**Voyager-Trip Planner**

**Authors:**

**Vrushank Gude**

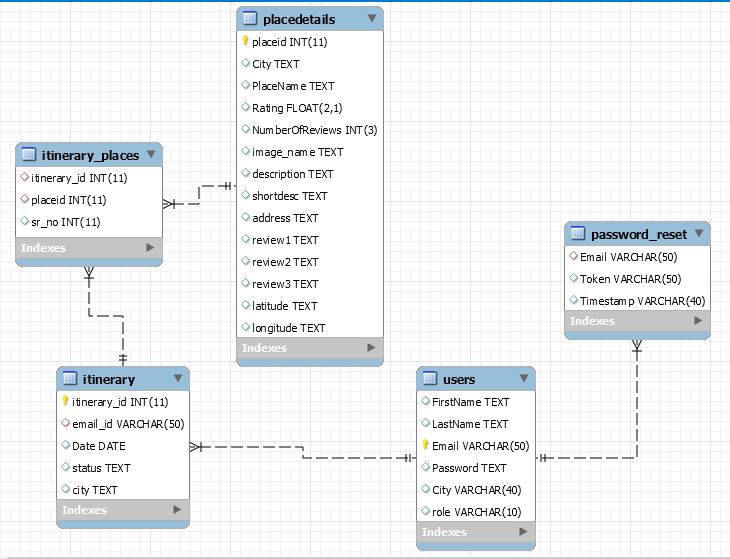
**Prachi Bari**

**Project Background:**

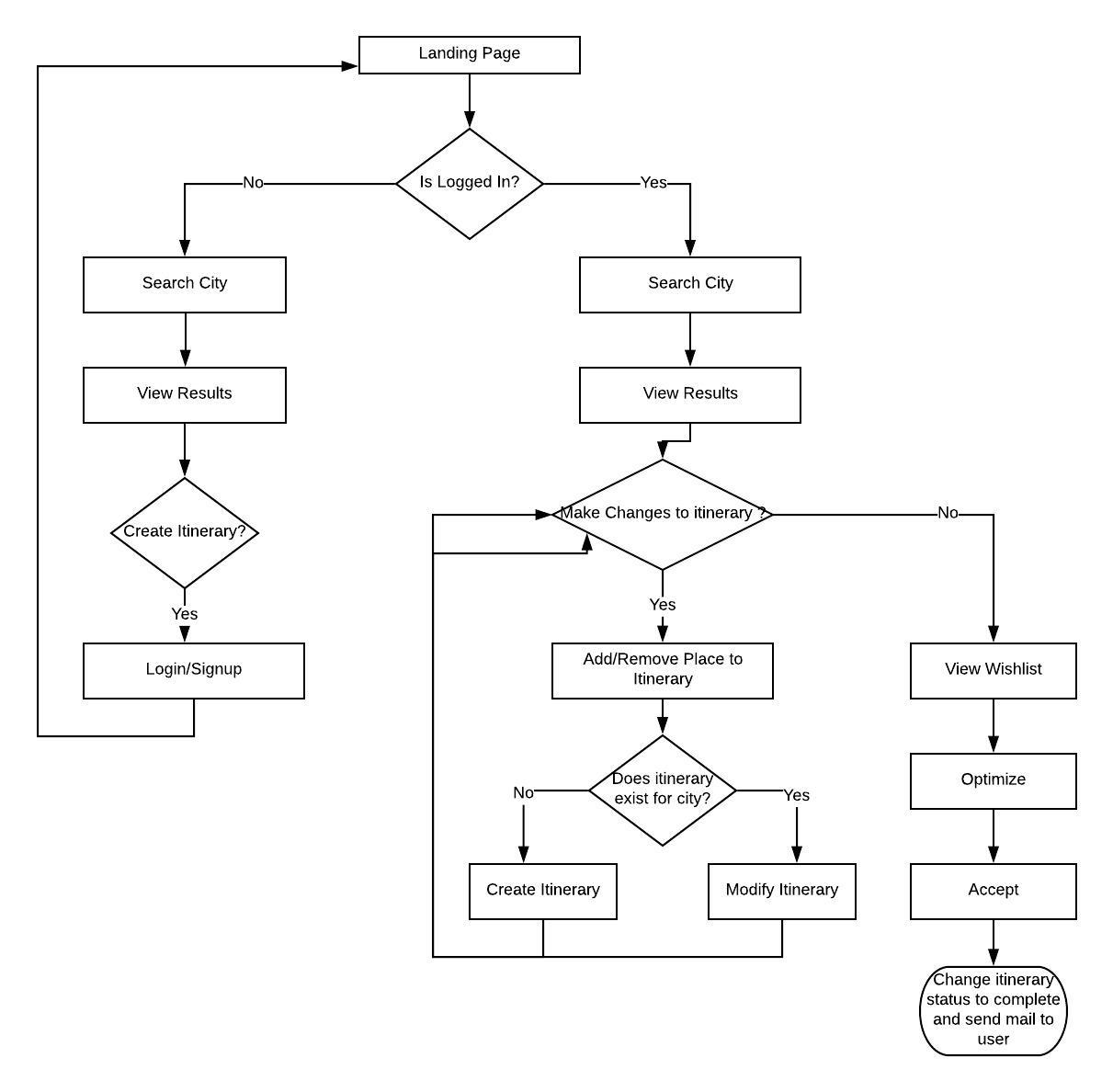
* The main idea behind this project was to create a product which could mimic a shopping website experience to enable users to plan their itinerary.
* Our product provides an intuitive experience for the end users and allows them to create an itinerary for the city of their choice with minimal clicks.
* We have tried to create a one-page solution for the end users so they don’t have to experience the inconvenience of being redirected to different pages on each click.

**Design:**

Database Design:

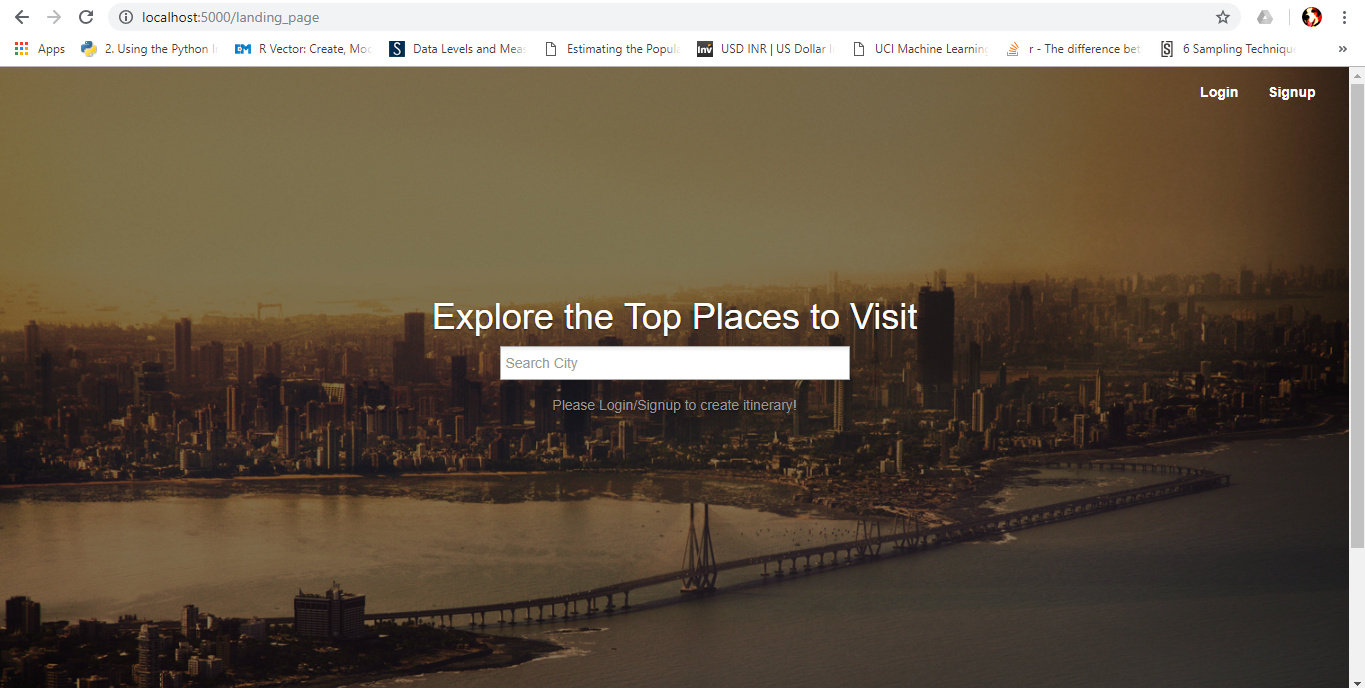


Application Flow:

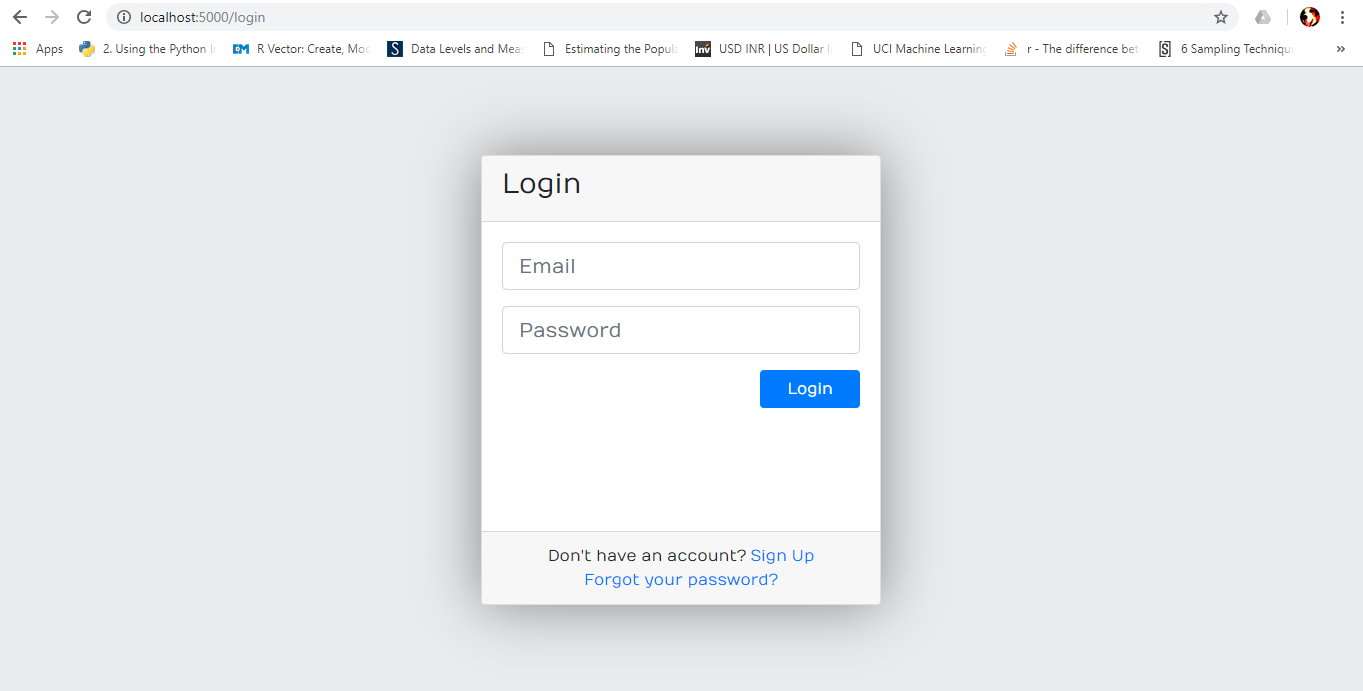


User Interface Design:

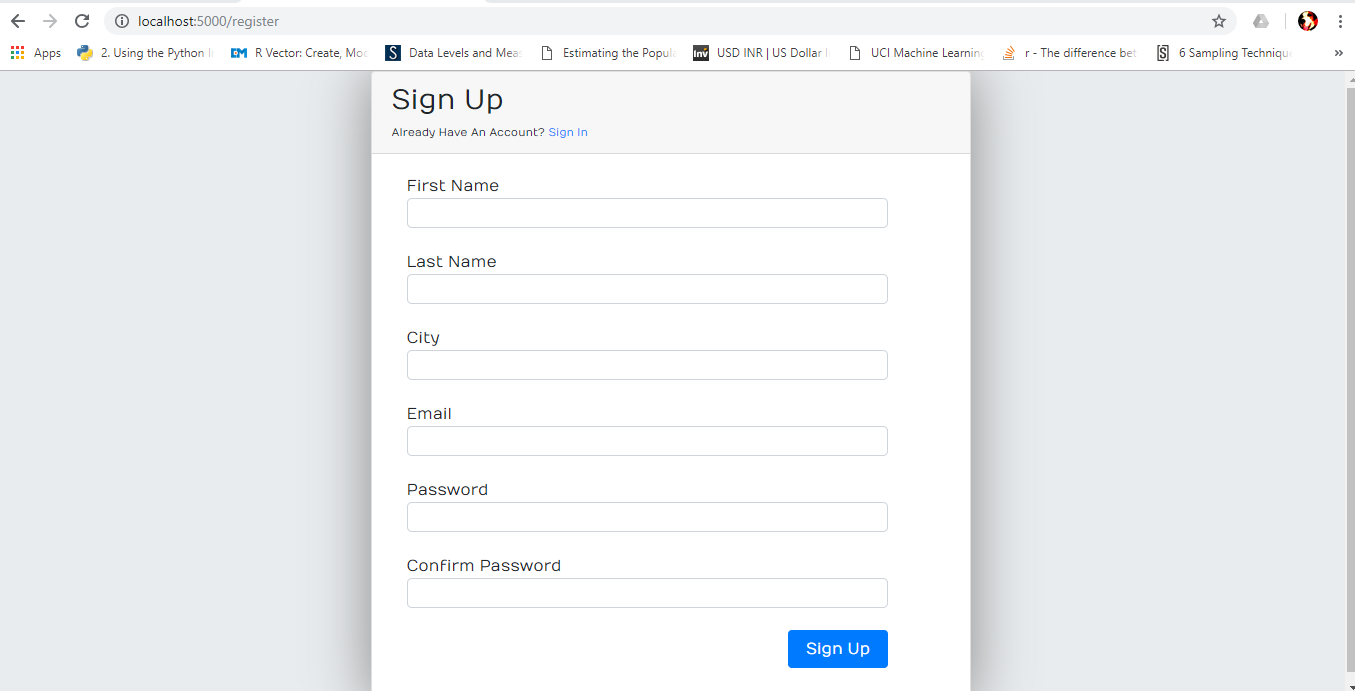
* Landing Page



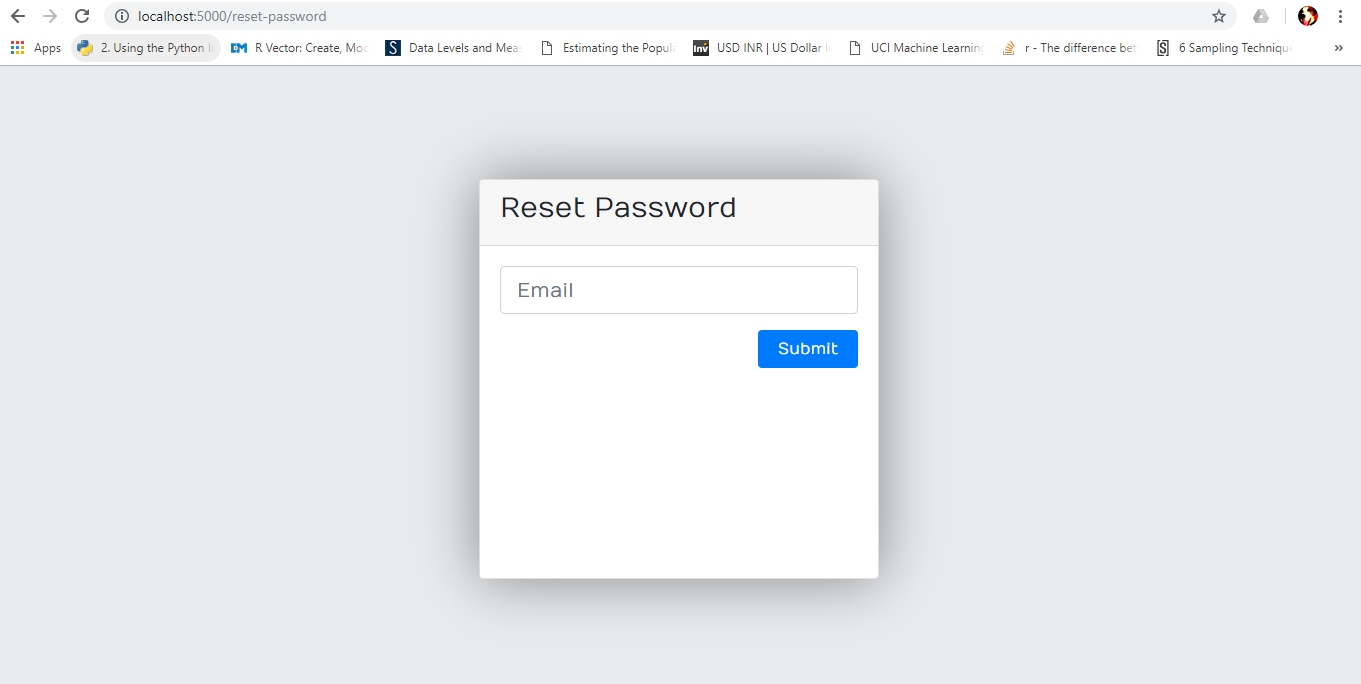
* Login Page



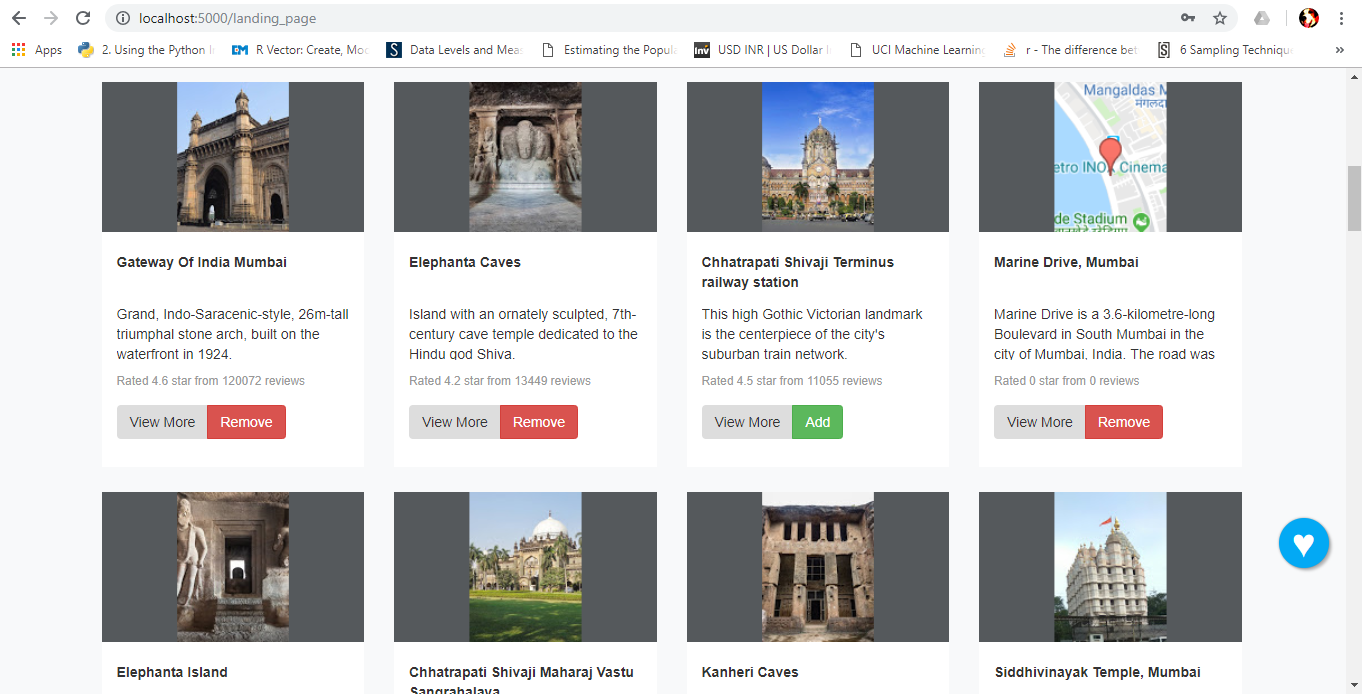
* Signup Page



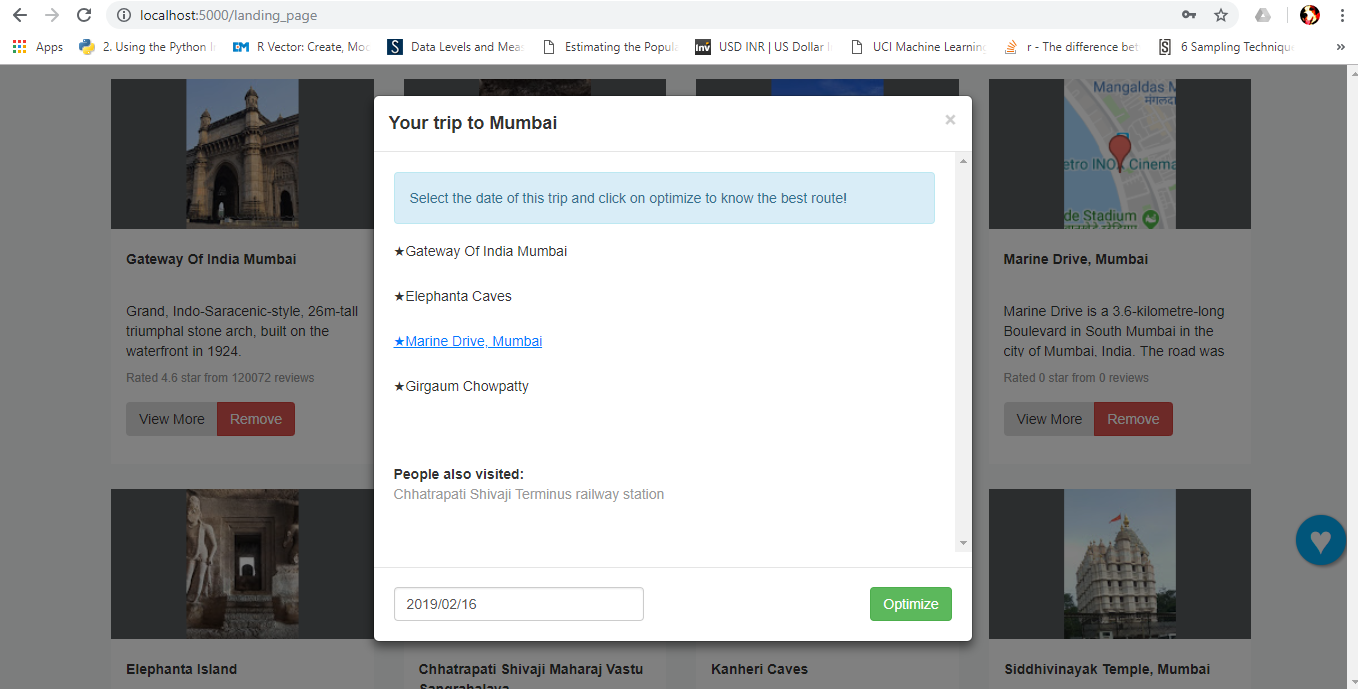
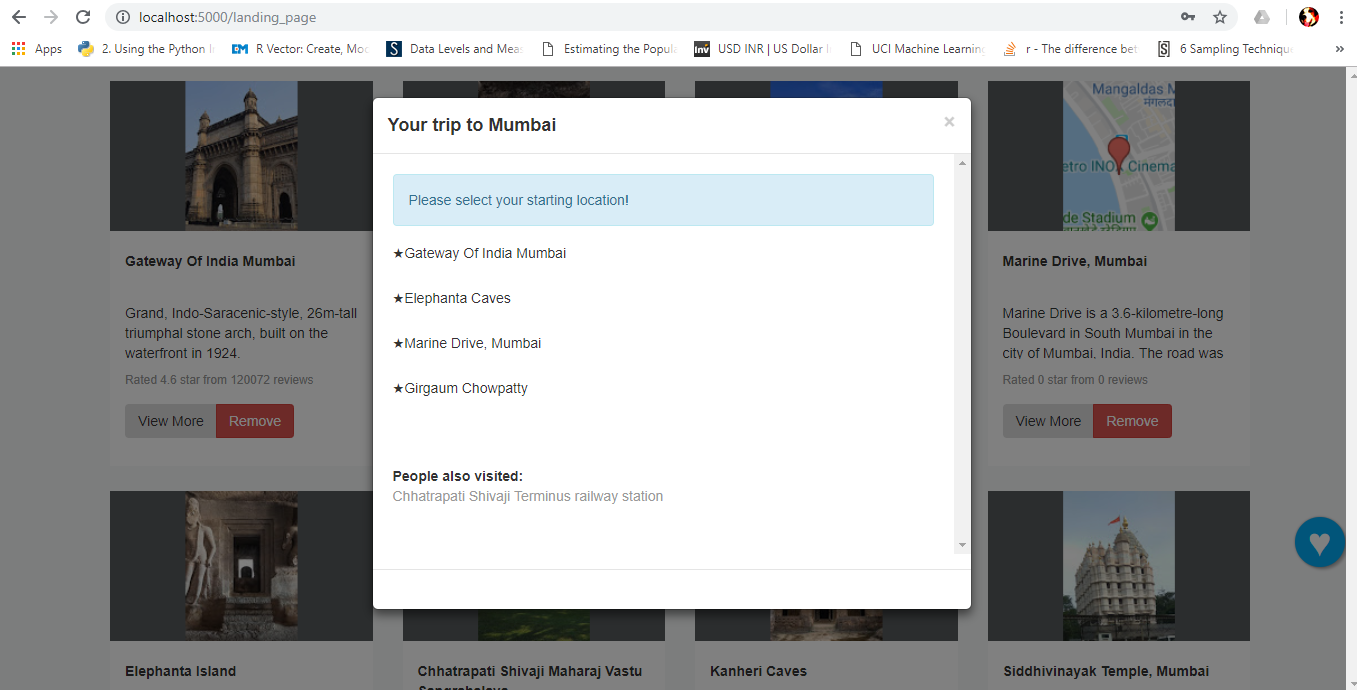
* Forgot Password



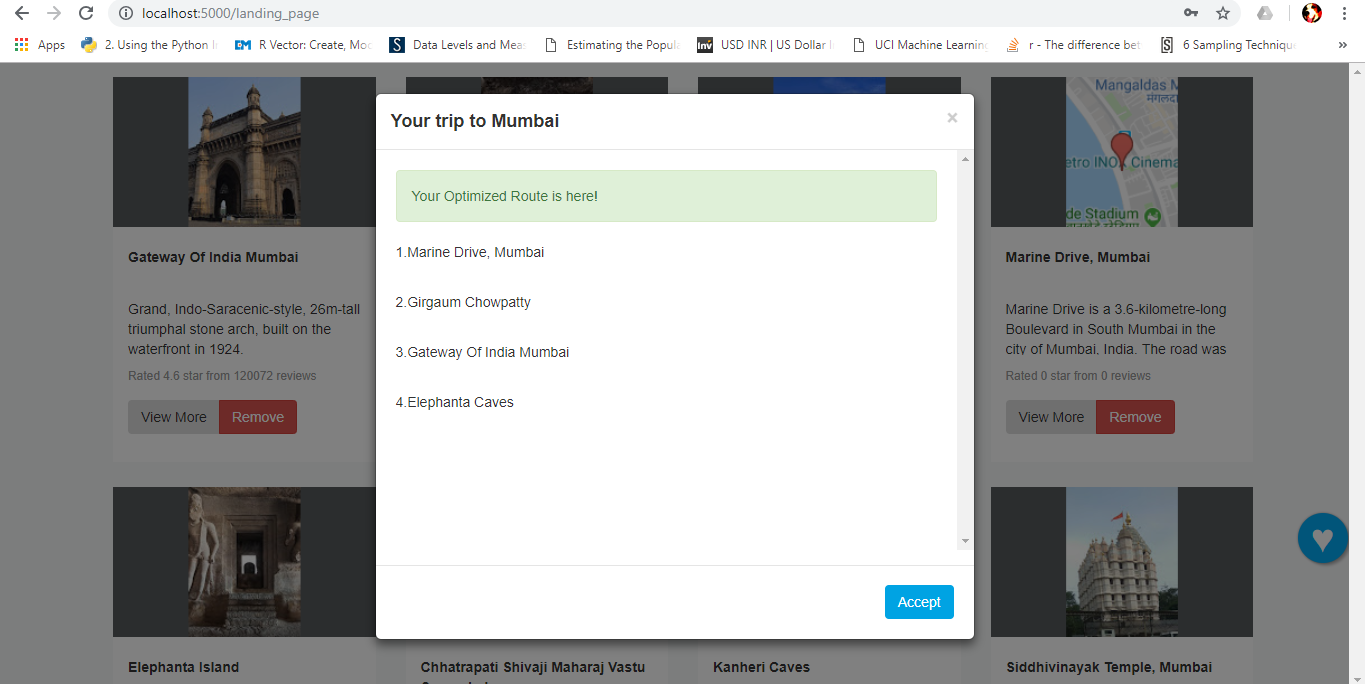
* View Places



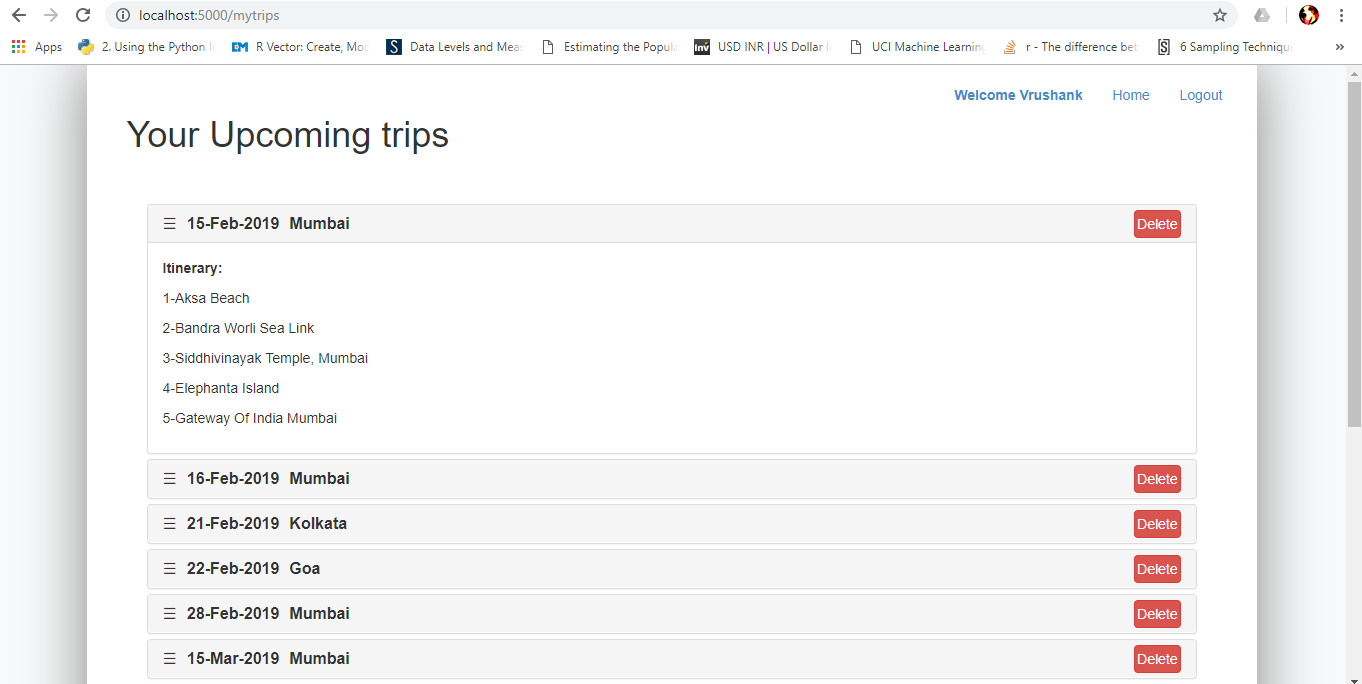
* View Itinerary



* Optimize Itinerary



* My Trips



Modules:

* Login
* Signup
* Forgot Password
* Search Places
* Add/Remove Places from Itinerary
* Optimize Route
* Recommend Places
* Finalize Itinerary
* Delete Itinerary
* View created trips

**Implementation:**

Libraries Used:

* Backend
* Flask and its extentions
* Selenium
* BeautifulSoup
* Pandas
* Numpy
* MySql
* Geopy
* Traceback
* Front End
* HTML
* jQuery
* Bootstrap CSS

Views:

* landing\_page.html(URL=/landing\_page)
* register.html(URL=/register)
* login.html(URL= /login)
* mytrips.html(URL= /mytrips)
* reset\_request.html(URL= /reset-password)
* reset\_token.html(URL= /reset-password/<token>)

REST API’s:

* /viewplace:
  + Request Method: GET
  + This method takes the city name as input and provides the list of places in that city as json output
* /searchCity:
  + Request Method: GET
  + This method takes the search string as input and returns the list of cities matching that search string as json output
* /addPlaceToItinerary:
  + Request Method: POST
  + This method takes the city, placeid and itinerary\_id as input and adds the placeid to the itinerary id. If itinerary id is not passed and it is not present in database, it will create a new itinerary id for that city.
* /fetchCitywiseWishlist:
  + Request Method: GET
  + This method takes the city as input and returns the list of place ids for which user has shown interest.
* /removePlaceFromItinerary:
  + Request Method: POST
  + This method accepts the place id and itinerary id as input and deletes the place id from the itinerary.
* /populateItineraryModal:
  + Request Method: GET
  + This method takes the city as input and returns the list of places the user has added to wishlist in that city.
* /optimizeItinerary:
  + Request Method: GET
  + This method accept the place id of the starting point and a list of places to visit and returns the place id list after optimizing.
* /checkIfOtherItineraryPresent:
  + Request Method: GET
  + This method accepts the start date and returns if the user has any itinerary present in the database for that date.
* /finalizeItinerary:
  + Request Method: POST
  + This method accepts the start date, itinerary id, place ids list and updates the order in Database and changes the status of that itinerary to complete. It then sends a mail to user about the itinerary details.
* /getSuggestions:
  + Request Method: GET
  + This method takes the city and place ids selected by user as input and returns the recommendation of places by taking into account what other users have booked.
* /delItinerary:
  + Request Method: POST
  + This method takes the itinerary id as input and deletes it from the database.

Business Logic Functions:

* verify\_reset\_token(token):This function verifies if the token received in URL is valid and not expired(Token is valid for 2 hours)
* find\_optimized\_route(to\_visit,start\_id): This function iteratively calculates the distance between start places and other places by taking the latitude and longitude into consideration and returns the optimized list of places.
* generateQuery(to\_visit,start\_id): This function generates the sql query using the both the lists of place ids passed to it.
* generateItineraryMail(cursor,itineraryid): This function creates the content for the itinerary creation mail and calls the asyncMail function in a new thread for sending the mail.
* recommend\_places(places\_list,city,emailid): This function will identify which places should be recommended to the user based on how other users have created their itineraries.
* asyncMail(app,msg,email): This function is called in a separate thread for sending the itinerary creation mail.