Team Members and Emails

*red text are changes

Austin Bahng – abahng@usc.edu
Daniel Ho – hsiaotuh@usc.edu
Jamin Chen – jaminche@usc.edu
Zexia Zhang - zexiazha@usc.edu

High-Level Requirements

The desktop application that would help users plan out roadtrips. A new user can create a password protected profile to save their own maps, while unregistered users can create routes but won't be able to save or share the map. Based on the start and end destination that the users enter, the application will help users create their own itinerary, which includes activities, restaurants, and lodging along the route of the trip. The first thing a user can do is input a starting position and a destination for the map to generate a route. Then the user can either manually search for points of interests or select suggested stops on the application. If a new stop is added, then the map regenerates the optimal route that includes the stop. For each stop, the user can specify how much time they would like to spend at that stop, and the application will recalculate the total time of the trip. Itineraries can be edited by multiple users. Itineraries that are created with our application will be easily shareable with other users, both privately and publicly. Public itineraries can be viewed by users across the application.

Technical Specifications

Requirements

- Web Server
- Database
- Desktop client

Client-side (43 Total Hours)

Login Page (3 hours)

- Login page with username and password for user sign-up or validation

Homepage create new map (6 hours)

- Homepage can create new maps
- Add shared map given id

Load Previous Itinerary (4 hours)

- Load previous itineraries from database and Google Maps API

Reloading itinerary based on real-time edits (6 hours)

- Update the itinerary and map displaying based on any user edits

Method to auto-save (2 hours)

- Make API call to update itinerary

Itinerary Page Sidebar (8 hours)

- Shows Itinerary details — name, id, stops, shared users
- Functionality to invite other users to edit our itinerary by username
- Method to enter amount time user wants to spend at each stop

Search Bar (8 hours)

- You can search for new locations and add them to your itinerary
- Search results will show on the map

Viewing public itineraries page (6 hours)

- Page to view public itineraries

Server-side (35 Total Hours)

Database connection (2 hours)

- Connect front-end application with back-end server

Web hosting (2 hours)

- Hosting database

Login/Signup (4 hours)

- Existing user authentication
- User creation

Create new itinerary (2 hours)

Load homepage data (2 hours)

- Load all itineraries associated with given user
- Load public itineraries to display

Itinerary API (8 hours)

- Load itinerary
- Update itinerary
- Delete itinerary

Alert Client of Changes (3 hours)

- Function that lets client know if itinerary has been changed since last load

Load Public Itineraries (4 hours)

- Load the itineraries from the database into the application page

Multi-User Editing (8 hours)

- Function to create threads for each user that is editing

Database (2 Total Hours)

Itinerary Object (1 hour)

- This object stores the data for the itinerary of a single trip.
- Consists of username of the owner, itinerary name, itineraryID, list of users who can edit, list of stops (includes start and end), timestamp (of last update), and the total time the trip will take

User Object (1 hour)

- This object stores data of a single registered user
- Consists of username, password, email, list of itineraries they're allowed to edit

---

Detailed Design

Hardware and Software requirements

Internet Access

Windows

- Windows 10 (8u51 and above)
- Windows 8.x (Desktop)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 and 2012 R2 (64-bit)
- RAM: 128 MB
- Disk space: 124 MB for JRE; 2 MB for Java Update
- Processor: Minimum Pentium 2 266 MHz processor
- Browsers: Internet Explorer 9 and above, Firefox

Mac OS X

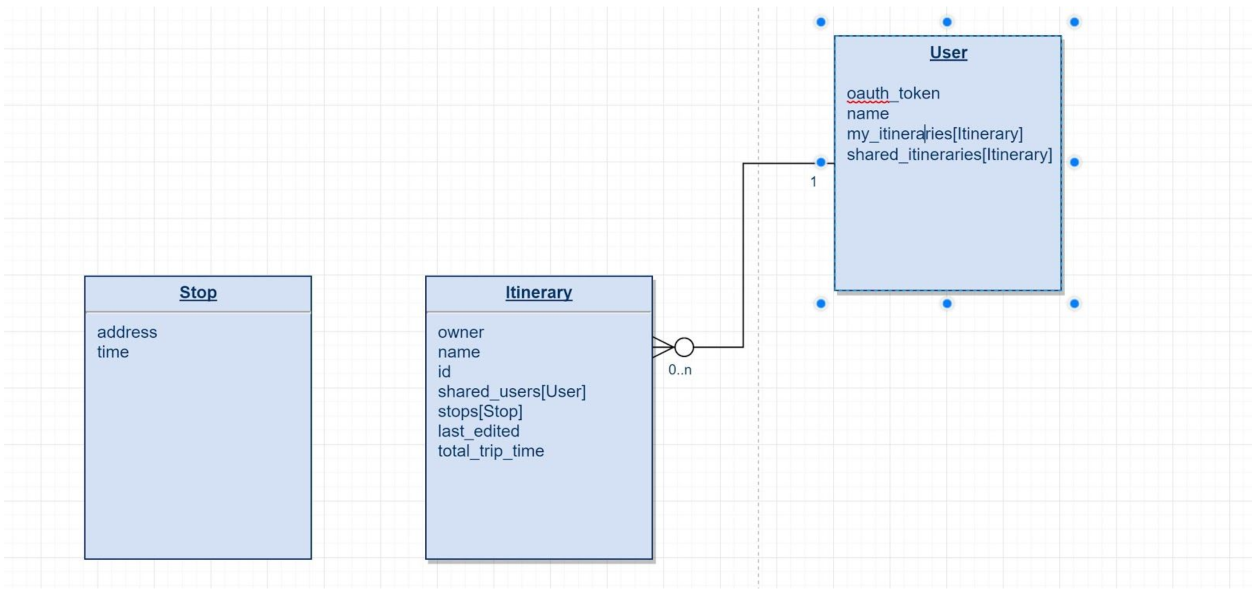
- Intel-based Mac running Mac OS X 10.8.3+, 10.9+
- Administrator privileges for installation
- 64-bit browser

Linux

- Oracle Linux 5.5+[1]
- Oracle Linux 6.x (32-bit), 6.x (64-bit)[2]
- Oracle Linux 7.x (64-bit)[2] (8u20 and above)
- Red Hat Enterprise Linux 5.5+[1], 6.x (32-bit), 6.x (64-bit)[2]
- Red Hat Enterprise Linux 7.x (64-bit)[2] (8u20 and above)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64-bit)[2] (8u31 and above)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 and above)
- Ubuntu Linux 15.04 (8u45 and above)
- Ubuntu Linux 15.10 (8u65 and above)
- Browsers: Firefox

Database Schema

- ERD (for MongoDB):



- Note: Stop is not a standalone JSON object (that's why it's not linked). Stops are nested in the Itinerary.

GUI Mockup

Image A - Login Page

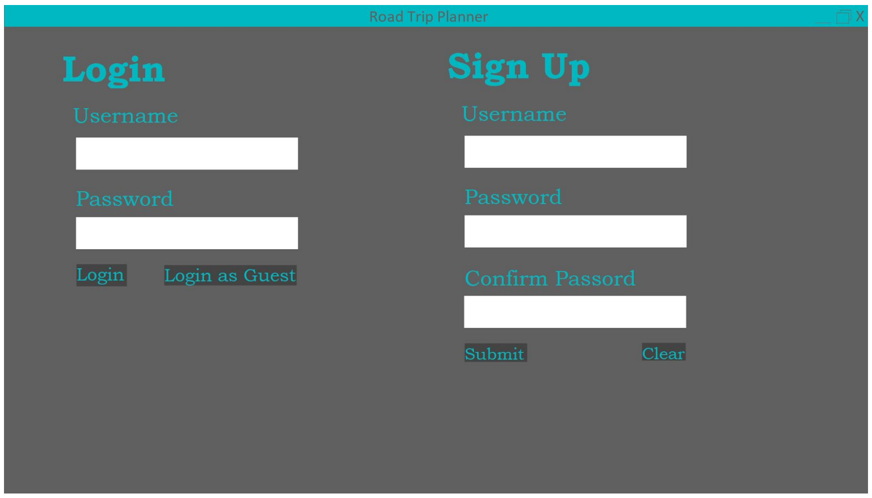
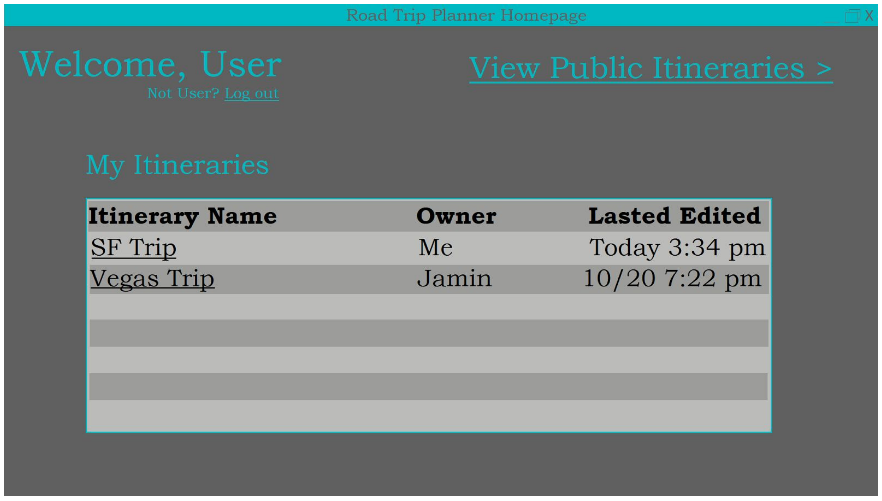
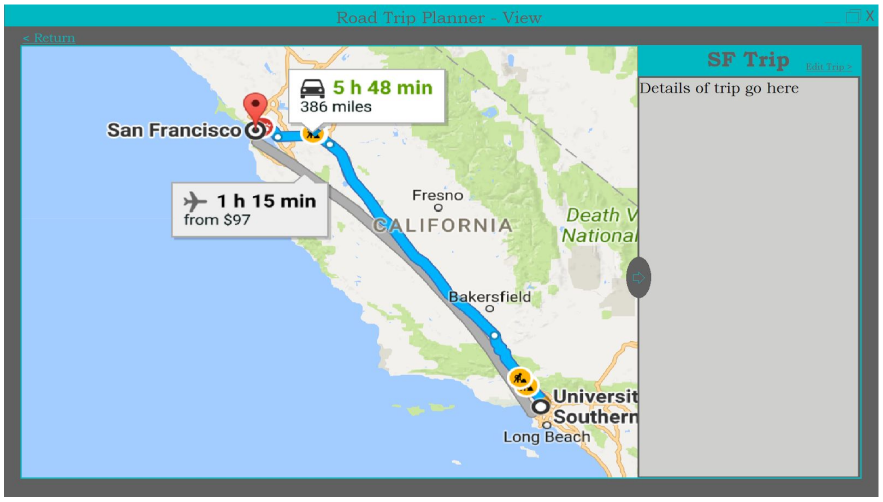


Image B - Homepage



Itinerary Name	Owner	Lasted Edited
SF Trip	Me	Today 3:34 pm
Vegas Trip	Jamin	10/20 7:22 pm

Image c - Map View/Edit Page



Details of trip go here

Classes

Frontend

- Login Page (Image A) - This page two options to either logging in with google (new window will pop up to choose google account) or log in as a guest
- Homepage (Image B) - This page is available to authenticated and guest users, and has two tabs, my itineraries and public itineraries. Authenticated users will have access to the my itineraries tab while guests will not. It will be sortable based on rating and name.
- Roadtrip view screen (Image C)- Available to both guest and authenticated users; this screen gets pulled up when users want to view the details for a roadtrip that they do not have edit access for. Details can be viewed in a sidebar that is collapsible.
- Render.js
  - Has the google maps API display the actual map
  - Using the Itinerary information, loop through each stop and add the stop to get a google maps route.
- Update.js
  - Updates the client-side Itinerary object whenever an action is performed (e.g. adding a new stop, changing the time, sharing with another user)
- save.js
  - Upon pressing the save button, returns the current Itinerary information to be updated

Frontend's Backend (Node.js Server)

- Templating engine: EJS
- Desktop framework: Electron
- User Class
  - Object {

```
                oauth_token: string,
                name: string,
                picture: string,
                my_itineraries: [itinerary.id]
                shared_itineraries: [itinerary.id]
        }
```

- Itinerary Class
  - Object {

```
                owner_name: owner_name,
                name: string,
                _id: {
                        oid: string,
                    }
                shared_users: [user.name],
                stops: [{
                    address: string
                    time: int
                    }],
                lastModified: java.util.Date()
                //total_trip_time: int
                public: boolean,
                thumbnail_url: a string that stores an image to that itinerary
            }
```

- Electron module
    - Handles all module calls, page routing, and display functionality.
    - When a POST is received on /update then the itinerary is passed to the RoadTrip API module to be updated in the database.
- RoadTrip API module
    - Communicates with the custom Java backend API.
    - When receiving an Itinerary, it calls the correct backend api (single/public) edit roadtrip in order to update the database item.
    - When given an itinerary_id, it verifies that the user session has access to it and then retrieves it using the custom roadtrip api.
- Authentication module
    - Upon entering a username/password, it will prompt Google OAuth for an authentication token. If success, compare with own database (by making a call to custom login api). If found, redirect to homepage and render with correct Itinerary object.
    - Upon signing up using Google OAuth, verify that oauth_token does not already exist in the database. If success, create a new User object and pass into custom signup API. redirect to home and render.

Backend

- Note: GSON will be used for all of the servlets because Mongo stores data as JSON.
- Login Servlet
    - This servlet will process user login authentication and account creation. There will also be the option for users to continue as guests.
    - For new users who are creating an account, the servlet will first determine if the username is already taken. If so, it will prompt the user to choose a different username. If not, it will put the username and password inside the login database and then approve login.

        - For previously existing users, the servlet will check if the submitted password hash (hashing done on front-end) matches with the stored password hash in the database. If yes, it will approve the login. If no, it will let the user know that the username/password combination is invalid.
        - For guest users, there will be a function in the servlet that gets called if the user continues as guest. Login will be "approved," but some sort of flag will be set such that functionality is limited.
    - Public RoadTrips Servlet - This servlet is used for the page where users can view other users' roadtrips. Clicking on any of them will then redirect to the single road trips servlet.
        - The servlet will connect to the database and fetch a list of public road trips and send the result as JSON to our Node server.
    - Private RoadTrips Servlet - This servlet is used for the "my road trips" page, for authenticated users to view the roadtrips that they have created or road trips they are part of.
        - For authenticated users, the servlet will connect to the database and return a JSON object containing which roadtrips the current user owns, and which roadtrips the current user has edit access to.
        - For authenticated users, there will be a function that allows users to create a new road trip, which will get routed to the single road trips servlet.
        - Attempting to call any of the functions above as guests (they should all be disabled on front-end anyways) will result in a redirect to the public road trips page.
    - Single RoadTrips Servlet - This servlet is used for handling interacting with one specific roadtrip. For example, it would be called when public users want to view a specific road trip, or when private users want to edit a road trip.
        - This servlet will have a method to return all the JSON information for a specific roadtrip.
        - If this road trip is owned by the current user, this method will also return a list of users who have edit access.
        - There will be a function to create a new road trip for authenticated users.
        - There will be a function to update the data for an already-existing road trip, for authenticated users. This data includes who is in the party and has edit access.
    - Database Calls REQUESTS
      Endpoint: http://roadtrip-env.us-west-1.elasticbeanstalk.com/<REQUEST>
        - GET /Authenticate Authenticate/<id_token> - returns: {user}
            - Login
            - Returns the user, if exists. Otherwise will return {success:false}

- POST /Authenticate - data: id_token - return: {user}
    - Signup
    - Creates a new user with the id_token if possible. Returns the user
- GET /Itinerary - Itinerary/<_id> - returns: {Itinerary}
    - Fetch user’s Itinerary based on id
    - Retrieve the Itinerary based on _id
- POST /Itinerary - data: {Itinerary object} - return: {Itinerary object}
    - Create new Itinerary or save Itinerary
- DELETE /Itinerary - Itinerary/<_id> - returns: {success:bool}
    - Delete an Itinerary based on the oid id
- GET /PublicItinerary - return: {itinerary[]}
    - Retrieve a list of all public itineraries. Returns the list
    - Upload a new or existing Itinerary to the database. Returns success.
- GET /MyItinerary/user’s name - return: {Itinerary[]}
    - Fetch user’s Itineraries
    - Retrieve all the user’s personal itineraries

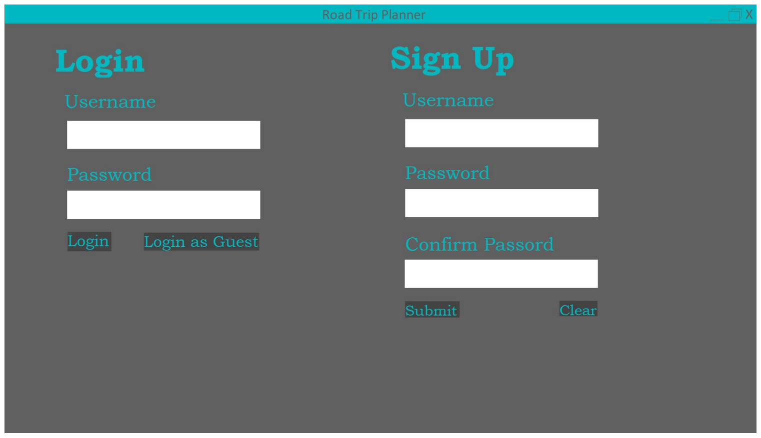
Test Cases

Database Test Cases:

1. Create new user and search this user in database, make sure the returned json is valid and correct (this include token, name, shared-itinerary, etc).
2. Search a user that does not exist in database, ensure a warning and null is returned.
3. Create an itinerary with a given user, make sure this user's access to itinerary is updated in the database
4. Search in the database for the itinerary just added, make sure the returned json is valid and correct (this include owner, name, id, users, stops, etc)
5. Modify a user information, make sure it's updated in database
6. Modify an itinerary information, make sure it's updated in database
7. Add users to an itinerary that already exists, make sure the database update the access change in user and itinerary
8. Multiple users edit the same itinerary, make sure the itinerary is updated dynamically, and new changes may overwrite past ones
9. Delete a user from access to one itinerary, then try to access it again in database, it should return a warning
10. Delete a user, try to access its information, should return a warning
11. Delete a user, make sure the itineraries owned by him is also removed from database
12. Delete an itinerary and try to access it, it should return a warning
13. When in multiple user editing mode, make sure the database sync the last edit time and total trip time
14. Pressure test. Constantly create new users, and make sure database still behaves normally. When it reaches the limit, make sure the most inactive user is removed from database.
15. Pressure test. Constantly create new itineraries, and make sure database still behaves normally. When it reaches the limit, make sure the itinerary with furthest last edit time is removed from database.

Login Page Test Cases (Image A):

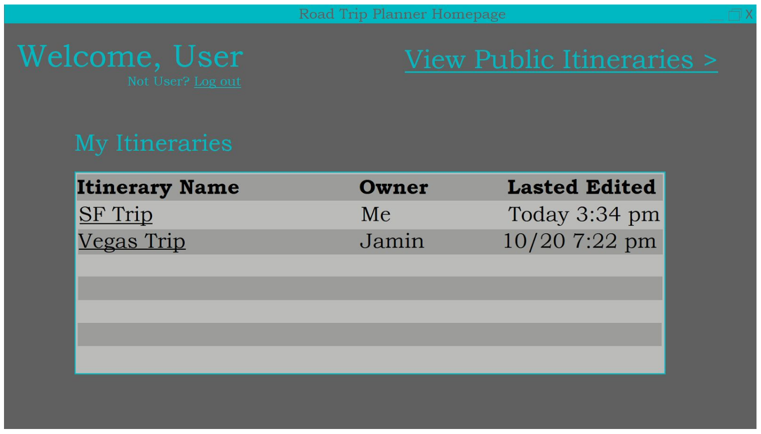
1. Test the login functionality with an existing username and password. The application should direct the user to the homepage.
2. Test the login functionality with an existing username and incorrect password. The application should stay at the login page and display an error message saying that the "username/password combination is invalid."



3. Test the login functionality with a non-existent username. The application should stay at the login page and display an error message saying that the “username/password combination is invalid.”
4. Test the login functionality with a blank username. The login button should be greyed out on the front-end, and on the back-end the login request should be rejected if the user disables the front-end javascript.
5. Test the login functionality with an empty password. The login button should be greyed out on the front-end, and on the back-end the login request should be rejected if the user disables the front-end javascript.
6. Test the sign up functionality by entering a non-existent username and valid password. The application should create a new account for the user and then direct them to the homepage.
7. Test the sign up functionality by entering an existing username. The application should display a message saying that “This username is already taken.”
8. Test the sign up functionality by entering a blank username. The sign-up button should greyed out on the front-end, and on the back-end the sign-up request should be rejected if the user disables the front-end javascript.
9. Test the sign up functionality by entering a non-existing username and a blank password. The sign-up button should greyed out on the front-end, and on the back-end the sign-up request should be rejected if the user disables the front-end javascript.
10. Test the guest functionality by clicking on the “continue as guest option.” The user should be redirected to the homepage, but certain privileges should be disabled (they should not be able to edit itineraries, etc).
11. Test the login functionality by entering a username with non-alphabetic and non-numerical characters. The username should be rejected on both the front-end and back-end (to prevent sql injection attacks)
12. Test the login functionality by going to the login page when the user already has a session active. It should redirect the user the homepage and retain the current state.
13. Test the login functionality by going to the login page when the user does not have an active session. It should not redirect the user anywhere.

Homepage/My itineraries Test Cases (Image B):

1. Clicking the logout button should direct the user back to the login page
2. Clicking on the view public itineraries button should direct the user to the public itineraries page
3. When a logged in user clicks the view public itineraries page, the public itineraries page should load with the view my itineraries button visible

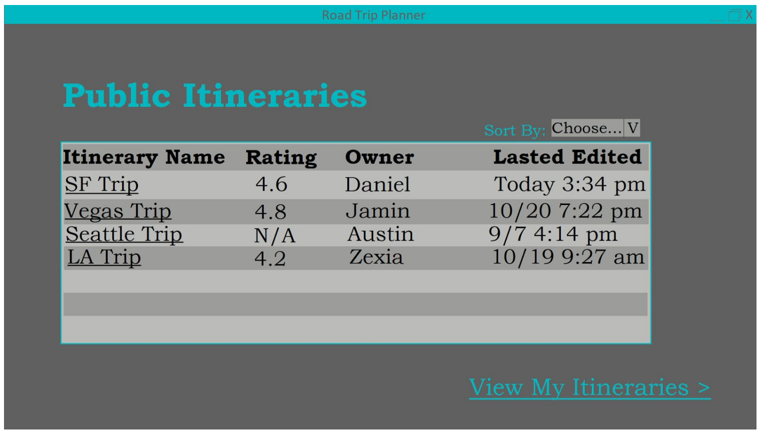


Itinerary Name	Owner	Lasted Edited
SF Trip	Me	Today 3:34 pm
Vegas Trip	Jamin	10/20 7:22 pm

4. Clicking an itinerary from the list should direct the user to the view itinerary page with the selected itinerary loaded
5. When an itinerary that is owned by the user is loaded, the edit option should be visible on the next page
6. When an itinerary that is not owned by the user and user does not have permission to edit, the edit option should not be visible on the next page
7. When an itinerary that is not owned by the user and user does have permission to edit, the edit option should be visible on the next page
8. When an itinerary is edited, the last edited time should be updated accordingly
9. When the create a new itinerary button is selected, a new itinerary should appear on the list that is titled “New Itinerary” and the owner as “Me”

Homepage/Public Itineraries Test Cases (Image B):

1. Selecting an itinerary from the list should direct the user to the itinerary view page with the selected itinerary loaded and the edit option disabled
2. When selecting sort by rating - high to low, the list should be sorted by rating with the highest rated itineraries showing first followed by lower rated itineraries
3. When selecting sort by rating - low to high, the list should be sorted by rating with the lowest rated itineraries showing first followed by higher rated itineraries
4. When selecting sort by name - A to Z, the list should be sorted by name in alphabetical order
5. When selecting sort by name - Z to A, the list should be sorted by name in reverse alphabetical order
6. If the user is a registered and logged in user, the view my itineraries page button should be visible
7. If the user is just a guest, the view my itineraries page button should be hidden



Road Trip Planner

Public Itineraries

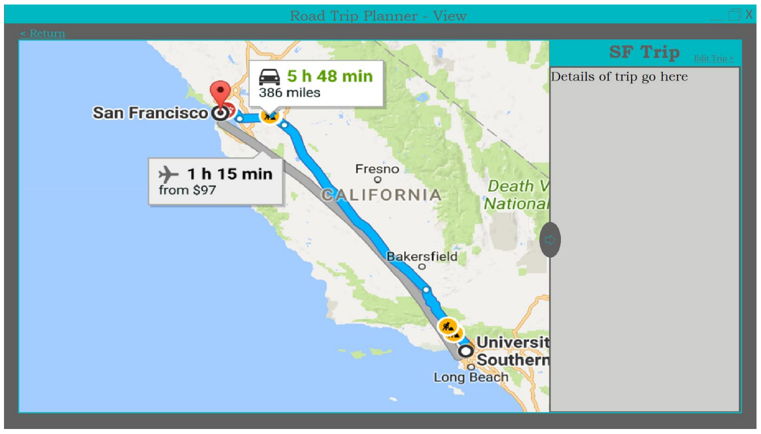
Sort By: Choose... V

Itinerary Name	Rating	Owner	Lasted Edited
SF Trip	4.6	Daniel	Today 3:34 pm
Vegas Trip	4.8	Jamin	10/20 7:22 pm
Seattle Trip	N/A	Austin	9/7 4:14 pm
LA Trip	4.2	Zexia	10/19 9:27 am

View My Itineraries >

Itinerary Viewing Page Test Cases (Image C):

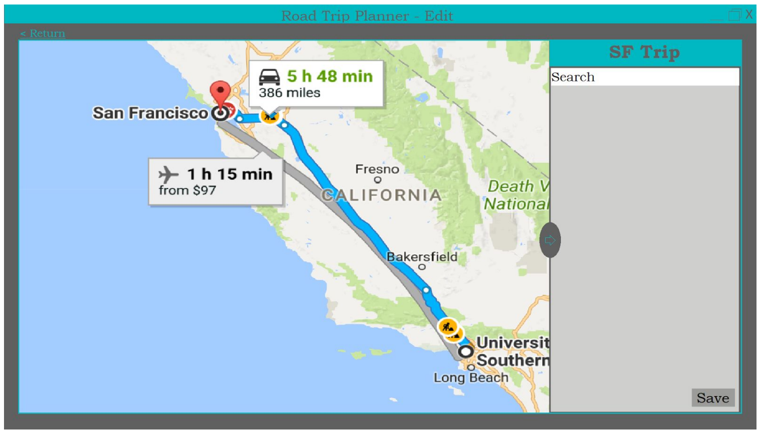
1. When the return button is clicked, user should be directed to either the my itineraries page if they came from my itineraries or public itineraries if they came from public itineraries
2. When user selects the arrow on the sidebar, the sidebar should collapse/expand



Details of trip go here

3. The edit button should only be visible if the user owns the itinerary or have permission to edit itinerary
4. Clicking the edit button should direct the user to the itinerary edit page
5. Clicking and dragging the map interface should move the map around
6. Selecting stops on the sidebar should highlight the location on the map with a pin
7. The correct total trip time should appear on the sidebar

Itinerary Editing Page Test Cases (Image C):



1. When creating a new itinerary, the webpage should be redirected to the maps page with an empty map.
2. When loading a private itinerary, the webpage should be redirected to the maps page with the itinerary loaded on the map
3. When loading a public/shared itinerary, the webpage should be redirected to the maps page with the itinerary loaded on the map. Any people currently on the same page should also show up “active”
4. When adding a new stop by address/name, the page should ask if it’s the correct one and allow the user to choose which to add.
5. When adding a stop that does not exist (bad address), the page should notify the user that there are no suitable stops.
6. When adding a stop that is unreachable by car, the page should notify the user that the stop is inaccessible.
7. When confirming to add a stop, the map should re-render to show the newly added stop and the route going through it.
8. When deleting a stop, the map should re-render to show the route not necessarily having to go through the stop.
9. When placing a pin on the map, the address information should show up
10. When placing a pin on the map, if the address is legal, the user should be prompted to add stop or not
11. When selecting a stop, the user should be able to insert an amount of time spent there (default is 0). The time should be added to the total time
12. When selecting a stop, if the user inputs an incorrect amount of time (e.g. negative), then a warning is given for incorrect notation
13. When opening the stops tab, the user should be able to drag around stops to change their order.
14. Each time a stop is modified (order changed, added, deleted), the map should re-render to display the new route
15. If the user does not input at least two stops, a beginning and an end, then a warning should be given asking for at least two or more inputs

Deployment

Server

1. Install MongoDB
2. Install Java
3. Install Node.js
4. Install Electron.js through Node package manager
5. Start the database server
6. Start the Java back-end server

Client

1. Import project files
2. Launch the executable file