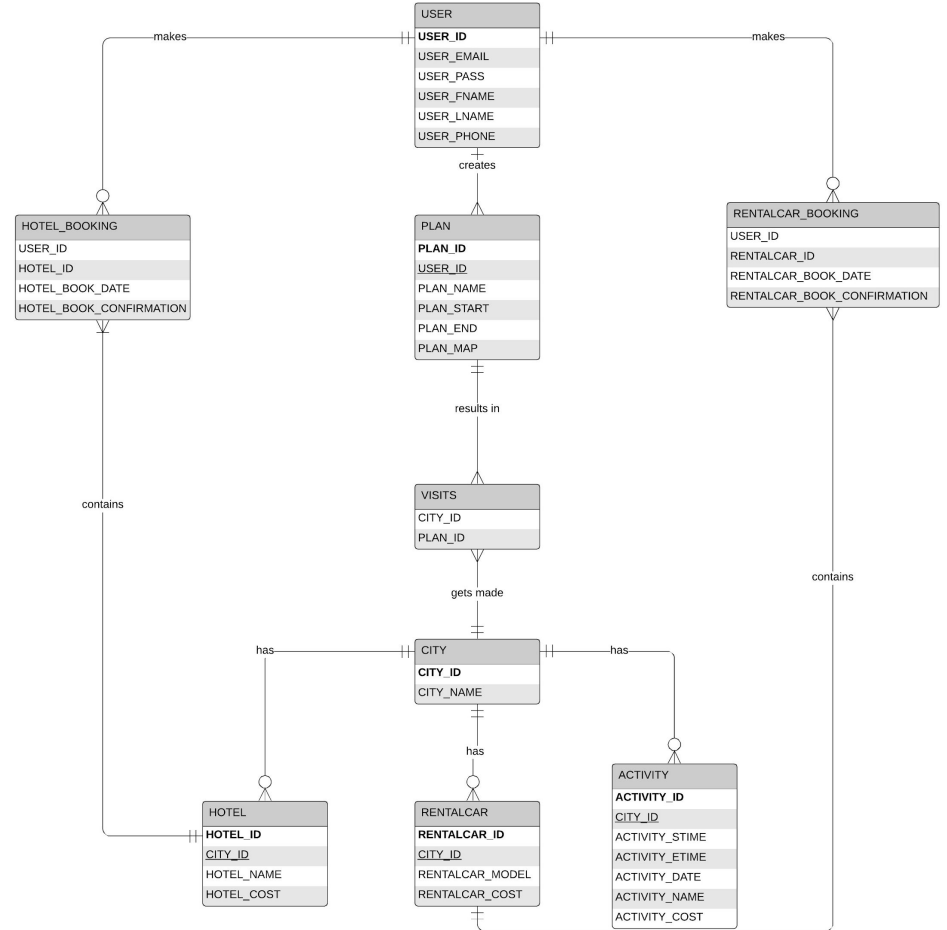
Our application is generated towards travelers who like to travel with a plan or keep many travel itineraries. Users will be allowed to create and organize their itineraries with all of pertinent information- hotel and car rental bookings, activities etc. Integral to our road trip management system will be our MySQL database -it will allow us to store, retrieve, update and remove road trip plans as well as set up user authentication. We believe an application like this should remove a lot of headache that comes with having to plan a vacation in your head and will instead make the whole process a lot more organized and exciting. This application is effective for travelers to save time and cost of travelling.



# Functional Requirements

- Allow user to sign up for an account (input needed: email (string), password (alphanumeric) - user validation operations will be conducted)
- Allow user to log in and log out (user authentication operations conducted)
- Allow user to change password (input needed: new alphanumeric, update operation conducted)
- Allow user to create a new road trip plan (inputs needed: name (string), start date, end date - create operation)
- Allow user to see all created plans on front page (read operation)
- Allow user to delete plan (delete operation)
- Allow user to modify plan - like name or range of dates (update operation)
- Allow user to upload, update, or remove a map image of their proposed road trip (input needed: image (not mandatory), create, update, and delete operations)
- Allow user to record cities they will be visiting (inputs needed: name (string), start date, end date (must ensure that this range of dates falls under the general plan's range of dates) - create operation conducted)

# ER Diagram



# Tools we used

- PopSQL
  - Used to be able to collaborate on the database
- IntelliJ
  - Software used to develop the project
  - Project written in JavaFX
- Github
  - Used to store our project so we could collaborate
- GroupMe
  - Used to stay in contact with each other
- Google drive
  - Used to store our live documents and ideas
- MVC model used

Demo