

## EXPERIMENT 3 B

**NAME: Shivam Pawar**

**UID: 2019230068**

**NAME: Vishal Salvi**

**UID: 2019230069**

**NAME: Shreyas Patel**

**UID: 2018130043**

**CLASS: TE COMPS**

**BATCH: C**

**Aim:** Creating basic web application by using framework.

**Theory:**

**Databases**

**Django officially supports the following databases:**

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

There are also a number of database backends provided by third parties.

Django attempts to support as many features as possible on all database backends. However, not all database backends are alike, and we've had to make design decisions on which features to support and which assumptions we can make safely.

### **Optimizing PostgreSQL's configuration**

Django needs the following parameters for its database connections:

- **client\_encoding:** 'UTF8',
- **default\_transaction\_isolation:** 'read committed' by default, or the value set in the connection options (see below),
- **timezone:**
  - when **USE\_TZ** is **True**, 'UTC' by default, or the **TIME\_ZONE** value set for the connection,
  - when **USE\_TZ** is **False**, the value of the global **TIME\_ZONE** setting.

If these parameters already have the correct values, Django won't set them for every new connection, which improves performance slightly. You can configure them directly in **postgresql.conf** or more conveniently per database user with ALTER ROLE.

Django will work just fine without this optimization, but each new connection will do some additional queries to set these parameters.

## MySQL

### Version support

Django supports MySQL 5.6 and higher.

Django's **inspectdb** feature uses the **information\_schema** database, which contains detailed data on all database schemas.

Django expects the database to support Unicode (UTF-8 encoding) and delegates to it the task of enforcing transactions and referential integrity. It is important to be aware of the fact that the two latter ones aren't actually enforced by MySQL when using the MyISAM storage engine, see the next section.

### Storage engines

MySQL has several storage engines. You can change the default storage engine in the server configuration.

MySQL's default storage engine is InnoDB. This engine is fully transactional and supports foreign key references. It's the recommended choice. However, the InnoDB autoincrement counter is lost on a MySQL restart because it does not remember the **AUTO\_INCREMENT** value, instead recreating it as "max(id)+1". This may result in an inadvertent reuse of **AutoField** values.

The main drawbacks of MyISAM are that it doesn't support transactions or enforce foreign-key constraints.

### MySQL DB API Drivers

MySQL has a couple drivers that implement the Python Database API described in **PEP 249**:

- mysqlclient is a native driver. It's **the recommended choice**.
- MySQL Connector/Python is a pure Python driver from Oracle that does not require the MySQL client library or any Python modules outside the standard library.

These drivers are thread-safe and provide connection pooling.

In addition to a DB API driver, Django needs an adapter to access the database drivers from its ORM. Django provides an adapter for mysqlclient while MySQL Connector/Python includes its own.

#### *mysqlclient*

Django requires mysqlclient 1.4.0 or later.

#### *MySQL Connector/Python*

MySQL Connector/Python is available from the download page. The Django adapter is available in versions 1.1.X and later. It may not support the most recent releases of Django.

## Time zone definitions

If you plan on using Django's timezone support, use `mysql_tzinfo_to_sql` to load time zone tables into the MySQL database. This needs to be done just once for your MySQL server, not per database.

## Creating your database

You can create your database using the command-line tools and this SQL:

```
CREATE DATABASE <dbname> CHARACTER SET utf8;
```

This ensures all tables and columns will use UTF-8 by default.

## *Collation settings*

The collation setting for a column controls the order in which data is sorted as well as what strings compare as equal. It can be set on a database-wide level and also per-table and per-column. This is documented thoroughly in the MySQL documentation. In all cases, you set the collation by directly manipulating the database tables; Django doesn't provide a way to set this on the model definition.

By default, with a UTF-8 database, MySQL will use the **utf8\_general\_ci** collation. This results in all string equality comparisons being done in a *case-insensitive* manner. That is, "Fred" and "freD" are considered equal at the database level. If you have a unique constraint on a field, it would be illegal to try to insert both "aa" and "AA" into the same column, since they compare as equal (and, hence, non-unique) with the default collation. If you want case-sensitive comparisons on a particular column or table, change the column or table to use the **utf8\_bin** collation.

Please note that according to MySQL Unicode Character Sets, comparisons for the **utf8\_general\_ci** collation are faster, but slightly less correct, than comparisons for **utf8\_unicode\_ci**. If this is acceptable for your application, you should use **utf8\_general\_ci** because it is faster. If this is not acceptable (for example, if you require German dictionary order), use **utf8\_unicode\_ci** because it is more accurate.

## Customizing authentication in Django

The authentication that comes with Django is good enough for most common cases, but you may have needs not met by the out-of-the-box defaults. Customizing authentication in your projects requires understanding what points of the provided system are extensible or replaceable. This document provides details about how the auth system can be customized.

Authentication backends provide an extensible system for when a username and password stored with the user model need to be authenticated against a different service than Django's default.

You can give your models custom permissions that can be checked through Django's authorization system.

You can extend the default **User** model, or substitute a completely customized model.

## Other authentication sources

There may be times you have the need to hook into another authentication source – that is, another source of usernames and passwords or authentication methods.

For example, your company may already have an LDAP setup that stores a username and password for every employee. It'd be a hassle for both the network administrator and the users themselves if users had separate accounts in LDAP and the Django-based applications.

So, to handle situations like this, the Django authentication system lets you plug in other authentication sources. You can override Django's default database-based scheme, or you can use the default system in tandem with other systems.

See the authentication backend reference for information on the authentication backends included with Django.

Specifying authentication backends:

Behind the scenes, Django maintains a list of “authentication backends” that it checks for authentication. When somebody calls `django.contrib.auth.authenticate()` – as described in How to log a user in – Django tries authenticating across all of its authentication backends. If the first authentication method fails, Django tries the second one, and so on, until all backends have been attempted.

The list of authentication backends to use is specified in the **AUTHENTICATION\_BACKENDS** setting. This should be a list of Python path names that point to Python classes that know how to authenticate. These classes can be anywhere on your Python path.

By default, **AUTHENTICATION\_BACKENDS** is set to:

```
['django.contrib.auth.backends.ModelBackend']
```

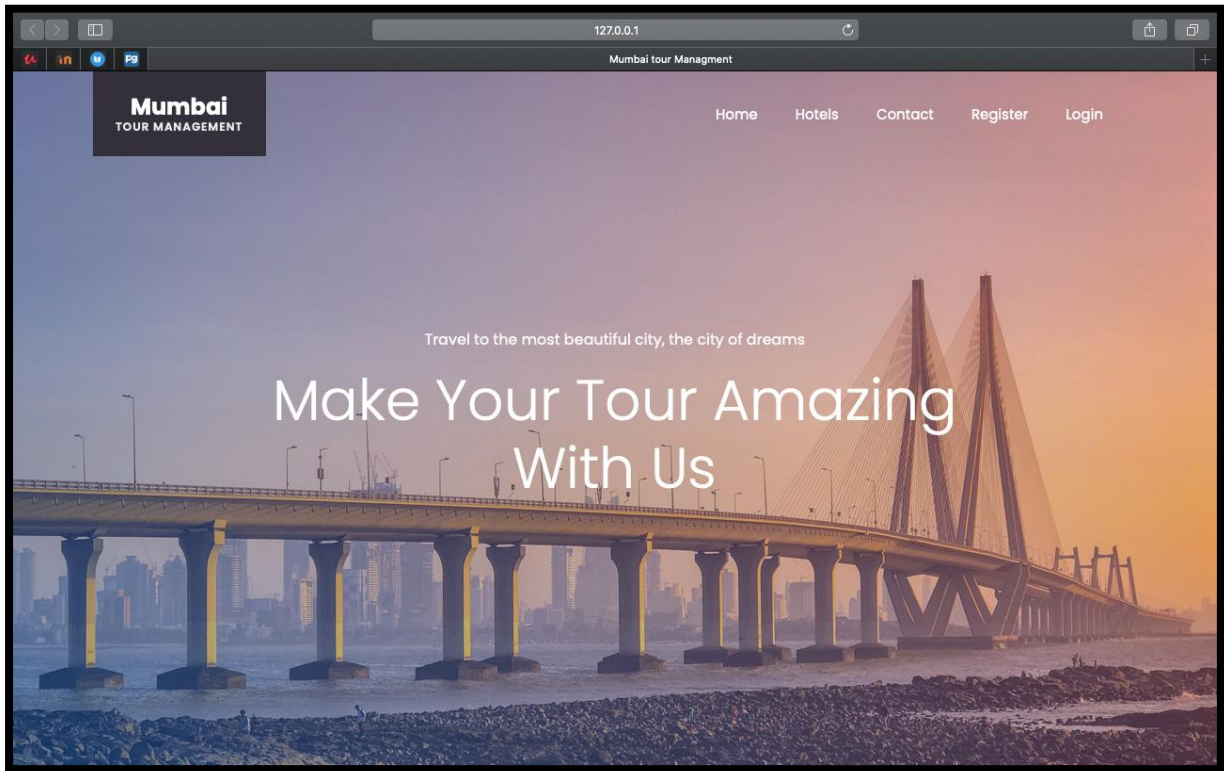
That's the basic authentication backend that checks the Django users database and queries the built-in permissions. It does not provide protection against brute force attacks via any rate limiting mechanism. You may either implement your own rate limiting mechanism in a custom auth backend, or use the mechanisms provided by most Web servers.

The order of **AUTHENTICATION\_BACKENDS** matters, so if the same username and password is valid in multiple backends, Django will stop processing at the first positive match.

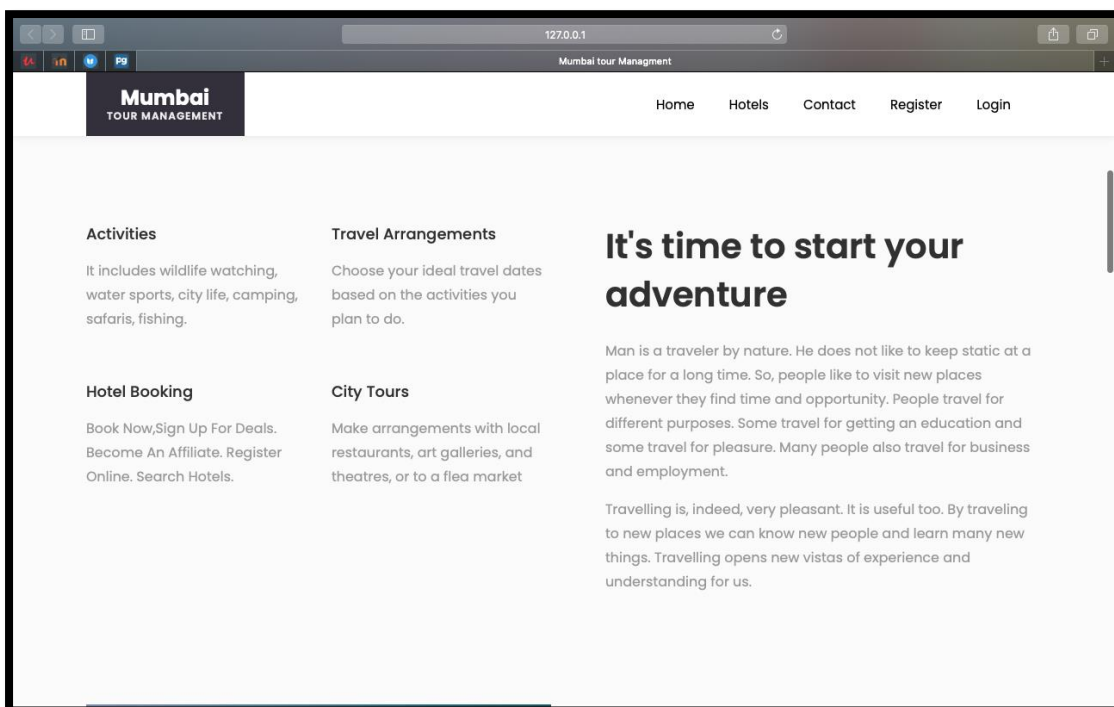
## Screenshot:

## Run Server

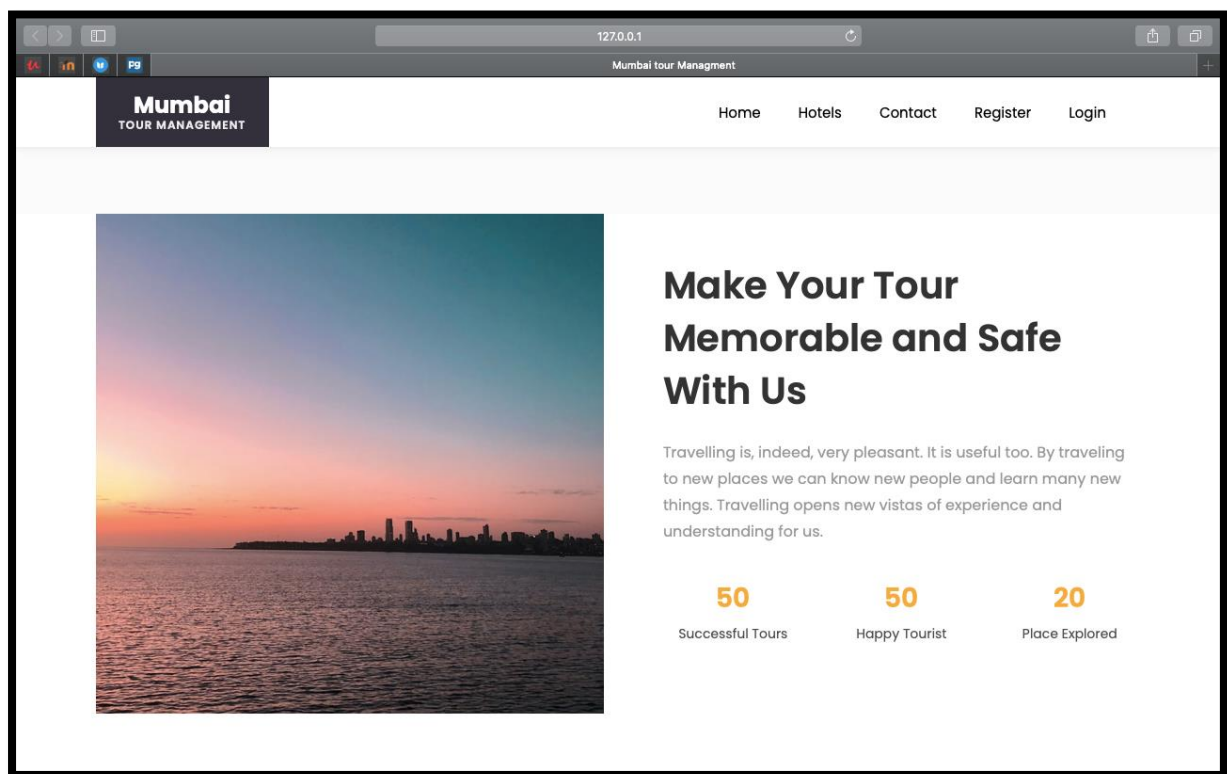
This is the home page of Mumbai Tour and guide management.



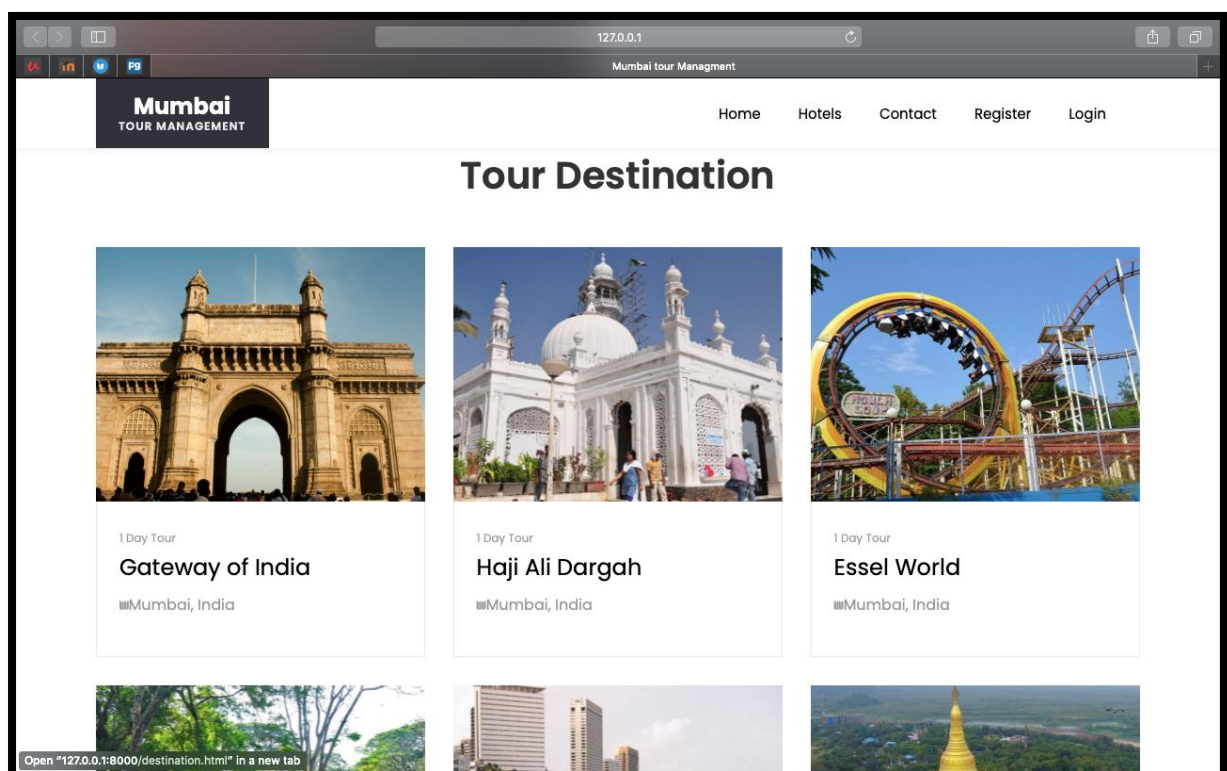
Here user can see whatever activities, tours as well as hotel booking features in system

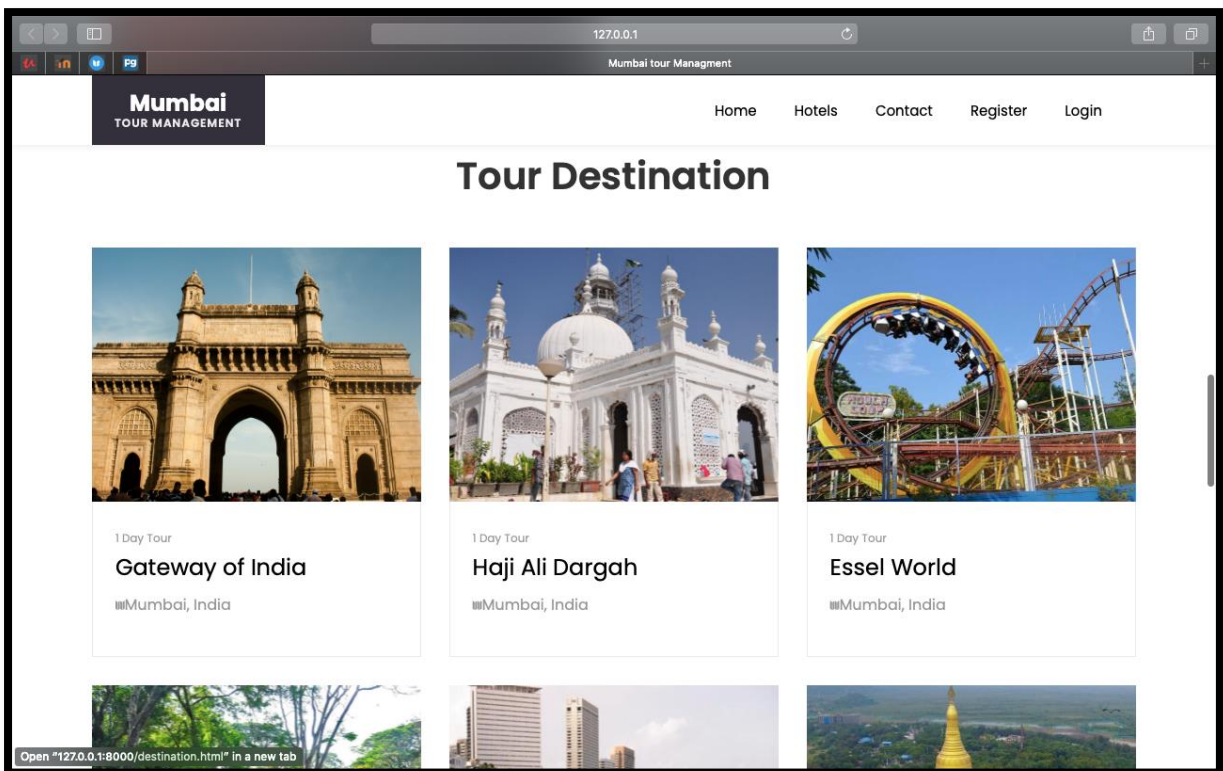
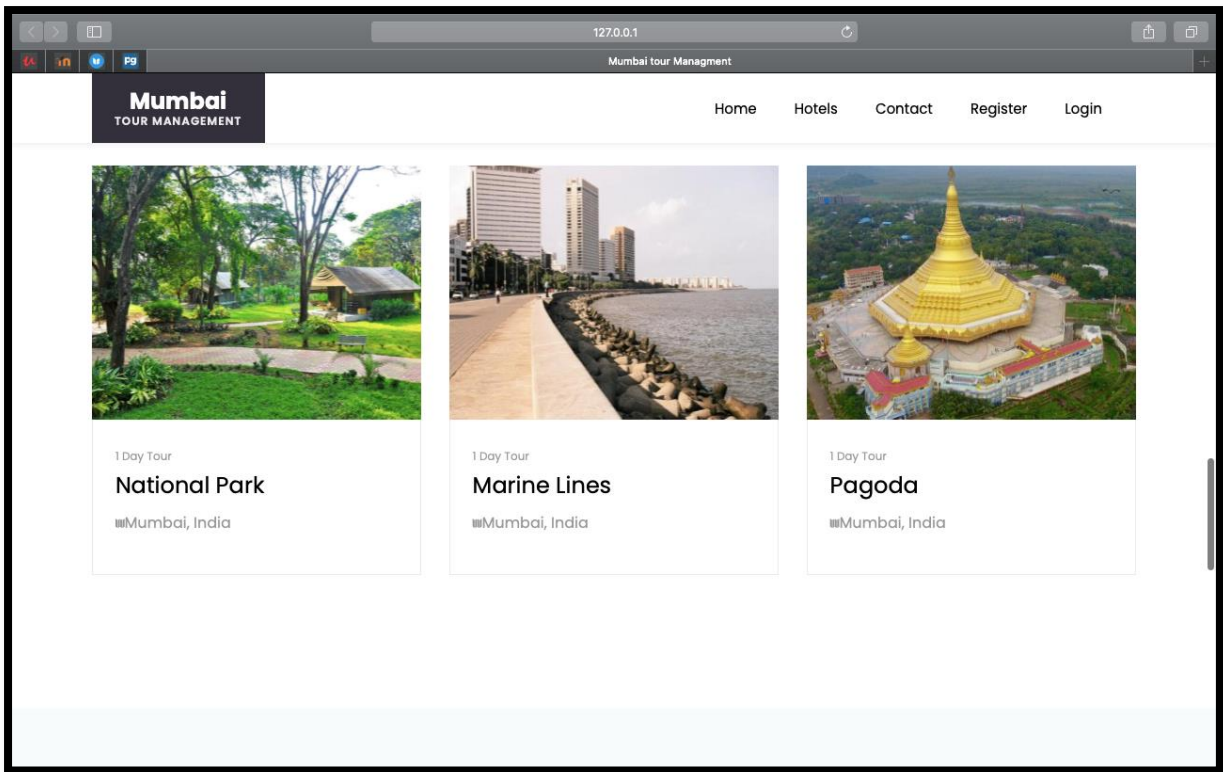


User can see count of happy tourist and place explored by tourist

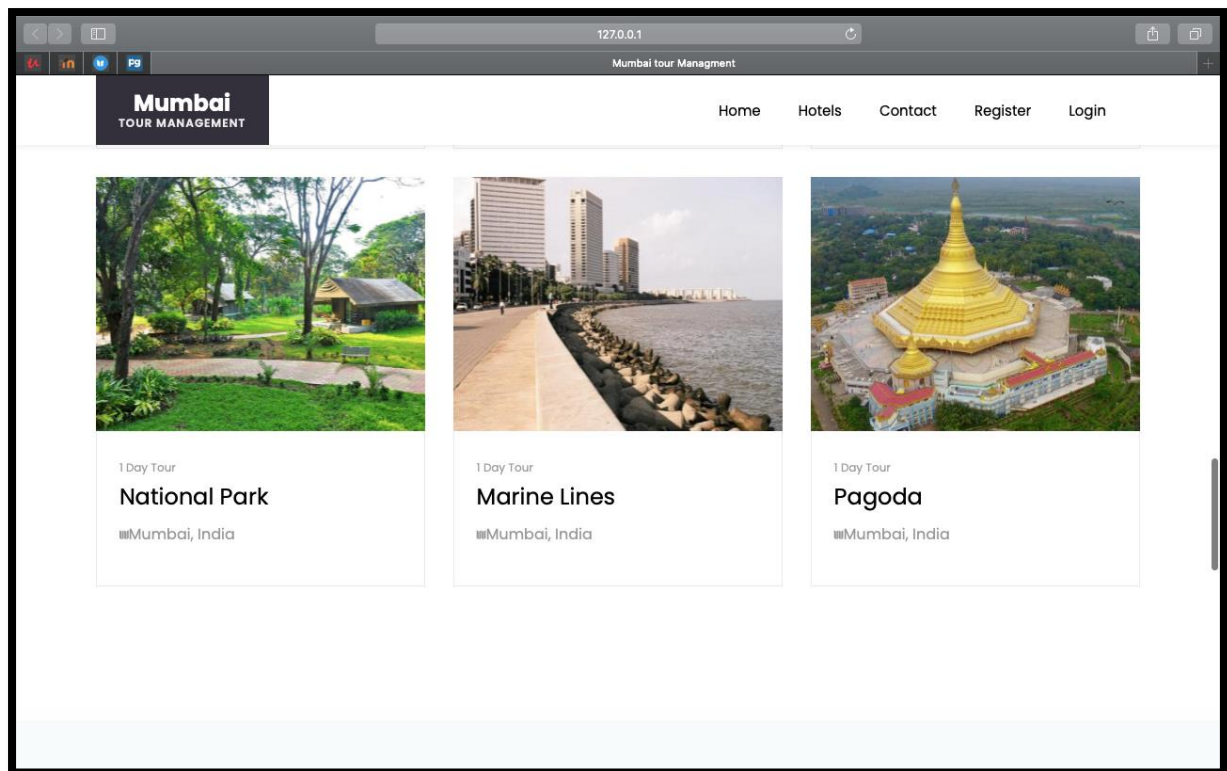


After that user can view tour destination

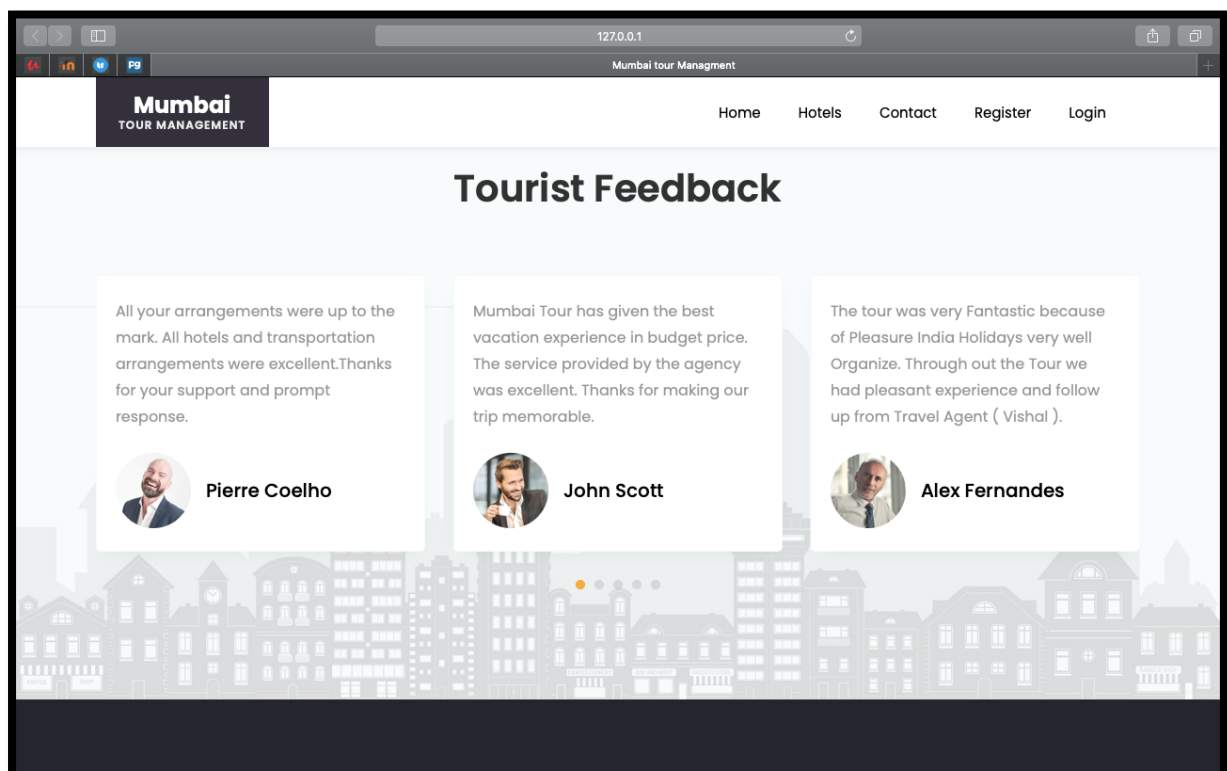






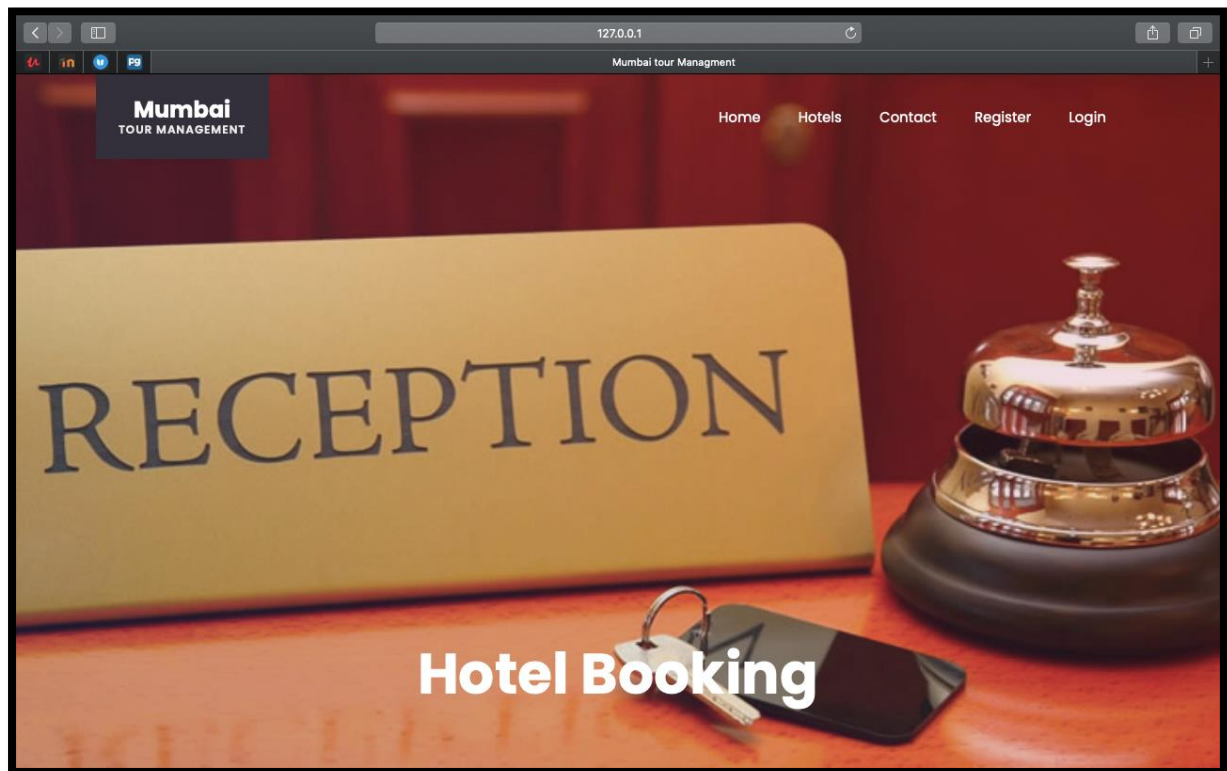


Then user get tourist feedback

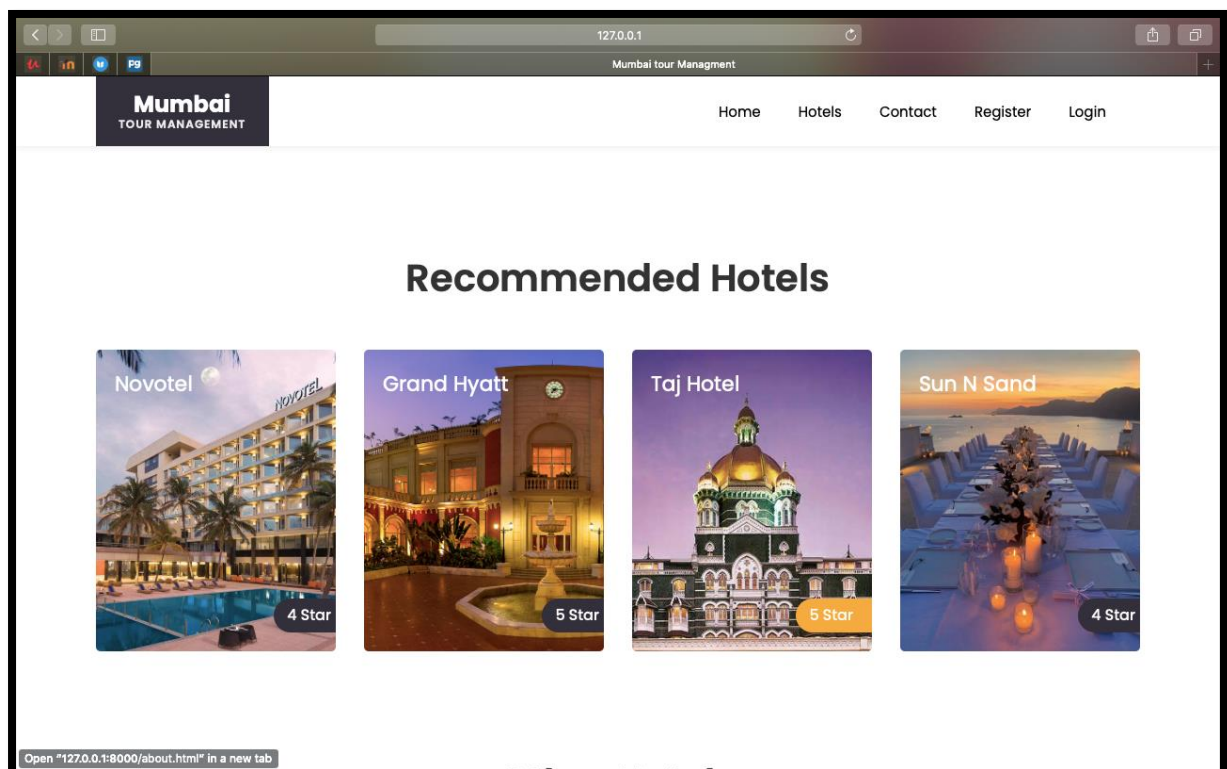




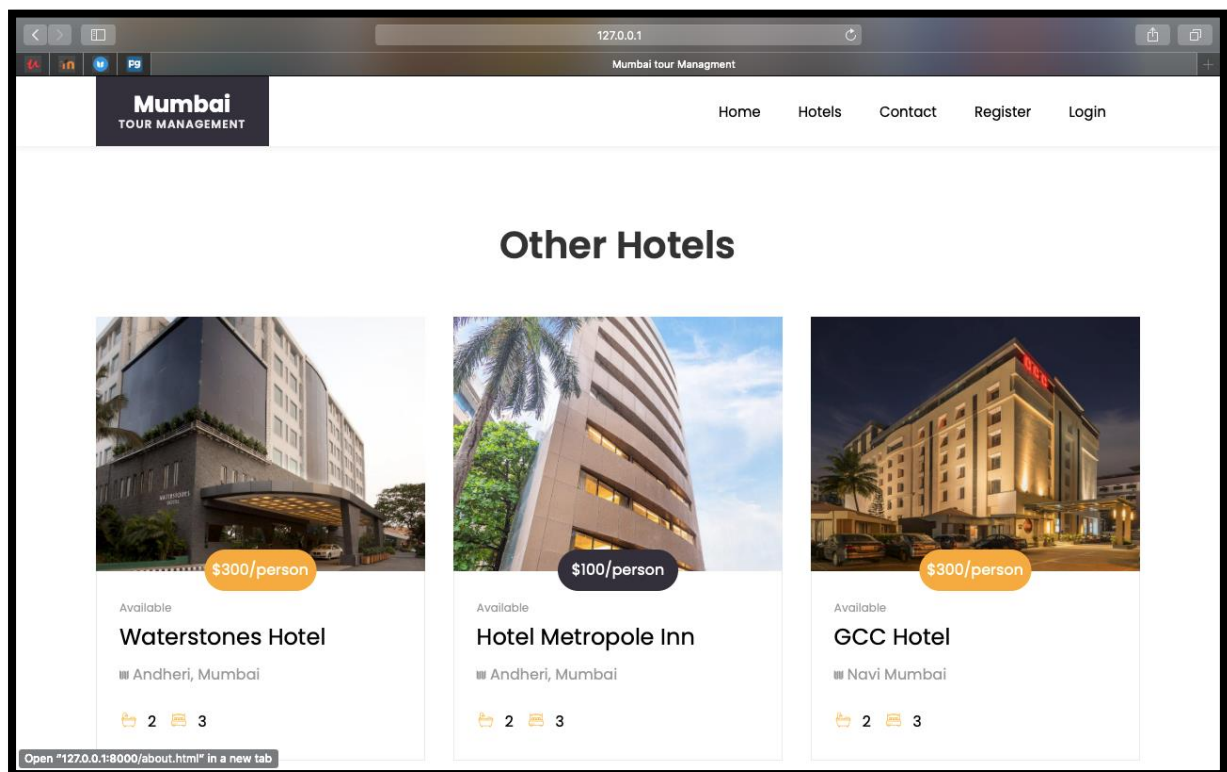
This is Hotel Reception view



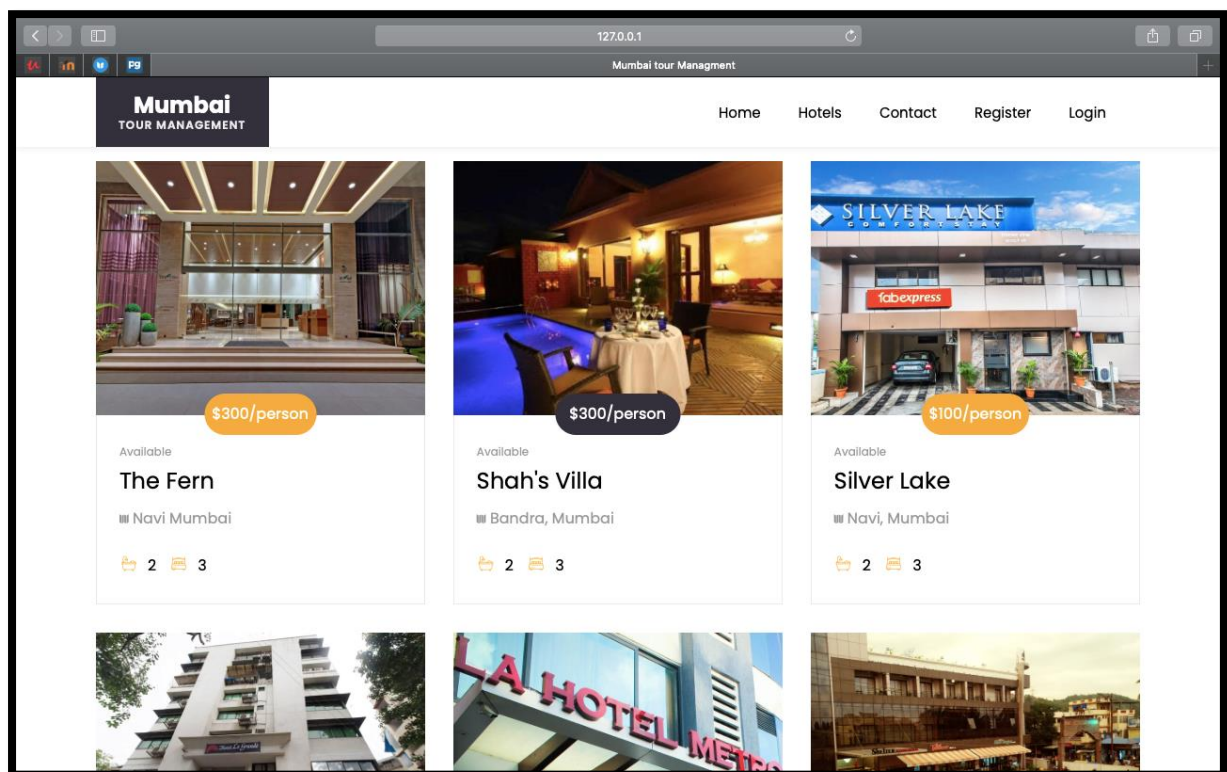
Here, User can see Recommended Hotels which are near by tour location

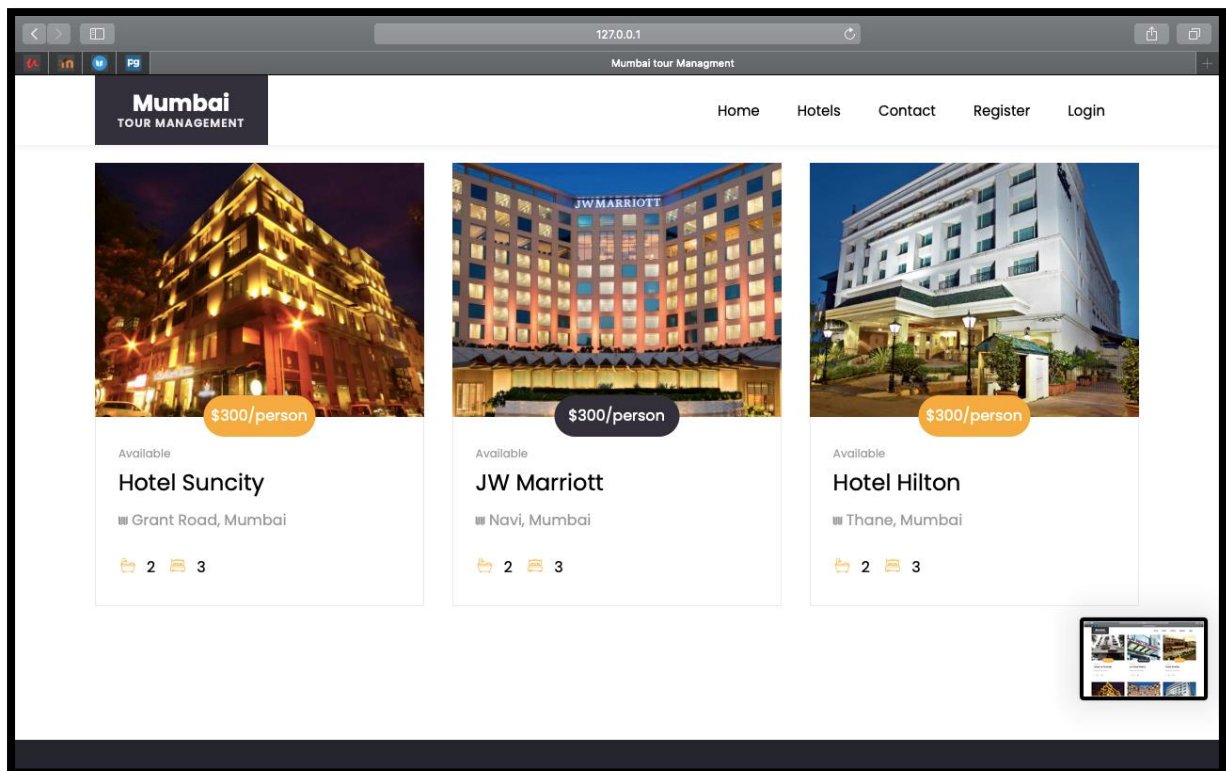
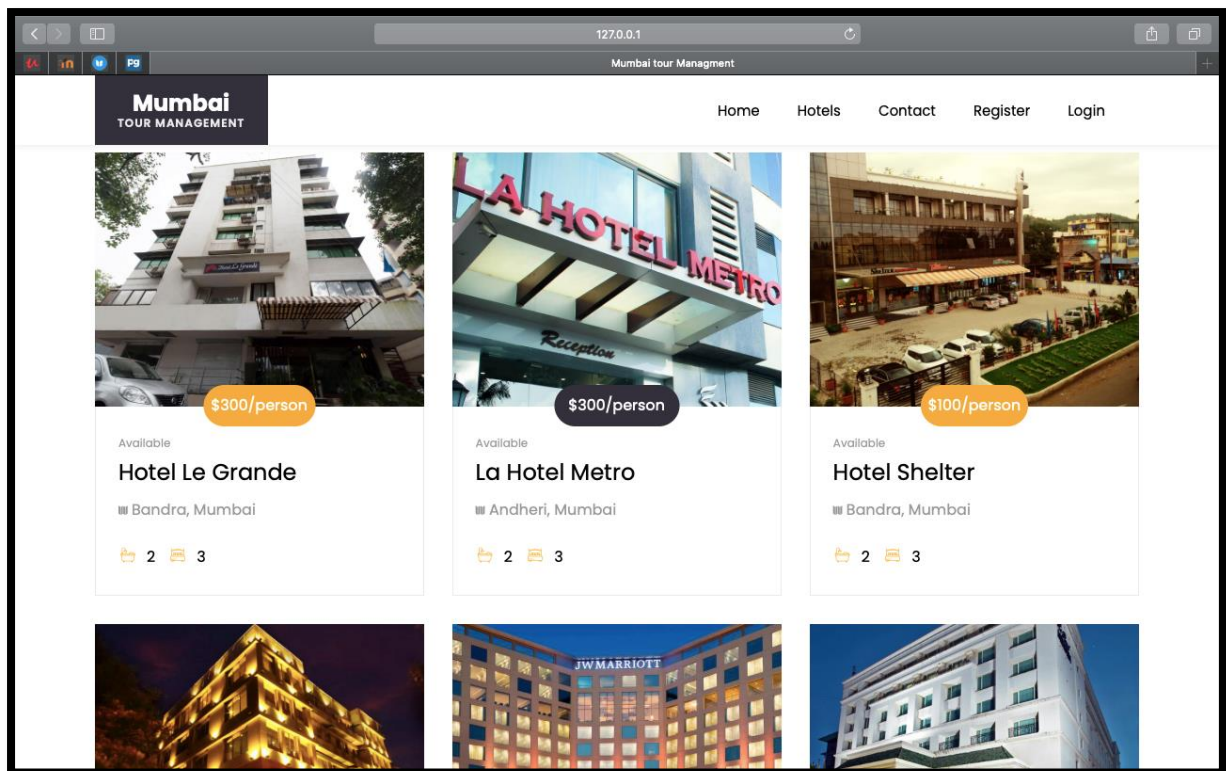


## Other Hotels



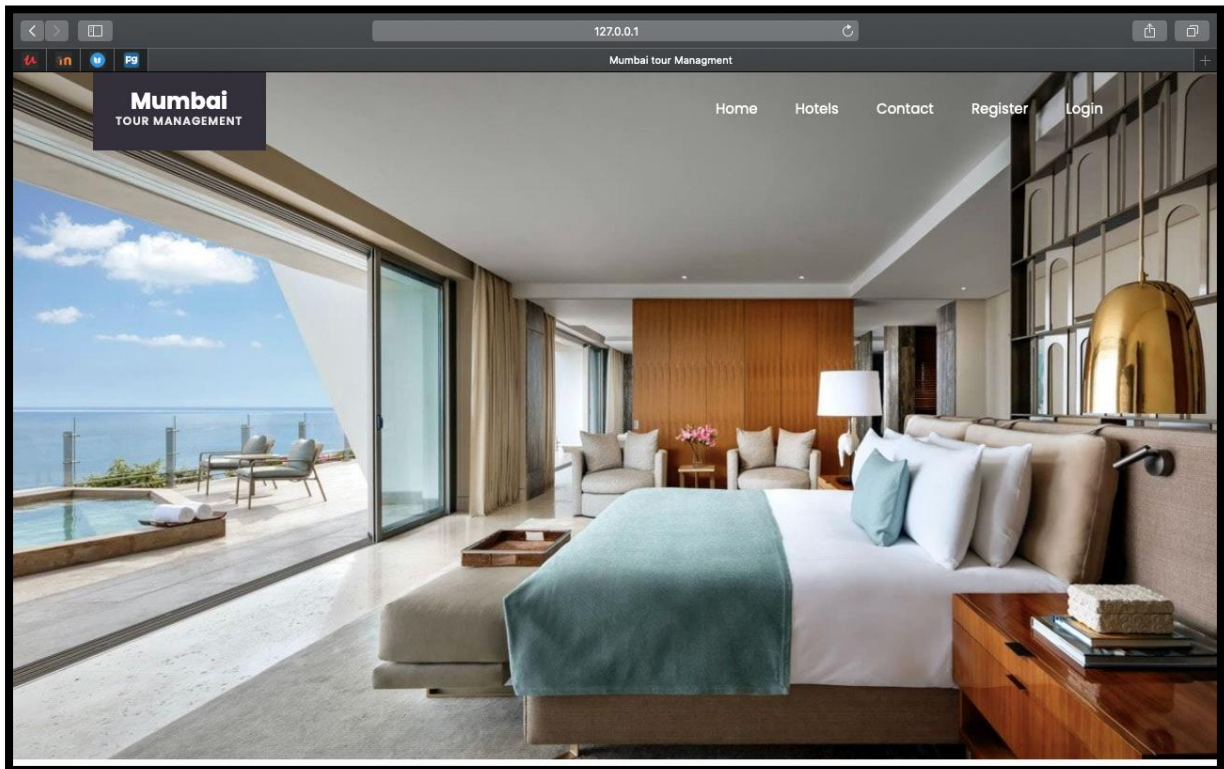
User can view different hotels as well as details about hotel and hotel rating also



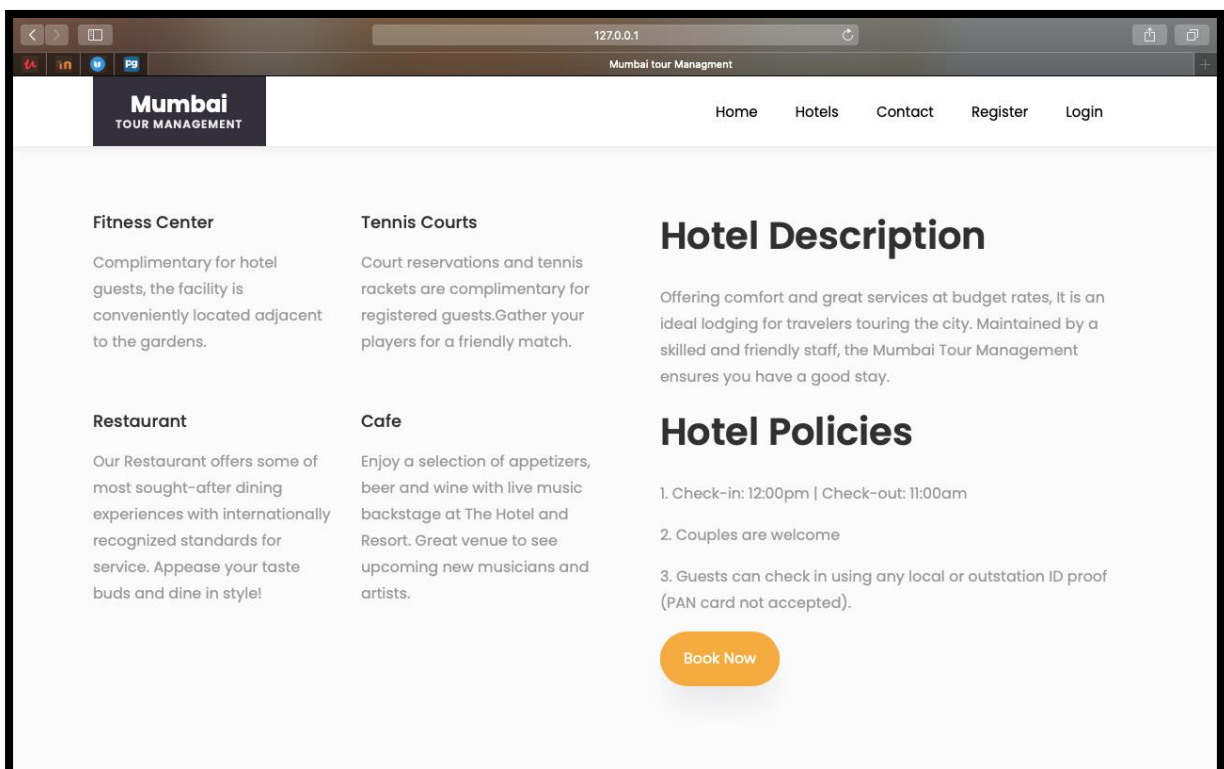




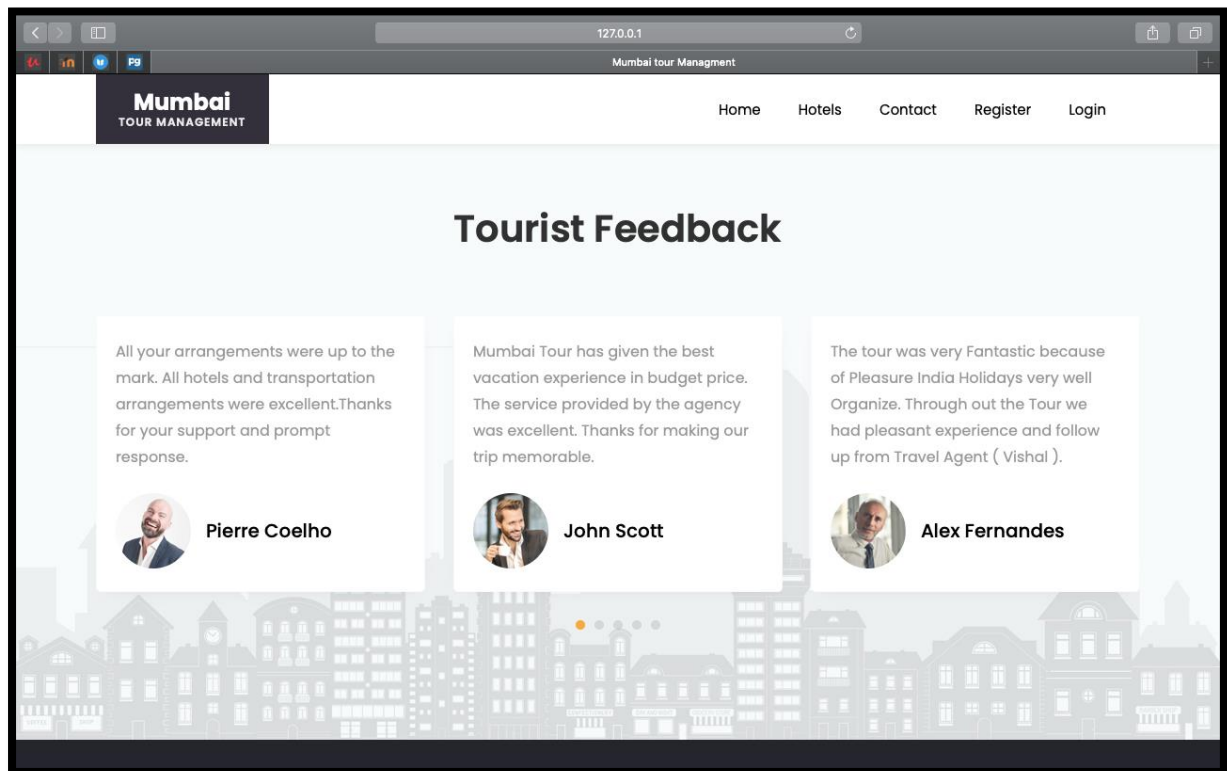
Then, User can view of room in hotel.



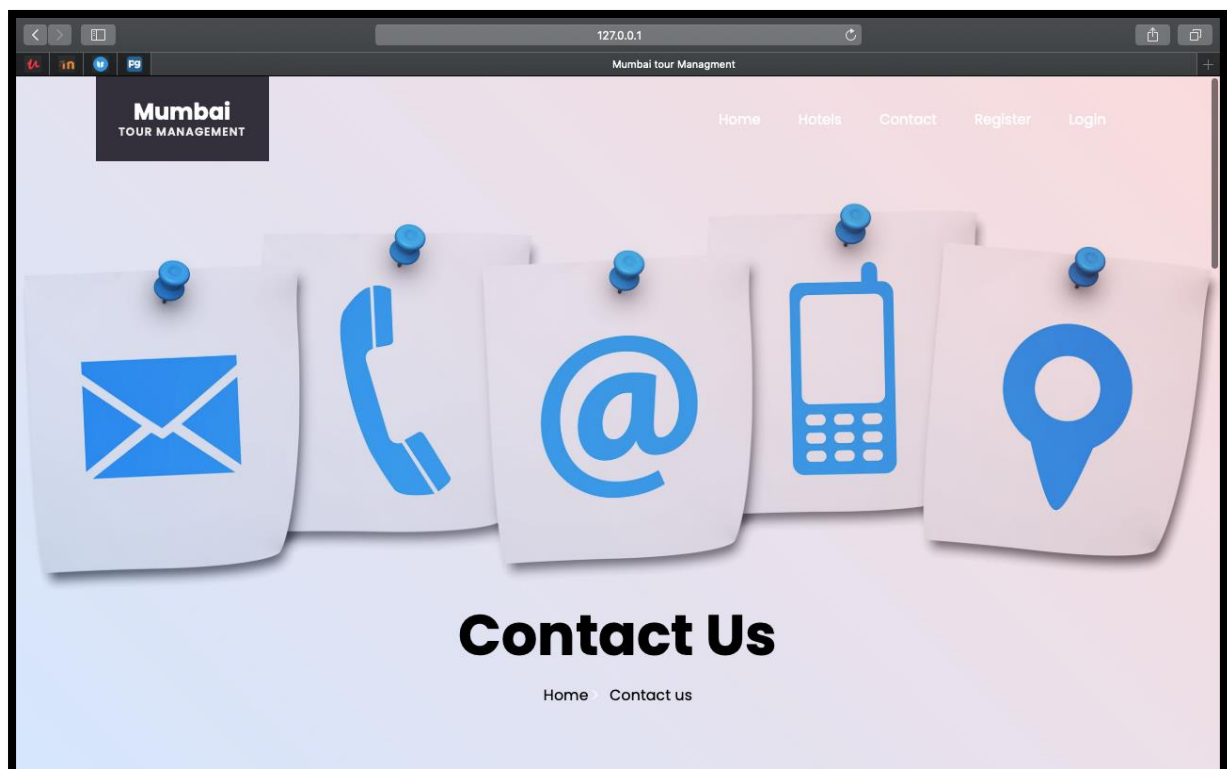
After that can get features of hotel like Fitness centre, Tennis courts, Cafe etc.



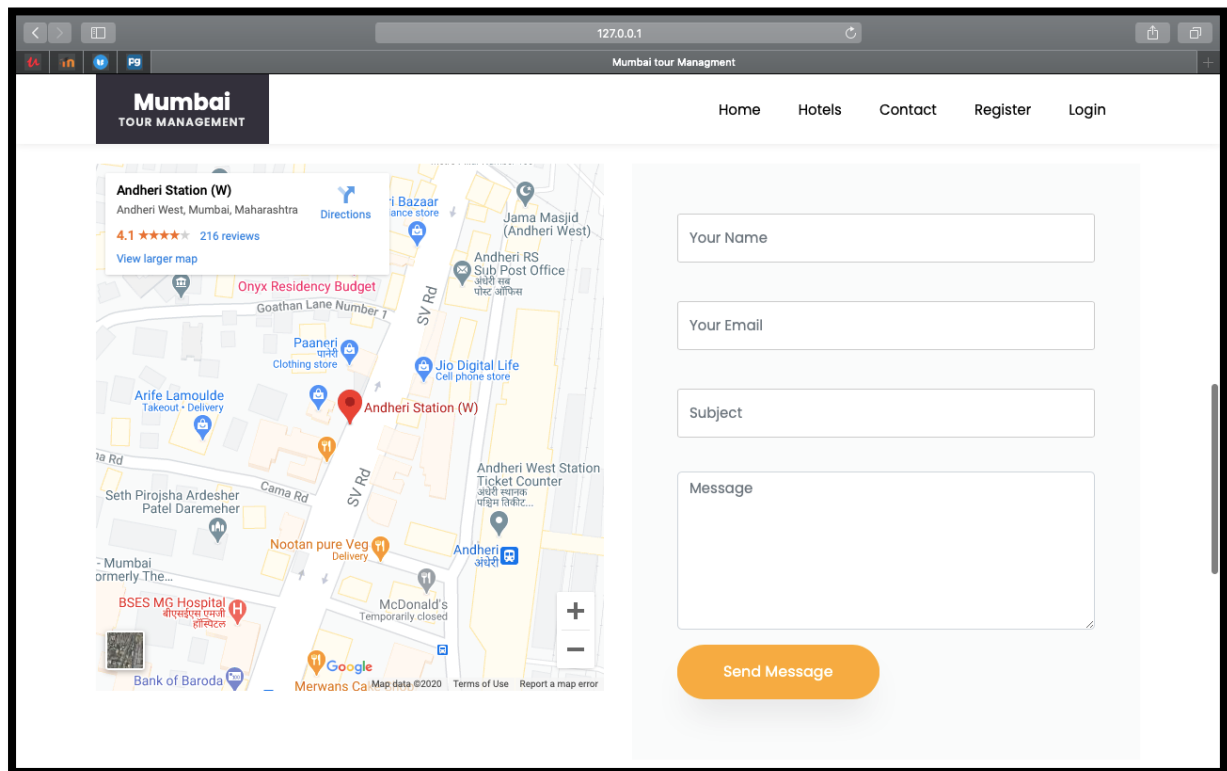
Then user get tourist feedback



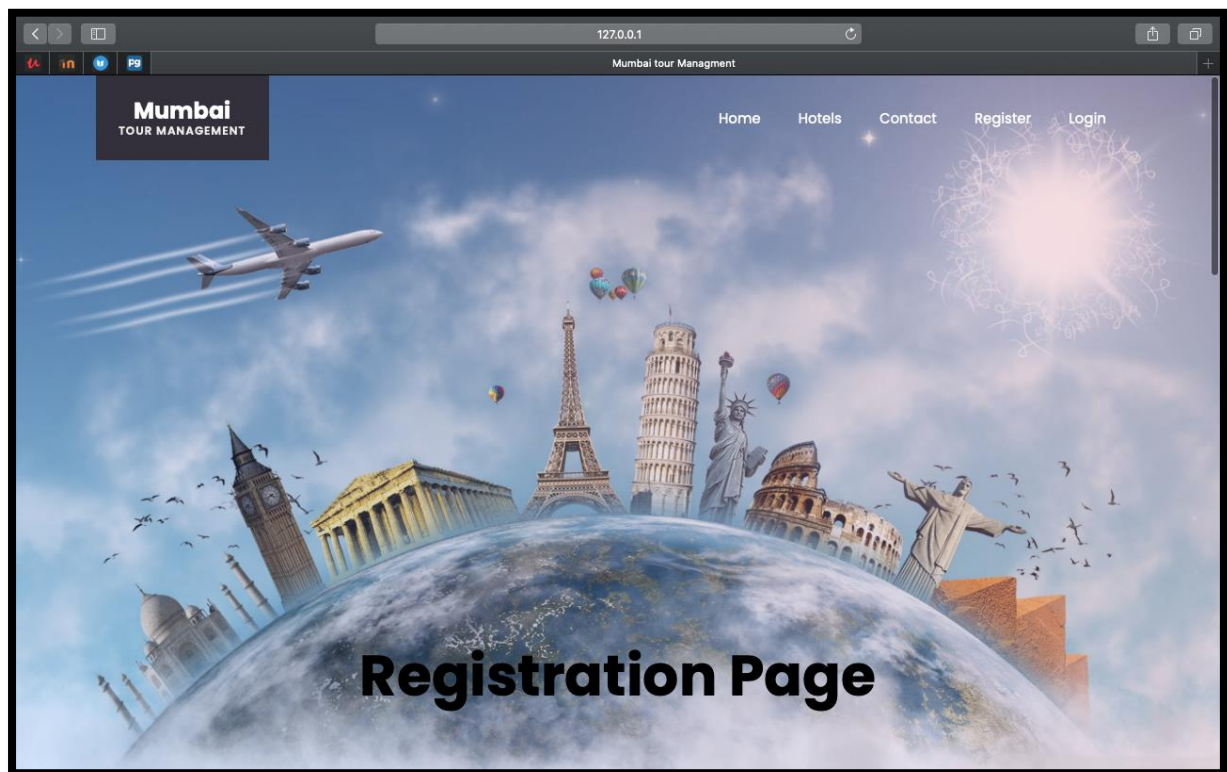
Next is contact page



User can send message to hotel as well as can add review.

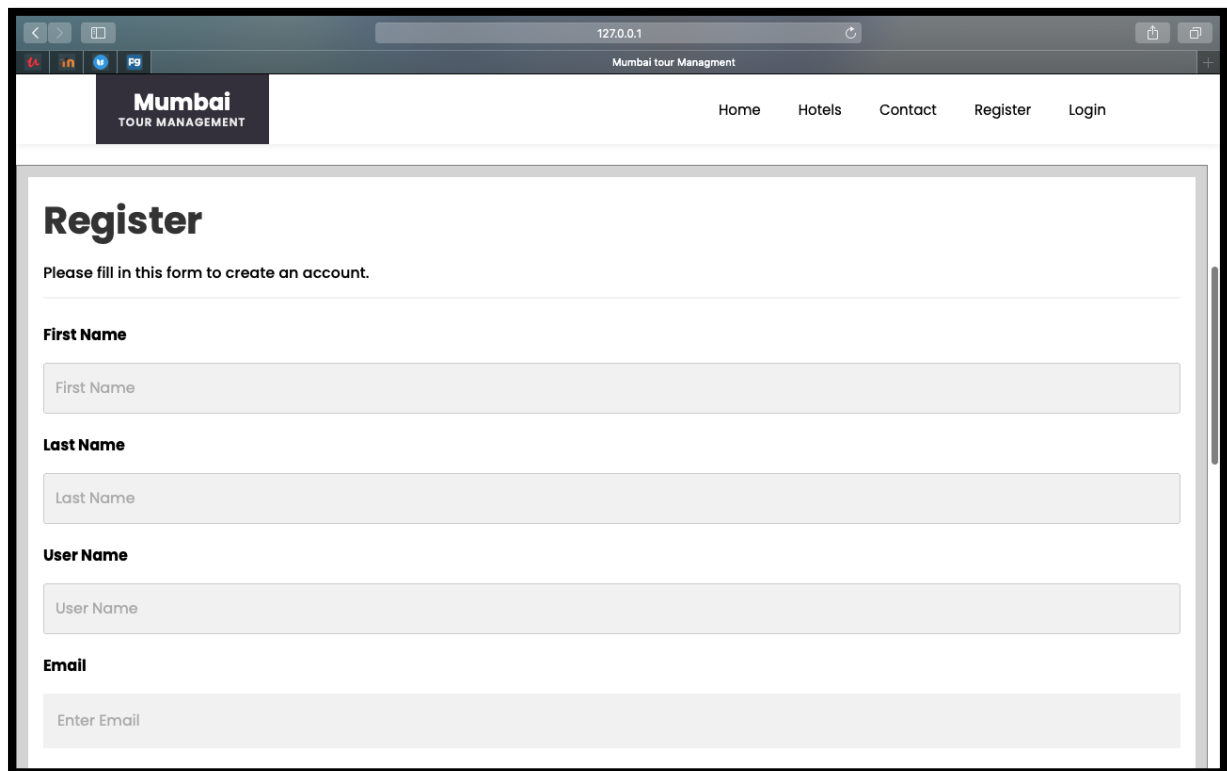


For booking hotel user need to log in system .If user not register in the system then he/she need to register in the application .





For registration user need to fill the details like First name, Last name User Name, Email as well as Password etc.



A screenshot of a web browser displaying the 'Mumbai TOUR MANAGEMENT' website. The browser's address bar shows '127.0.0.1'. The website has a navigation bar with links: Home, Hotels, Contact, Register, and Login. The 'Register' page is active, showing a heading 'Register' and a subtext 'Please fill in this form to create an account.' Below this, there are four input fields: 'First Name', 'Last Name', 'User Name', and 'Email'. Each field has a placeholder text indicating what to enter.

**Mumbai**  
TOUR MANAGEMENT

Home Hotels Contact Register Login

## Register

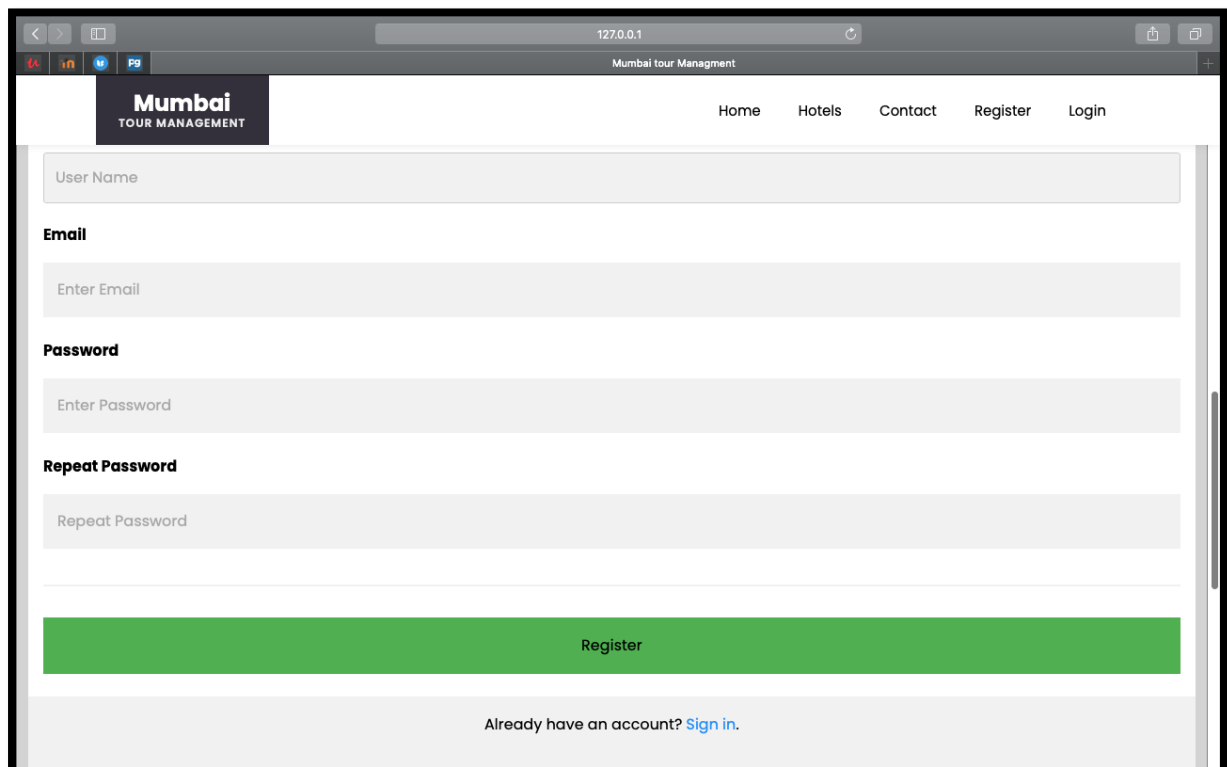
Please fill in this form to create an account.

**First Name**

**Last Name**

**User Name**

**Email**



A screenshot of the same 'Mumbai TOUR MANAGEMENT' website, showing the bottom part of the registration form. It includes the 'User Name' and 'Email' fields from the previous screenshot, followed by 'Password' and 'Repeat Password' fields. Below these fields is a large green 'Register' button. At the bottom of the page, there is a link that says 'Already have an account? Sign in.'.

**Mumbai**  
TOUR MANAGEMENT

Home Hotels Contact Register Login

**Email**

**Password**

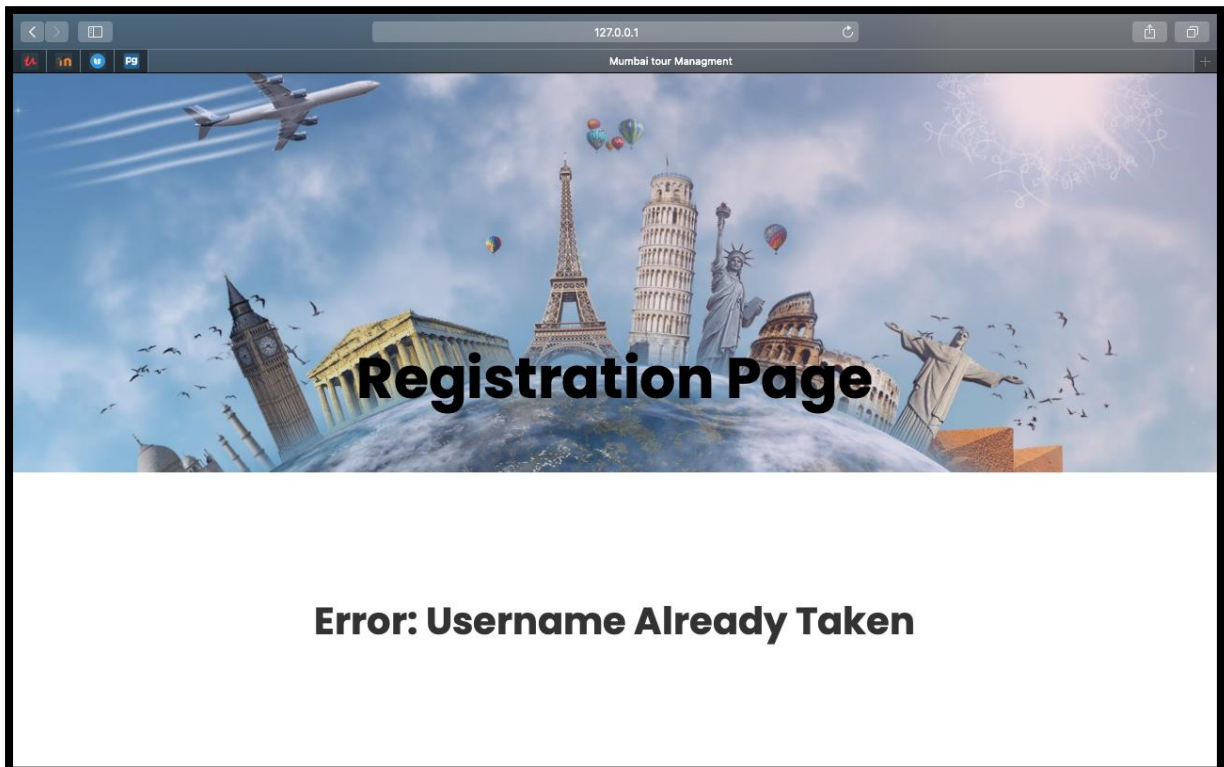
**Repeat Password**

**Register**

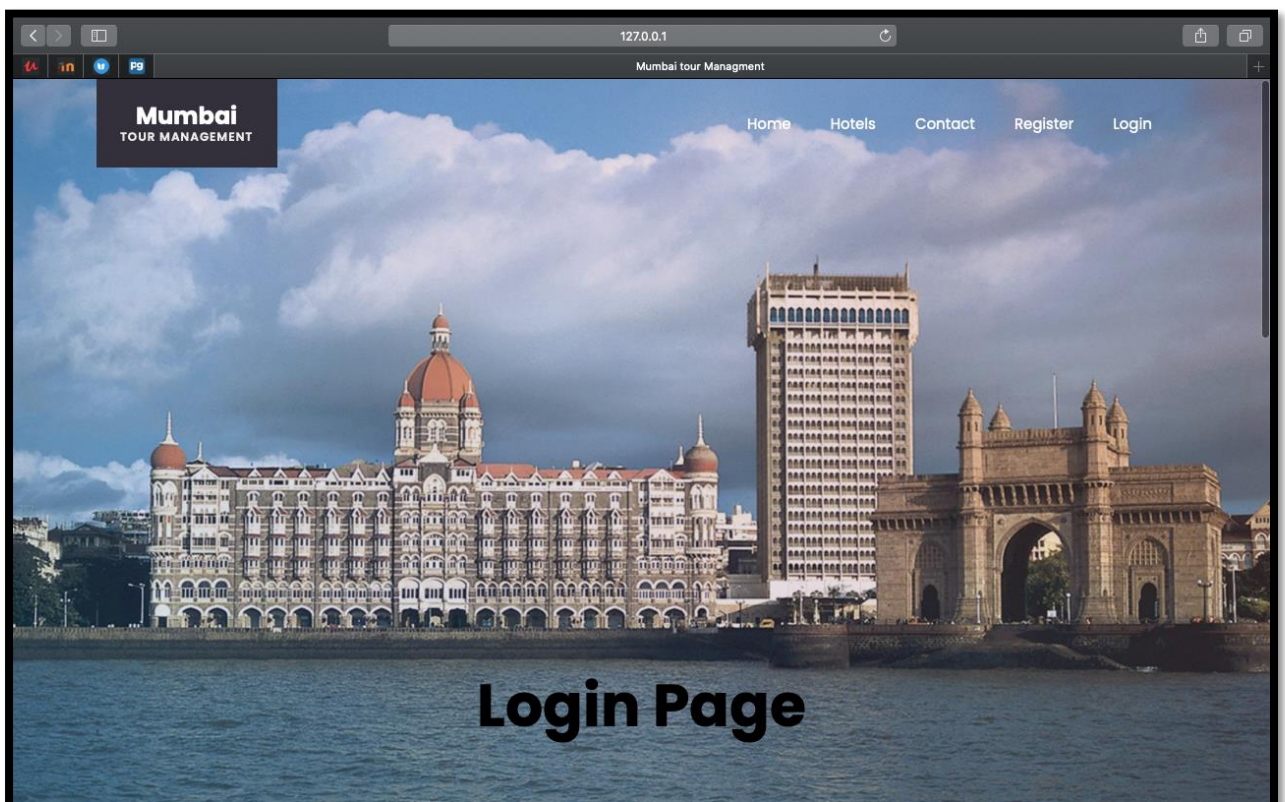
Already have an account? [Sign in.](#)

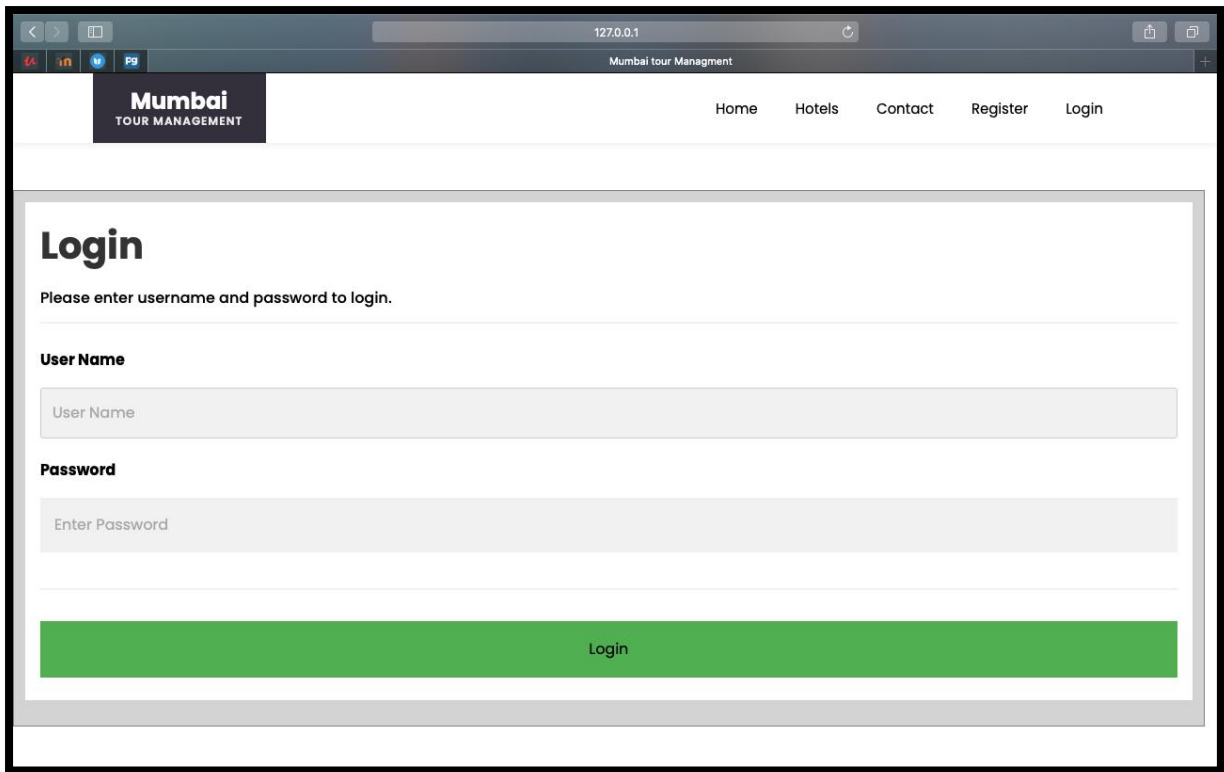
If username is same in database or user can't add already register username.

If user add register username then error message prompt on screen that is Username already taken



Once user register in system after that he/she can log in system for booking hotel.





The screenshot shows a web browser window with the address bar displaying "127.0.0.1" and the page title "Mumbai tour Managment". The website has a dark header with the "Mumbai TOUR MANAGEMENT" logo on the left and navigation links "Home", "Hotels", "Contact", "Register", and "Login" on the right. The main content area is titled "Login" and contains the instruction "Please enter username and password to login." Below this are two input fields: "User Name" and "Password". A green "Login" button is positioned at the bottom of the form.

Mumbai  
TOUR MANAGEMENT

Home Hotels Contact Register Login

## Login

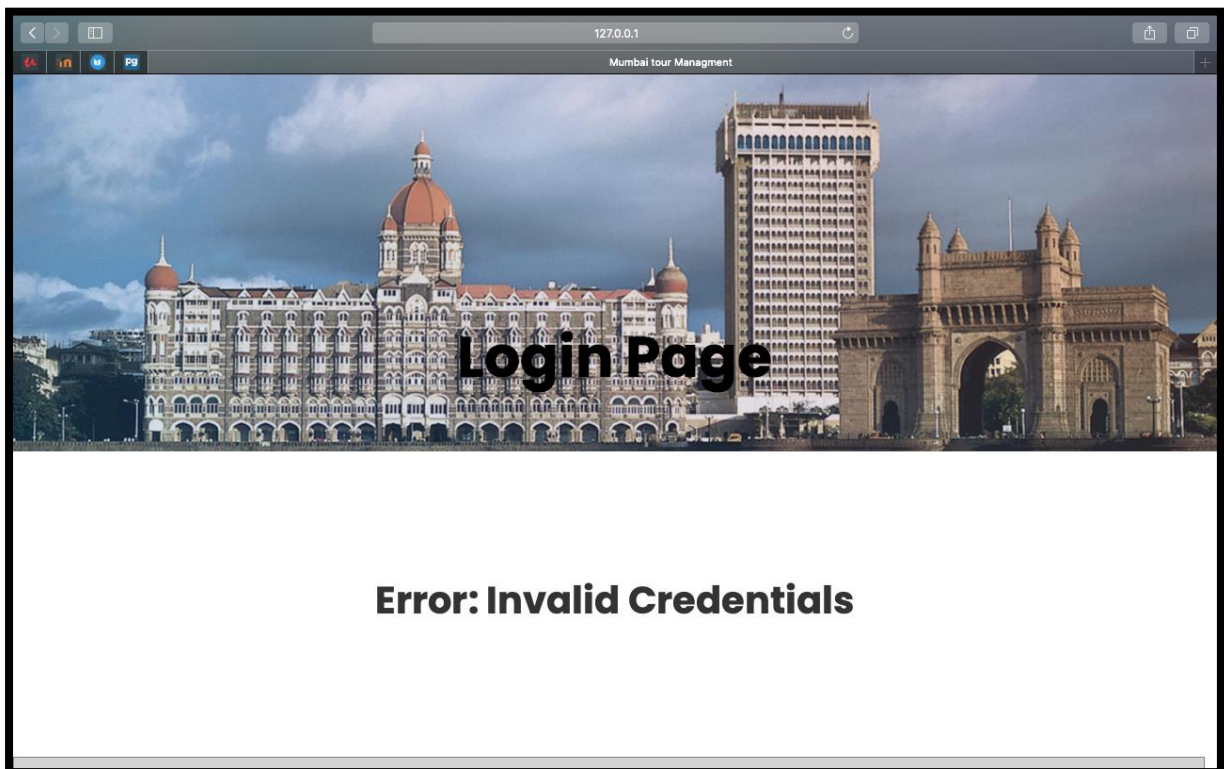
Please enter username and password to login.

**User Name**

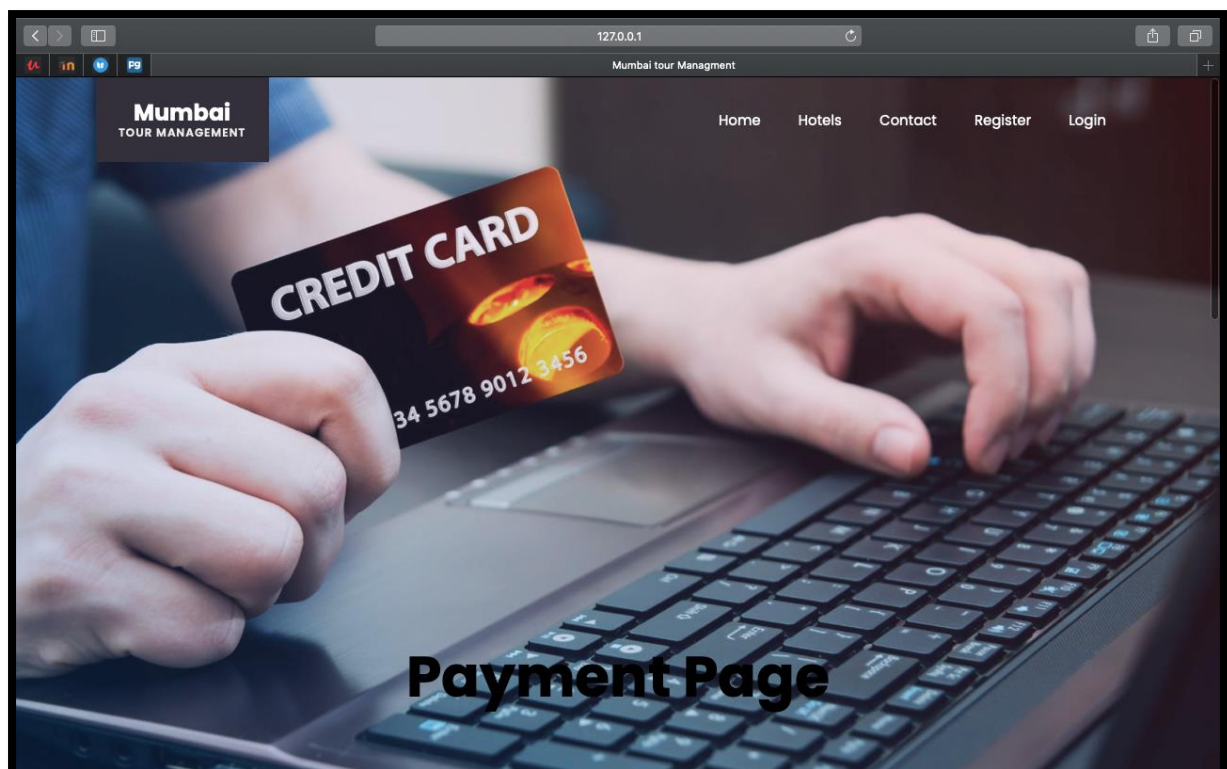
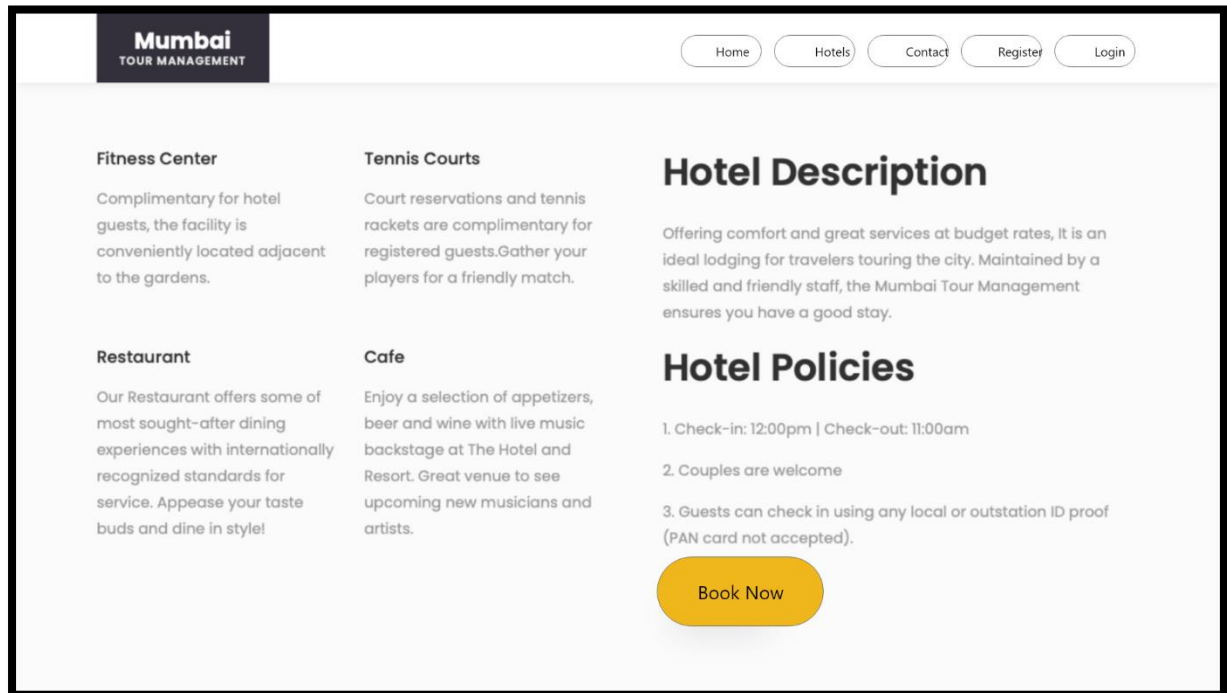
**Password**

Login

If username and password wrong then error message will display that is Invalid credentials



Click on Book Now Button for booking hotel.





After that user need to add details about payment like city, state, zip, Name on card , Credit card number, Exp month, Exp year and CVV etc.

When user click on make payment button if all required details are correct then payment will be successful.

The screenshot shows a web browser window with the URL 127.0.0.1. The website is titled "Mumbai TOUR MANAGEMENT" and has a navigation bar with links: Home, Hotels, Contact, Register, and Login. The main content area is divided into two columns: "Billing Address" and "Payment".

**Billing Address:**

- Full Name: XYZ
- Email: xyz@example.com
- City: Mumbai
- State: MH
- Zip: 10001

**Payment:**

- Name on Card: XYZ
- Credit card number: 1111-2222-3333-4444
- Exp Month: September
- Exp Year: 2028
- CVV: 352

A green "Make Payment" button is located at the bottom of the form.

## Contact Info

User can call to Hotel to ask some info regarding hotel.

The screenshot shows the footer of the Mumbai Tour Management website. It is divided into four columns: "Mumbai Tour Managment", "Information", "Experience", and "Have a Questions?".

**Mumbai Tour Managment:**

Mumbai Tour Managment allows the user to access all details to location, history and reviews.

**Information:**

- Online Enquiry
- General Enquiries
- Booking Conditions
- Privacy and Policy
- Call Us

**Experience:**

- Adventure
- Hotel and Restaurant
- Beach
- Nature
- Camping

**Have a Questions?:**

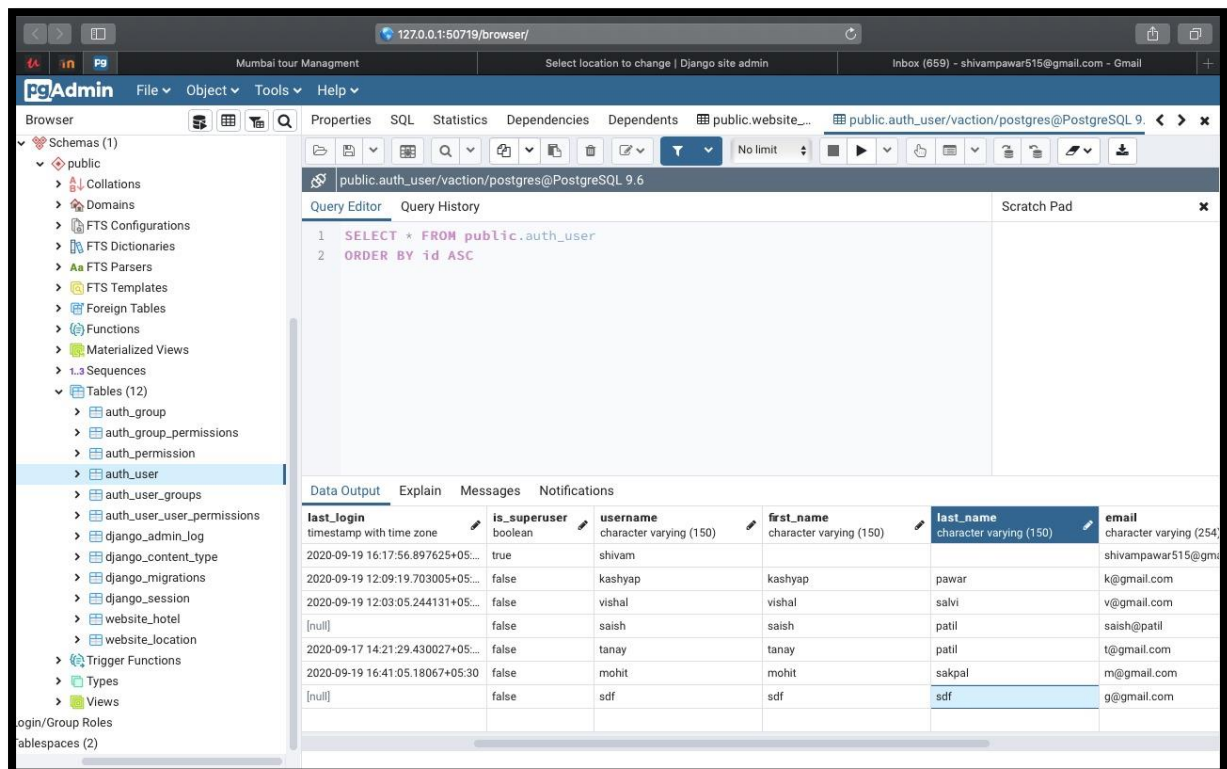
- 203 Fake St. Mountain View, Andheri West, Mumbai, India
- +91 8329502087
- info@gmail.com

Social media icons for Twitter, Facebook, and Instagram are shown. The footer also includes the text "Copyright ©2020 All rights reserved by Shivam".

## Database:

We are using PostgreSQL Database , below screen shot show the authentication user of the system that means the following user are register user in system.

Shivam is supervise user in system where as other are not supervise user so they indicate supervise user false while shivam has true.



The screenshot displays the pgAdmin interface for a PostgreSQL database. The left sidebar shows the 'public' schema with various tables. The main pane shows the 'auth\_user' table with columns: last\_login, is\_superuser, username, first\_name, last\_name, and email. The data output shows a list of users, including Shivam, Kashyap, Vishal, Saish, Tanay, Mohit, and Sdf.

last_login	is_superuser	username	first_name	last_name	email
2020-09-19 16:17:56.897625+05:...	true	shivam			shivampawar515@gmail.com
2020-09-19 12:09:19.703005+05:...	false	kashyap	kashyap	pawar	k@gmail.com
2020-09-19 12:03:05.244131+05:...	false	vishal	vishal	salvi	v@gmail.com
[null]	false	saish	saish	patil	saish@patil
2020-09-17 14:21:29.430027+05:...	false	tanay	tanay	patil	t@gmail.com
2020-09-19 16:41:05.18067+05:30	false	mohit	mohit	sakpal	m@gmail.com
[null]	false	sdf	sdf	sdf	g@gmail.com

**Conclusion:** Thus, we implement web application by using Django Framework also learn the authentication and database connectivity by using PostgreSQL database in Django framework.