CS575 (**Saturday** and **03.15 pm** of the Class)

CAPSTONE COURSE

San Francisco Bay University, Fremont, CA

FINAL REPORT

Term Fall 2022

Prof. Ahmed Banafa

Housepedia – Real Estate Living Simplified Website

Submission Date: 12/17/2022

Team Members

Shoumya Singh	**19566**
Gayatri Kolekar	**19615**

Table of Contents

Abstract 3
1 Introduction 4
2 Full Stack Web Development Techniques 4
2.1 Back-End 5
2.2 Front-End 5
2.3 Database 5
2.4 Deployment 5
3 Housepedia Web Django Architecture 6
4 Housepedia Django Folder Structure 7
5 Functionality Specifications 10
6 Screenshots of the Housepedia Website 11
6.1 Home page 11
6.2 About page 12
6.3 Listing page 13
6.4 Single listing page 14
6.5 Search Page 14
6.6 Register, Login & Dashboard Page 15
6.7 The User makes an Inquiry 16
6.8 The Realtor receives an Inquiry Email 17
6.9 Google Map in Single listing page. 18
7 Django Admin Area 19
8 Database Management 23
9 Deployment 23
9.1 Digital Ocean 24
9.2 Gunicorn & Nginx 26
10 Coding Snipnets 26
Conclusion 30
Table of Figures 31
References 32

Abstract

Over the last few years, the real estate industry has shifted online. The majority of homebuyers look for houses on real estate websites. As a user, they look for homes with locations near their workplace, near the school, near the shopping mall, etc. They filter as per their requirements, like location, how many rooms, and price range.

Our goal for the project Housepedia real estate website is to make users' lives hassle-free by making them search for the property per their requirements. The User can contact the property realtors quickly with a button click. To design the website, we used the Django Python framework with Bootstrap. It allows the User to log in and view or get details of various properties available, allows Admin to add properties, allows Admin to check the availability of properties, and the customer can view all properties online. The functionality specification like Customer inquiry email to the realtor, Google map location of the listed property, and Search of the listed property with keywords like pool, city, and number of bedrooms and bathrooms are achieved. It had been deployed and can be accessed from anywhere.

1 Introduction

Housepedia is a digital real estate platform where realtors, agents, and home buyers can connect to make a real estate purchase. Nowadays, 93% of home buyers in the United States use real estate websites to search for a home. Only 7% said they did not use online websites while looking for a home. As a result, nearly the entire sector went online. Real estate is undergoing significant change, and technology is the primary motivation for this change. For several years, the US real estate market has been entirely under the control of realtors. The goal of the Housepedia real estate platforms is to efficiently solve a user's problem: finding a new home quickly and efficiently. To accomplish this, an app should

- Provide a sufficient number of property options for potential buyers
- Allow users to find the best matching property.
- Make it possible to contact a real estate agent.
- Allow for efficient real estate research.

2 Full Stack Web Development Techniques

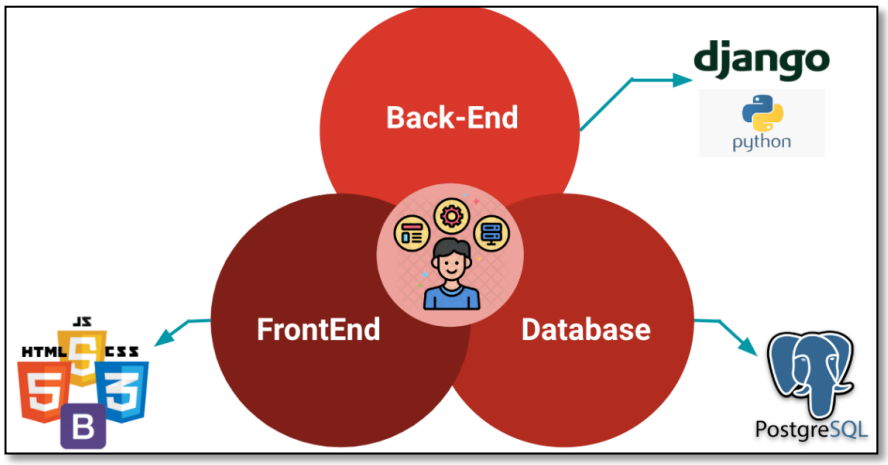

Figure 1:Full stack techniques used

Full stack web technology describes the entire depth of a computer system application, consisting of the front end, back end, and web development database. Everything that a client or site viewer can see and interact with is on the front end. On the other hand, the back end refers to all the servers, databases, and other internal architecture that power the application; the end user rarely interacts directly with this domain.

2.1 Back-End

This project's backend is being built with the Python and Django frameworks. In contrast, the Django framework is a Python web framework that provides for the rapid development of secure and maintainable websites.

2.2 Front-End

The Frontend is developed using Bootstrap. Bootstrap is a free, open-source CSS framework for frontend web development that is responsive and mobile-first. It includes design templates for typography, forms, buttons, navigation, and other interface components in HTML, CSS, and JavaScript.

2.3 Database

PostgreSQL is the database in this project, and it is managed with the pgAdmin management tool. PostgreSQL is a robust, free, open-source object-relational database that utilizes and extends the SQL language and includes numerous features for storing and scaling the most complex data workloads.

2.4 Deployment

This project will use Digital Ocean with Gunicorn & Nginx for deployment. DigitalOcean is a cloud hosting company that provides cloud computing and Infrastructure as a Service (IaaS). The

Web Server Gateway Interface (WSGI) is a standard interface between web server software and web applications that Gunicorn implements. Nginx is a type of web server. It is the public handler, also known as the reverse proxy, for incoming requests and can handle thousands of concurrent connections.

3 Housepedia Web Django Architecture

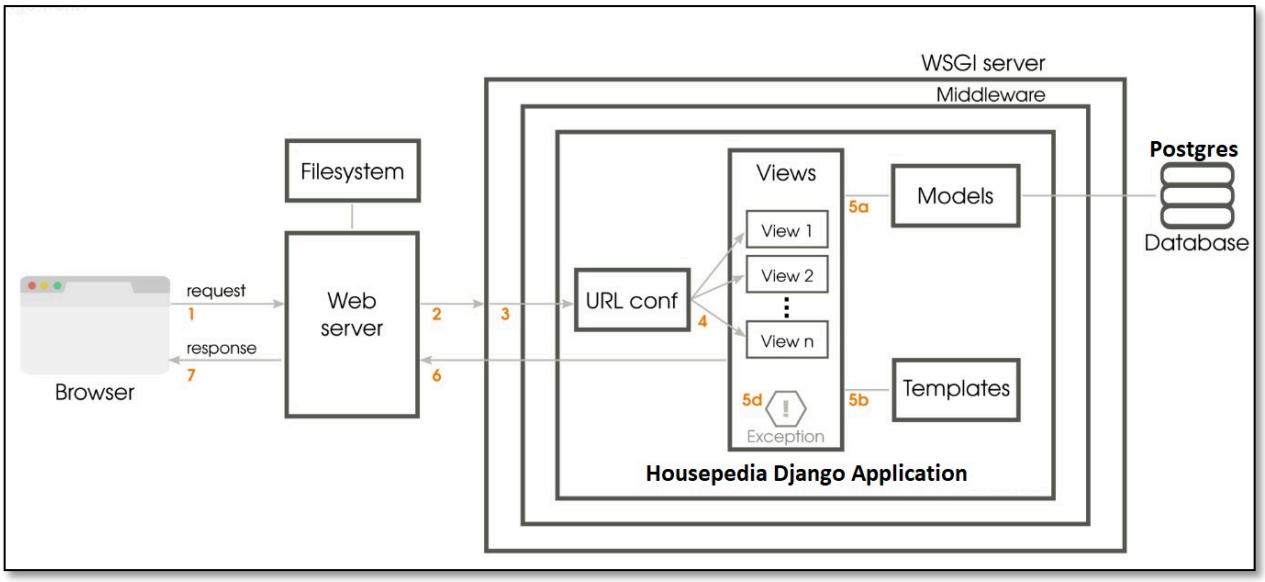

Figure 2 Housepedia Django Architecture

The diagram above depicts the simplified path of a web request from a user's browser to a Django application and back. The following are the numbered paths:

1. The browser sends the request (a string of bytes) to its web server.
2. The web server (for example, Nginx) routes the request to a Web Server Gateway Interface (WSGI) server (for example, uWSGI) or serves a file (for example, a CSS file) directly from the filesystem.

3. WSGI servers, unlike web servers, can run Python applications. The request contains a Python dictionary called environ and, if desired, passes through several layers of middleware before arriving at the Django application.
4. The URLconf (URL configuration) module in Housepedia urls.py selects a view to handle the request based on the requested URL. The request has been converted into a Python object called HTTP Request.
5. The chosen view usually does one or more of the following:
    1. The models communicate with a database.
    2. Using templates, it generates HTML or any other formatted response.
    3. Returns a plain text response (which is not displayed).
    4. An exception is raised.
6. The HTTP Response object is rendered into a string as the Django application exits.
7. The User's browser displays a beautifully rendered web page.

4 Housepedia Django Folder Structure

This folder structure is from the Visual studio code.

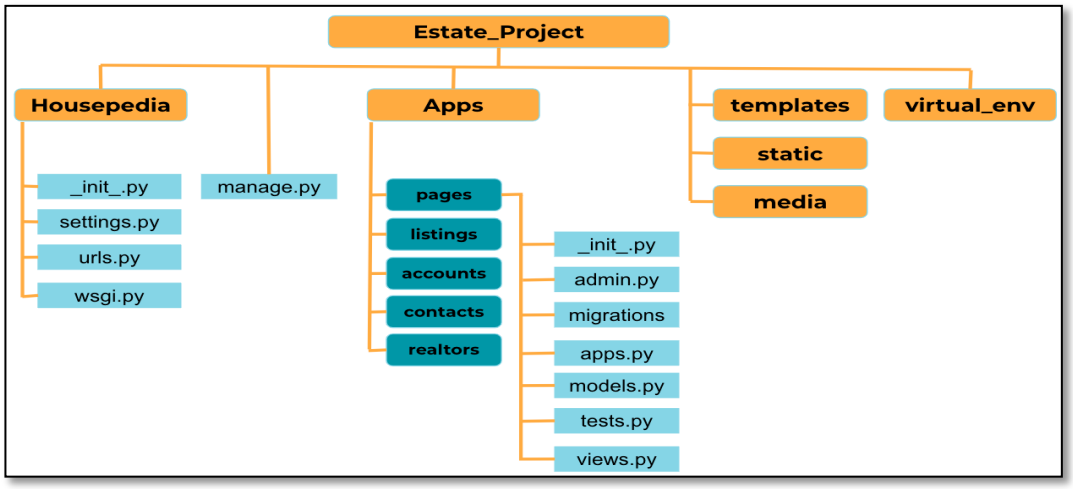

Figure 3 Folder Structure

When we create a Django project, Django creates a root directory for the project with the project name specified. In our case, it is Estate_Project. It contains the files required to provide basic functionality to our web applications.

Different files in Django Estate_Project Root Directory

- manage.py - For our projects, this file serves as a command-line utility. This file will debug, deploy, and run our website. This python file contains the code for running the server, making migrations or migrations, and many other commands we execute in the code editor.
- Housepedia - Housepedia is the name we gave our project when we ran the Django-admin start project (project name). It contains the project's configuration files.
    - _init_.py – This is an empty file. The goal of this file is to inform the Python programmer that this directory is a package and that the presence of this __init_.py python file makes it a Python project.
    - settings.py – It contains the Django project settings. The most important file is setting.py, which is used to add all of the applications and middleware applications. This is the Django project's main configuration file. It uses SQLite as the default database; we changed it to a PostgreSQL database. It comes with some pre-installed apps and middleware that provide basic functionality.
    - urls.py – A URL is a universal resource locator; it contains all of the endpoints that our website should have. It is used to provide you with the address of resources (images, webpages, websites, and so on) that are available on the internet.
    - wsgi.py – This file will be used during the deployment. WSGI is an abbreviation for Web Server Gateway Interface, which describes how servers interact with applications.

- Apps - Django manages codes using the concept of "projects" and "apps" and provides them in an understandable form. A Django project has one or more apps that work parallelly to provide a smooth web application operation. In our case, Housepedia is the project name, pages is an app for different pages on the website, listings is an app for property listings, and other apps are accounts, contacts, and realtors. After the app's creation, each has a few files generated with different functions.
    - _init_.py – This file provides the same functionality as the Django project structure's _init_.py file. It is an empty file that does not require any changes. It simply indicates that the app directory is a package.
    - admin – The Admin.py file is used to register Django models in the Django administration. It is used in the Django admin panel to display the Django model. It carries out three primary functions: model registration, superuser creation, and logging in and using the web application.
    - migrations – all the migration files are generated here.
    - apps.py - Apps.py is a file that allows the User to include application configuration for their app.
    - models.py - Models.py is a Python module that represents web application models as classes. It is regarded as the most crucial aspect of the App file structure. Models define the database's structure. It describes the overall design, data set relationships, and attribute constraints.
    - tests.py - Users can use Tests.py to write test code for their web applications. It is used to test the app's functionality.

- views.py - When discussing the Django app structure, views play an important role. Views provide an interface for interacting with a Django web application. It contains all of the views as classes.

- Templates – The template folder stores all the HTML pages created, like the homepage, about page, search page, log in, register, and dashboard.
- Static – The static folder contains all the bootstrap-related files, such as CSS, JavaScript, and static files.
- Media – The media folder has all the uploaded images and photos used on the website.
- Virtual Environment – This folder contains virtual environment-related files.

5 Functionality Specifications

The web page will have the following functionality as follows:

- Manage listings, realtors, contact inquiries, and website users via the admin page.
- Role-based users (staff and non-staff)
- Display listings in the app with pagination
- Ability to set listings to unpublished
- Search listings by keyword, city, bedrooms & price (Homepage & search page)
- List realtors on the about page with "seller of the month" (Control via Admin)
- The listing page should have fields listed below
- The listing page should have five images with lightbox
- The lightbox should scroll through images
- The listing page should have a form to submit an inquiry for that property listing
- Form info should go to the database, and notify Realtor (s) with an email
- Frontend register/login to track inquiries

- Google Map with the property location maker in the single listing page.
- Google map with zoom feature.
- Address prediction in the admin page while adding property addresses.
- Both unregistered and registered users can submit the form. If registered, one can only submit one per listing.

6 Screenshots of the Housepedia Website

6.1 Home page

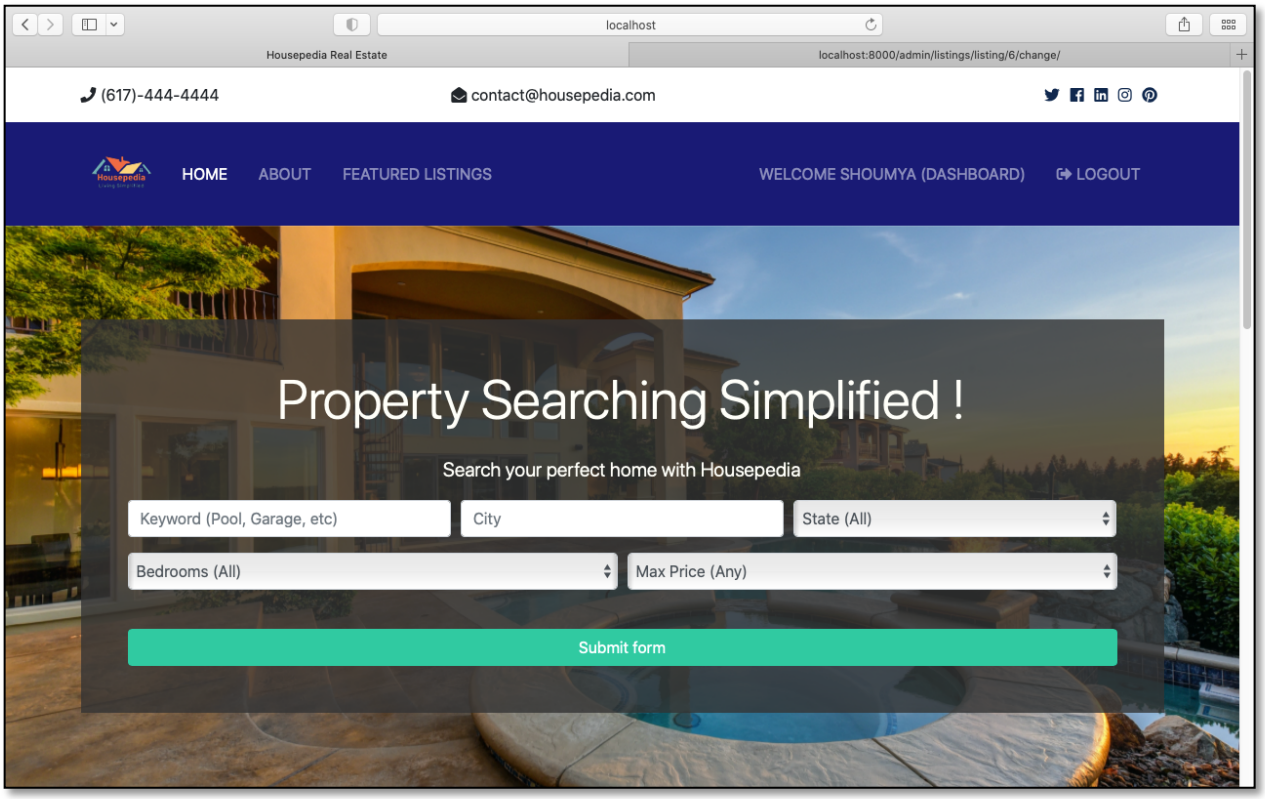

Figure 4 Home Page of Housepedia

The homepage allows us with following functions:

a. Register, Login & Logout

b. View users Dashboard

c. Property Search

d. Navigate to the About page, Featured listing page

As we scroll down on the homepage, it also shows the latest listings with minimal details; for details of the property, the User needs to select the More Info button, which takes the User to the single listing page with detailed information about the listed property.

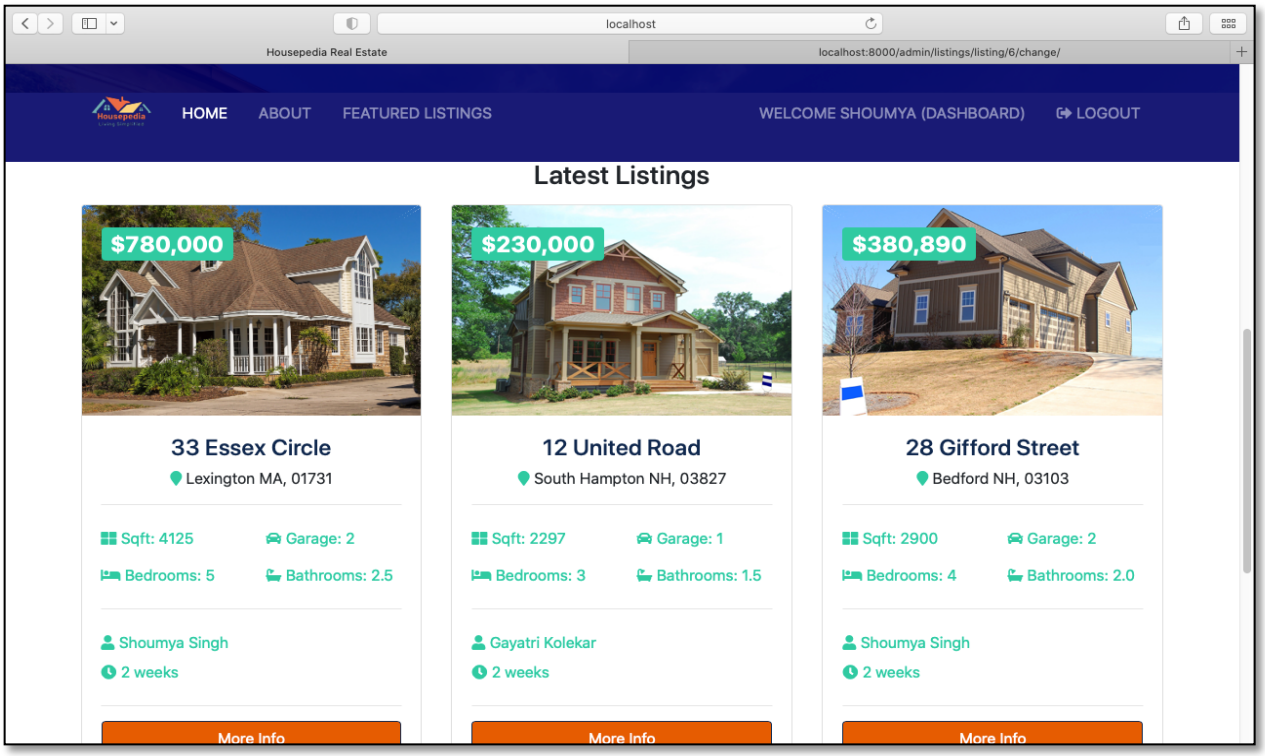

Figure 5 Homepage Continue

6.2 About page

The About page allows us with following functions:

a. Seller of the Month (Realtor)

b. Know about Housepedia

c. View Our Team

d. Navigate to the Home page, Featured listing page

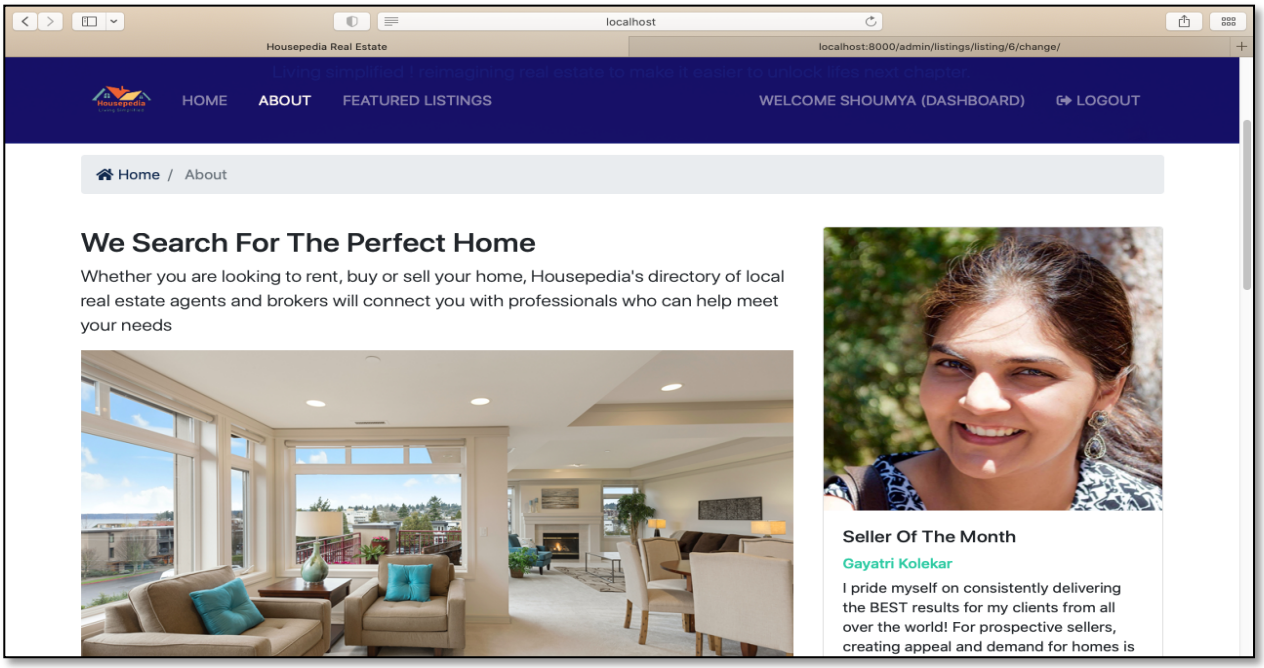

Figure 6 About page

6.3 Listing page

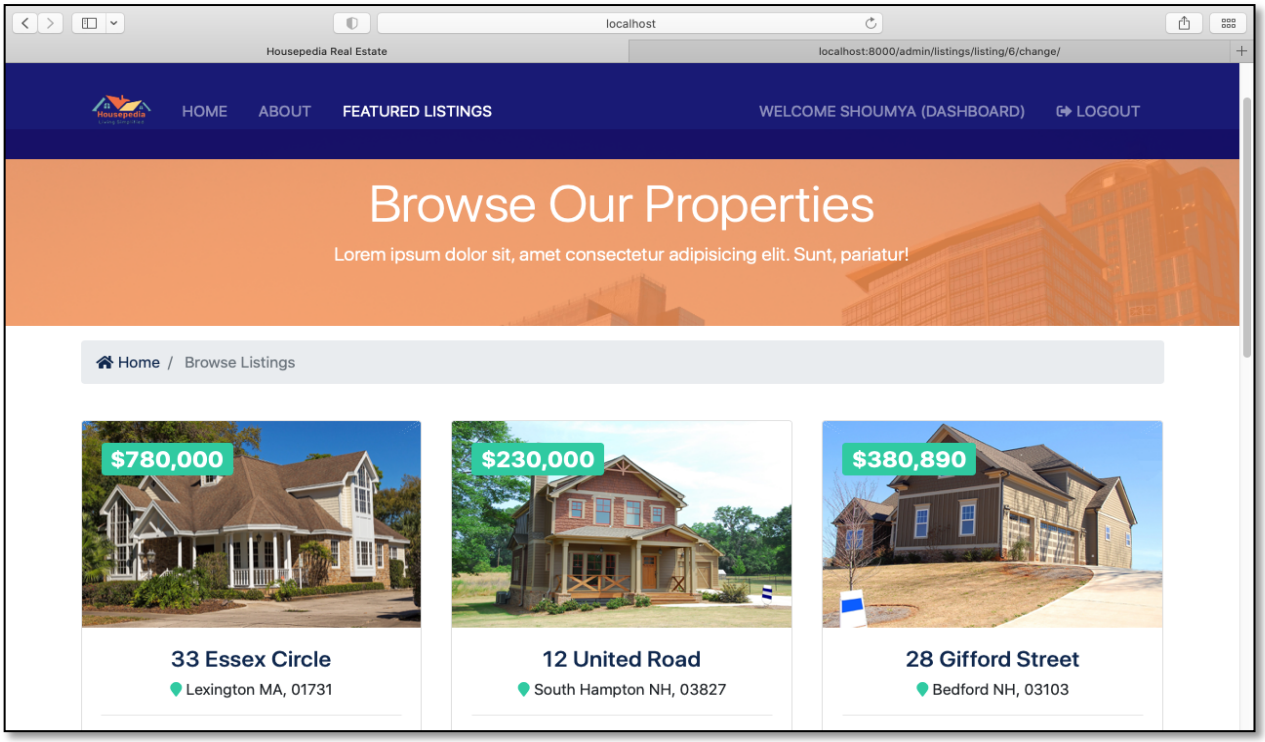

Figure 7 Listing page

The Listing page allows us with following functions:

a. View Property listings

b. Navigate to the single listing

6.4 Single listing page

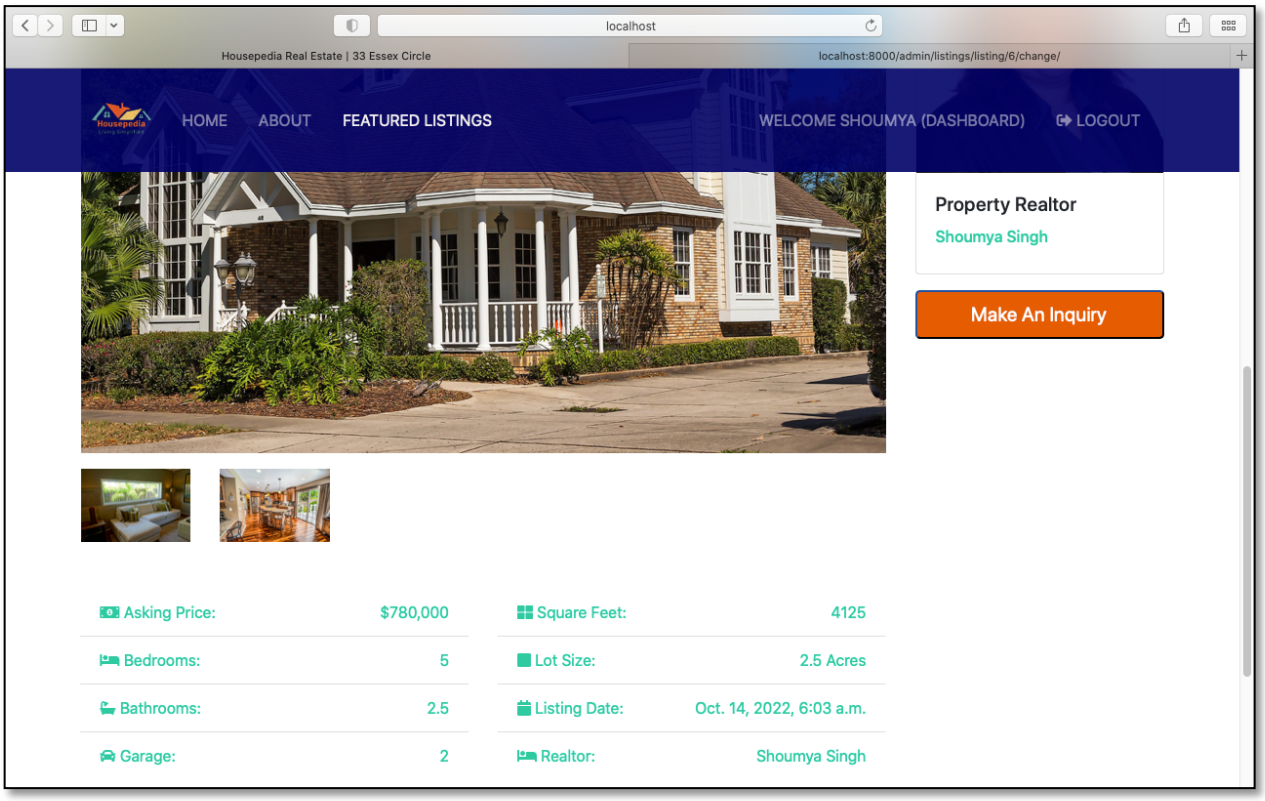

Figure 8 Single listing page

The Listing page allows us with following functions:

a. Detailed view of the Property with Images (the images can be scrolled in lightbox.

b. Make an Inquiry about the property to the assigned property realtor.

6.5 Search Page

The search page allows the User with following:

a. View Searched property

b. Property Search

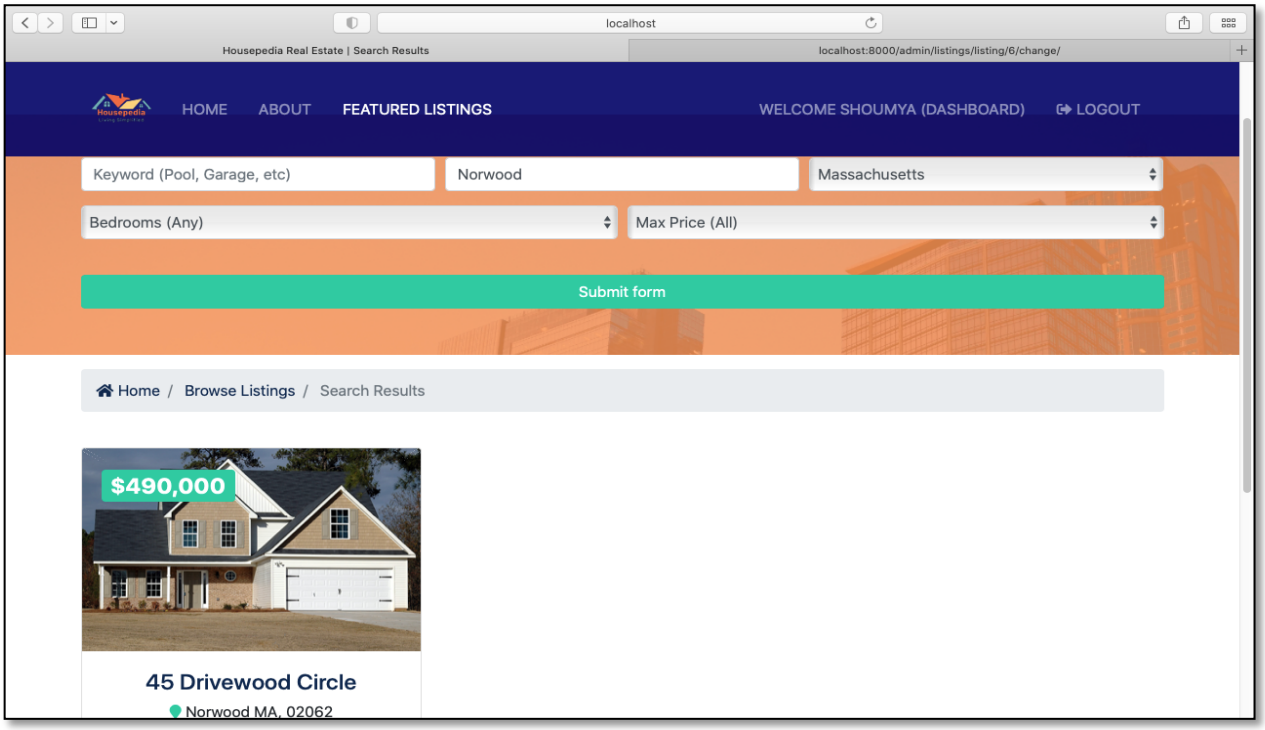

Figure 9 Search page

6.6 Register, Login & Dashboard Page

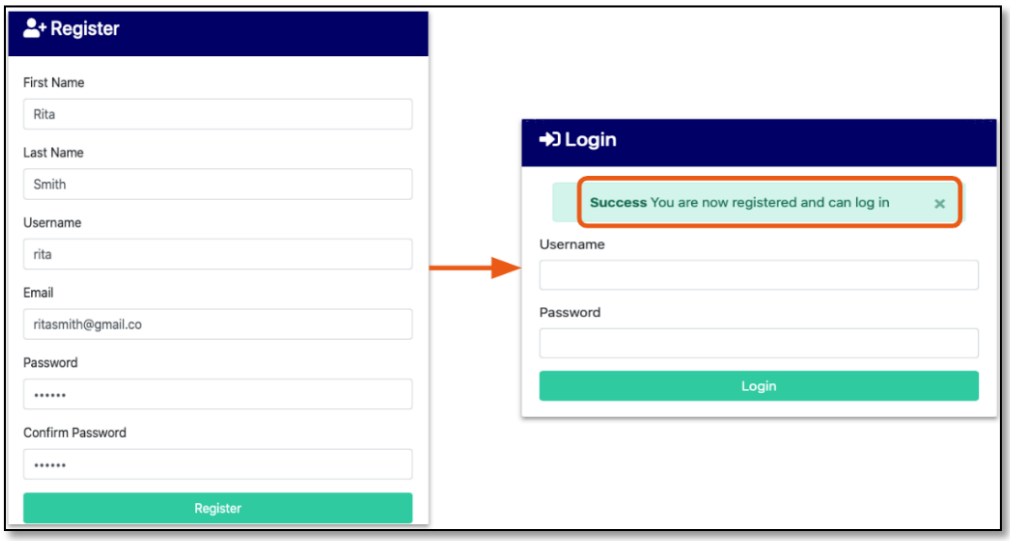

Figure 10 Register & Login

As the User registers from the homepage by filling up a few details like first name, last name, username, email, and password and clicks on the register button, it takes the User to the login. It pops up a message as "Success. You are now registered and can log in. " figure 10.

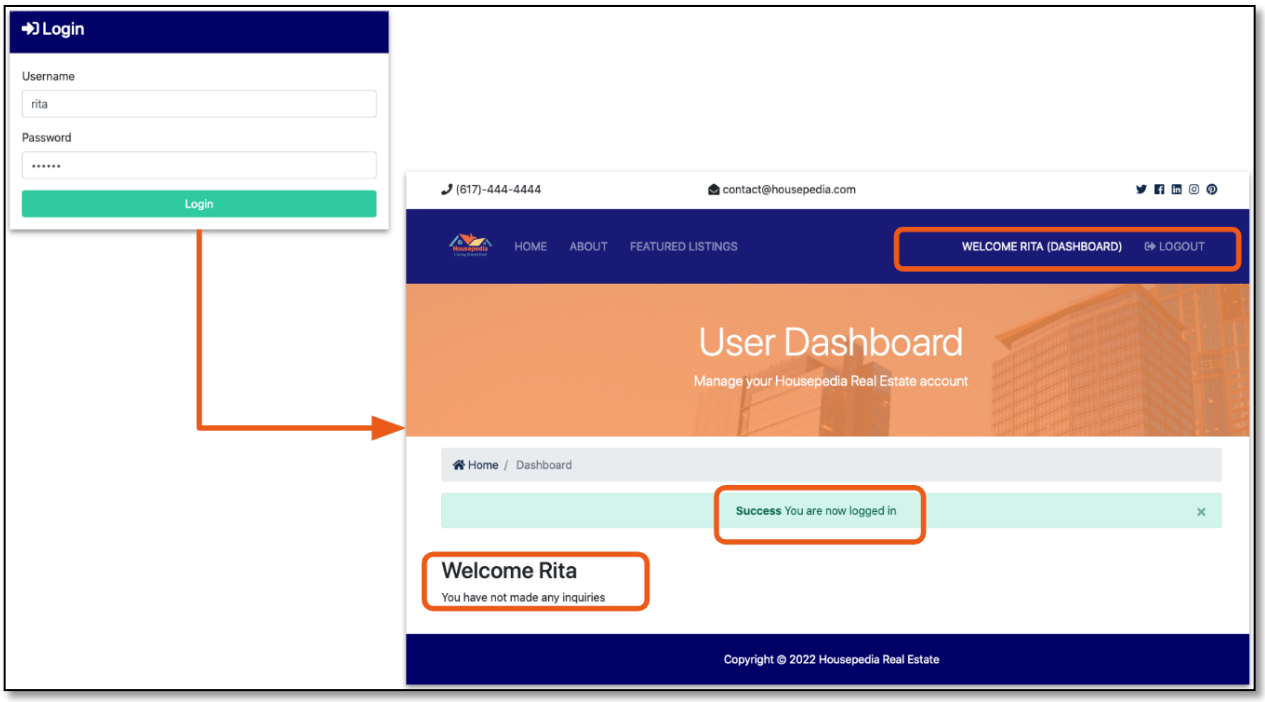

Figure 11 Login & Dashboard

Figure 11 shows that the User logs in and takes the User to the User Dashboard. The User Dashboard welcomes the User with their first name and shows the pop message "Success! You are now logged in. "

6.7 The User makes an Inquiry

Now, if the User likes the property listed on the website, they can inquire by clicking the button provided on the single listing page as "Make an Inquiry." The new dialogue box appears. If the User is already registered, then some of the information will be already filled like property address, name, email, and phone. Only a message has to be written before clicking send button.

The User must fill out the entire form and send the inquiry if not a registered User. It can be seen the Figure 12.

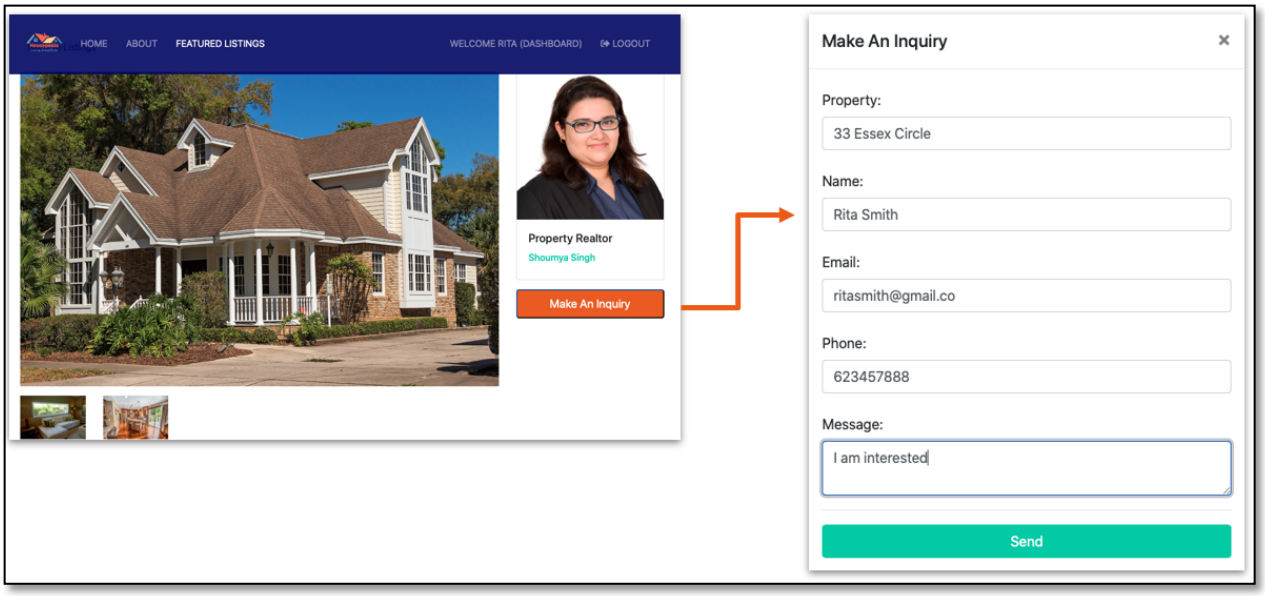

Figure 12 Inquiry dialogue box

6.8 The Realtor receives an Inquiry Email

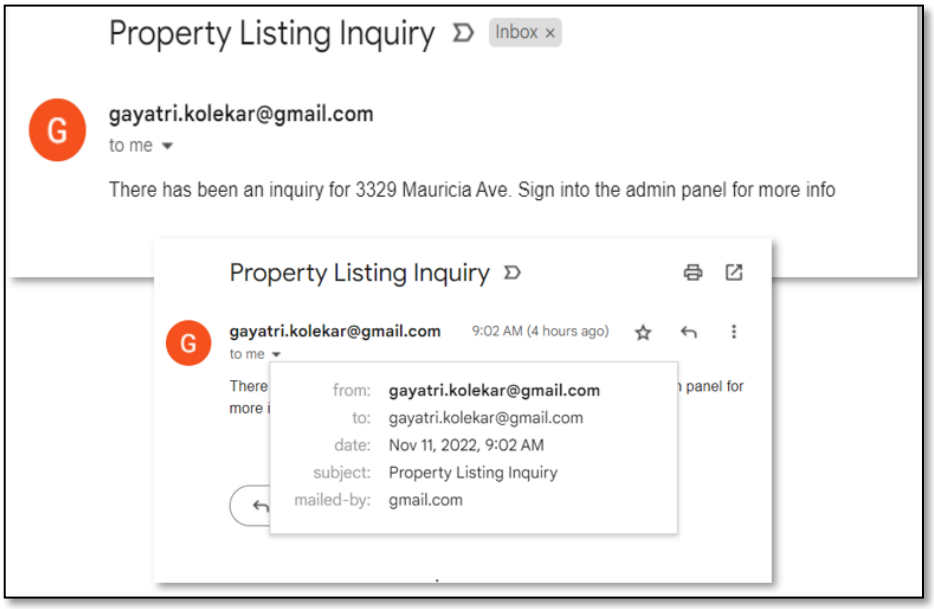

Figure 13 Email to Realtor

The above screenshots Figure 13 shows the Inquiry email received by the Realtor. The message in the email is "There has been an inquiry for (address of the property). Sign in to the admin panel for more info." It means the Realtor has to log in to the admin page to get the customer's details to contact them back. The admin view will be shown in the admin section in the further report.

6.9 Google Map in Single listing page.

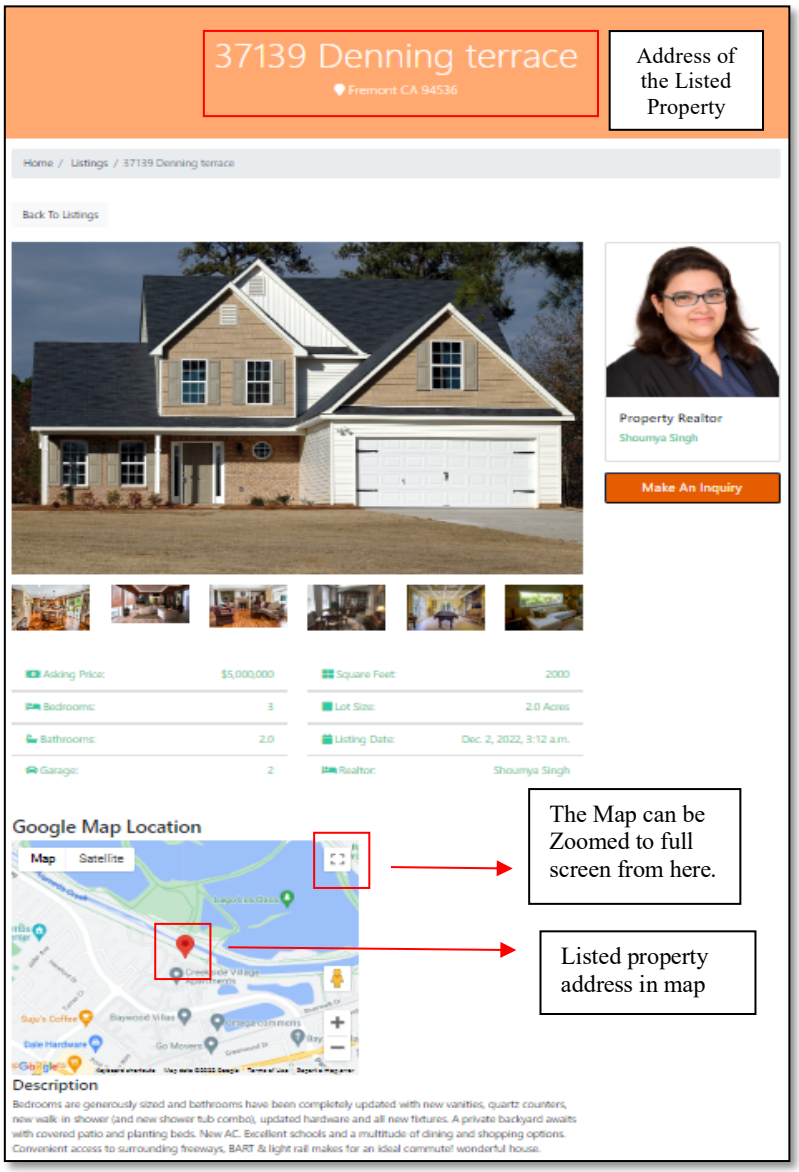

Figure 14 Google Map in Single listing page

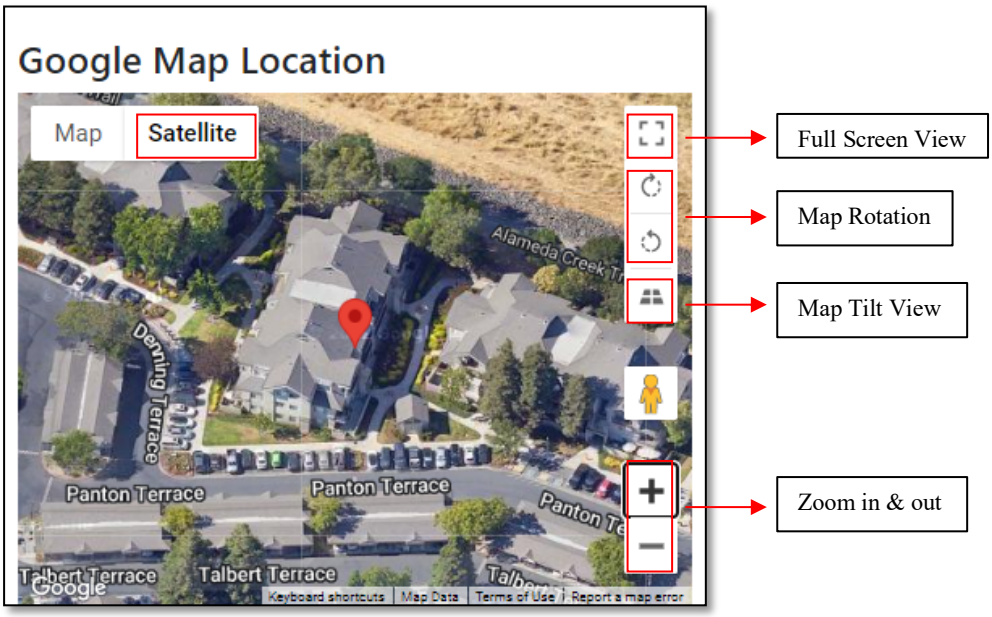

Figure 15 Google Map in Zoomed View

The Google Maps feature is integrated into the single listing page. When the User clicks on the "More info " button on the liked property, it takes the User to the more detailed page, which is the single listing page. This page shows all the details of the property. The google map shows the location of the property. It makes it easier for the User to see the location's neighborhood with the zoom-in and out feature on the map. The map can be zoomed to full screen with a click and navigation. Google map has the feature to rotate and tilt the map . It can be viewed in Satellite view with or without labels. Figure 14 is the screenshot of the single listing page. Figure 15 is the zoomed view of the property location, marked in red.

7 Django Admin Area

The admin area can control the entire web page by user authorization, adding and removing listings, adding and removing realtors, controlling seller of the month, and publishing and unpublishing the listings. Following are the screenshots of the admin areas.

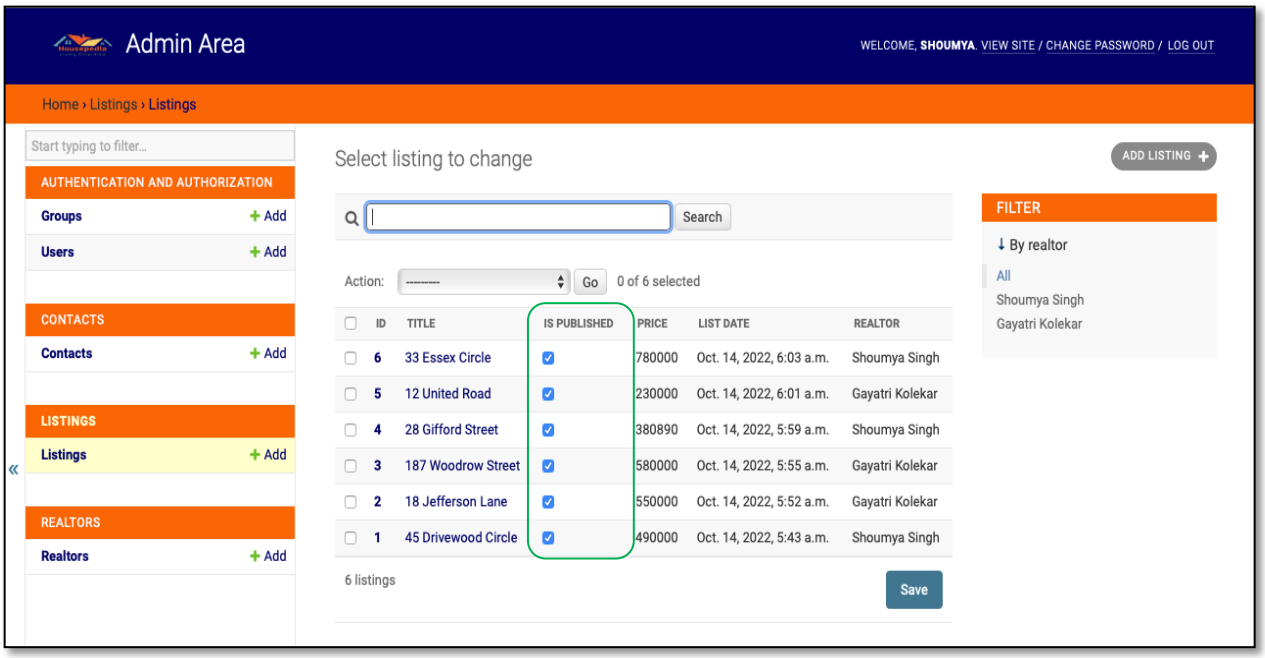

Figure 16 Admin Listing Page

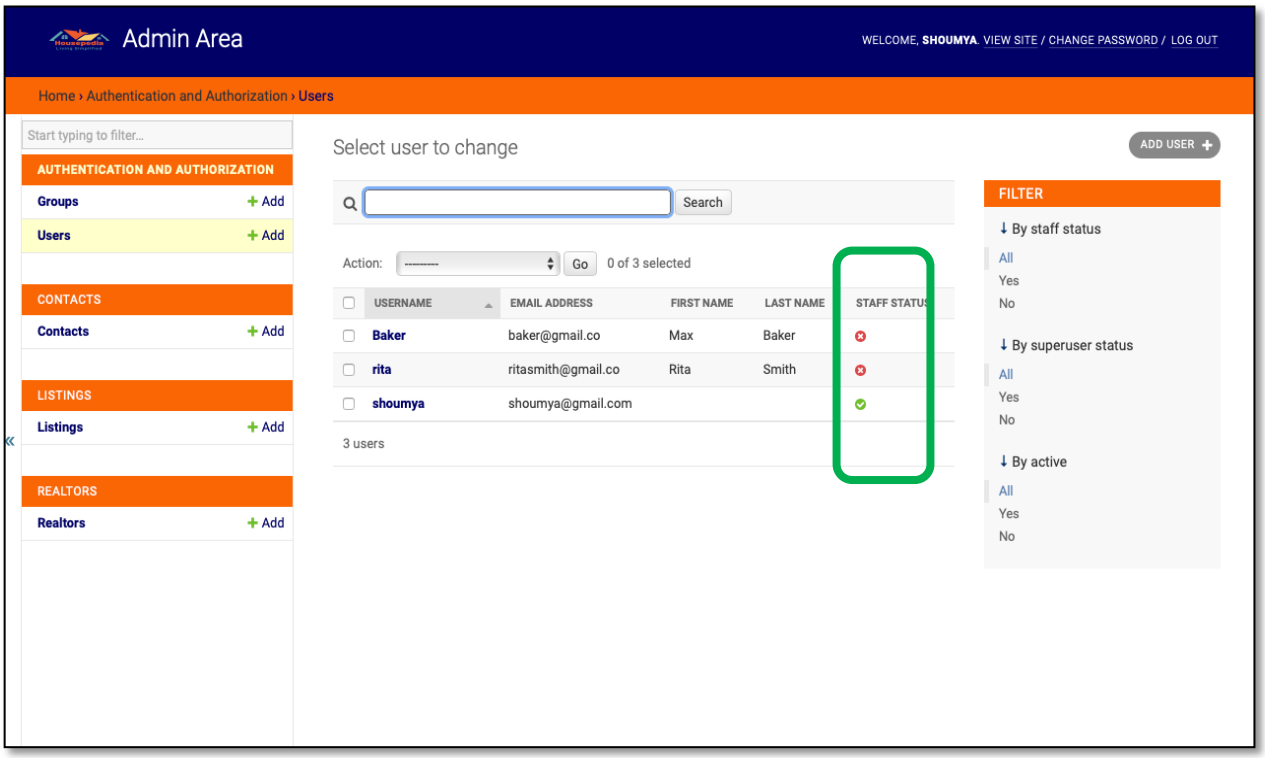

Figure 17 Users in the Admin

Figure 16 shows the properties published on the website, and Figure 17 shows the admin page users. The green marked area indicates staff status, where the green indicates a staff and the red indicates no staff.

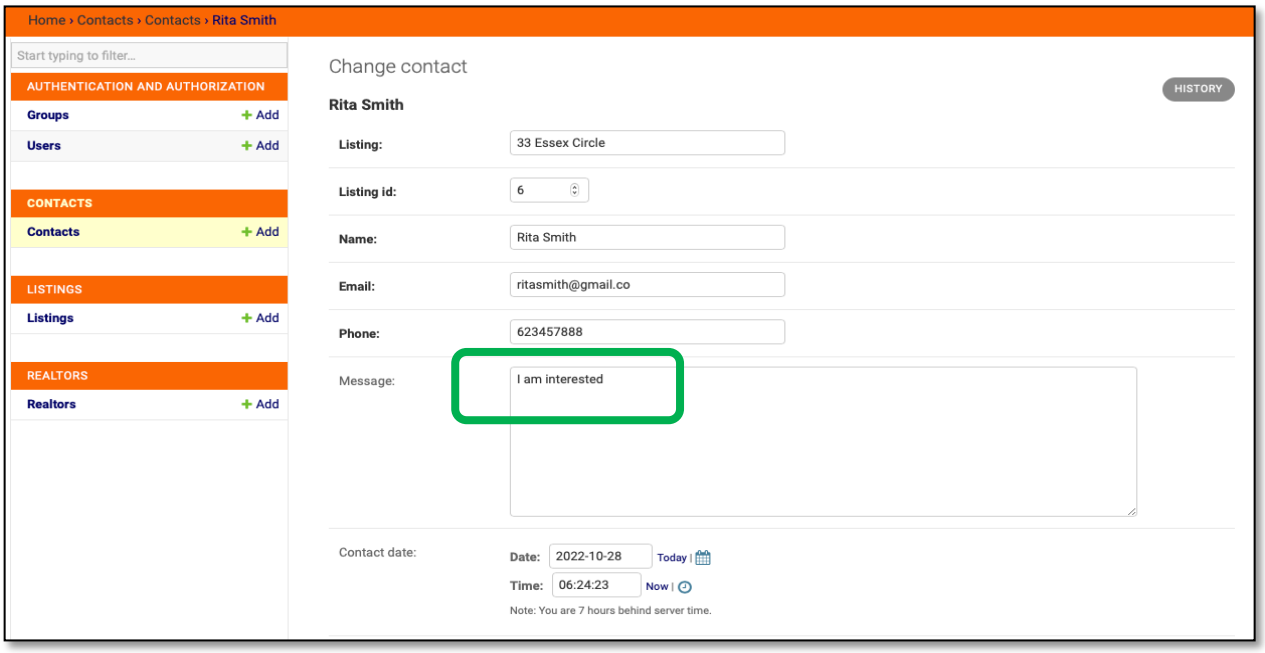

Figure 18 Admin page of received messeage for inquiry

Above Figure 18 shows a message received by the realtor from the User inquiring about the property in the admin section. Figure 19 shows the list of Realtors on the website.

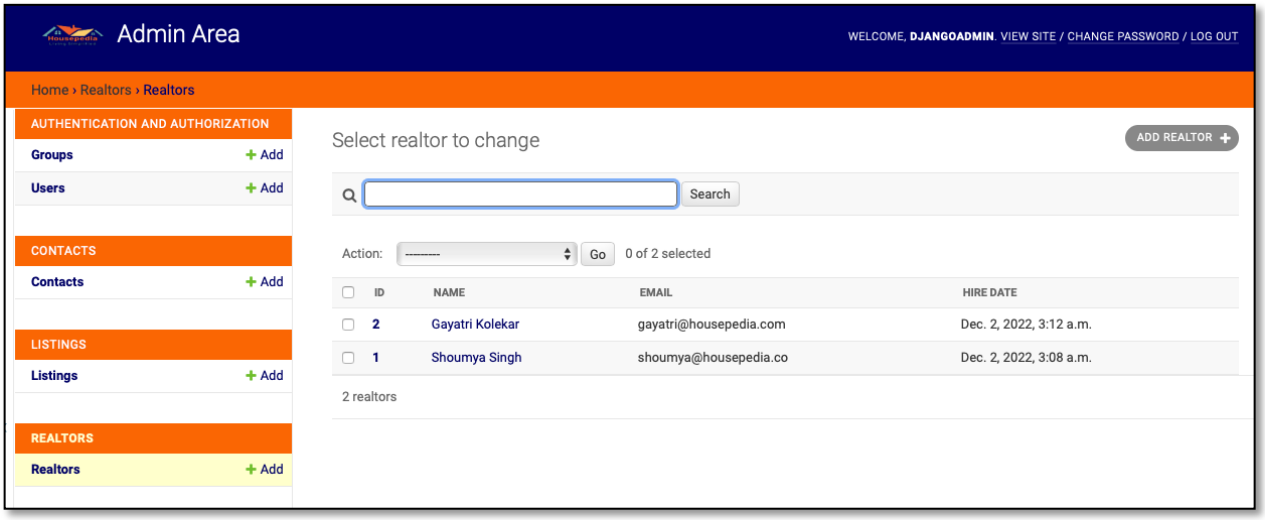

Figure 19 Realtor Admin Page

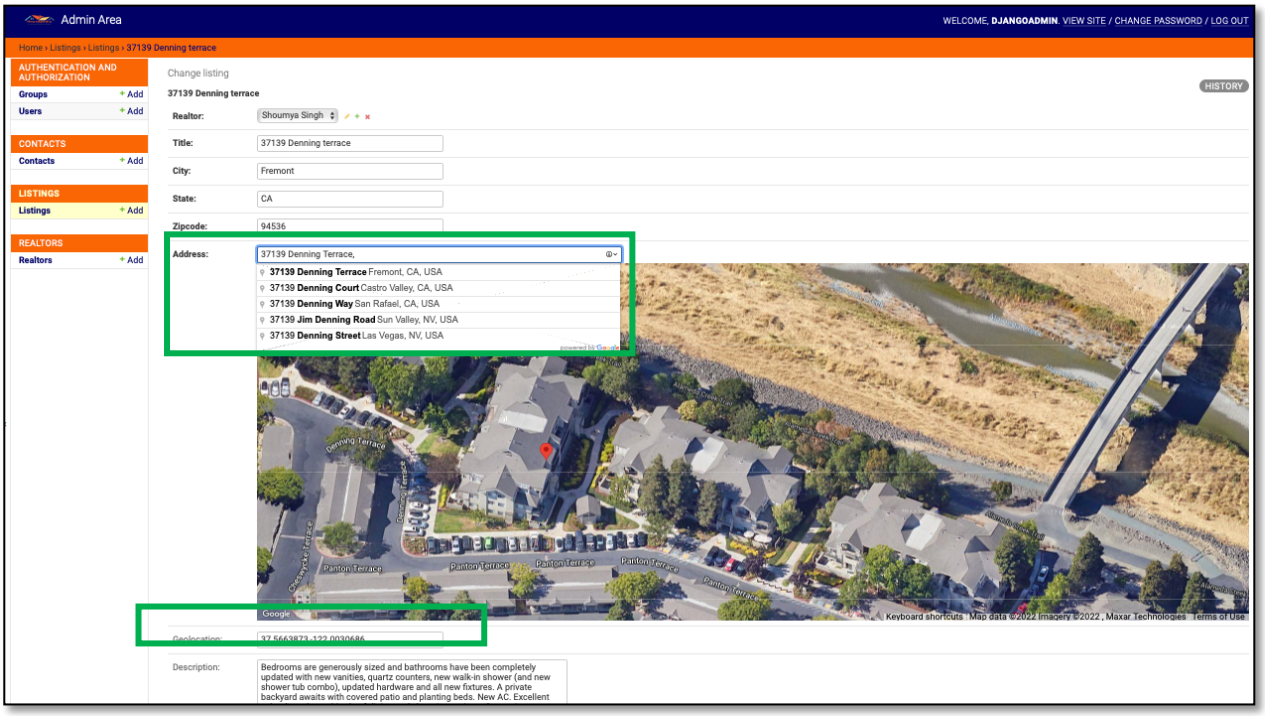

Figure 20 Google Map on Admin side

Figure 20 is the admin area for adding a listing on the website. As we write the address, it shows the predicted addresses in the dropdown box. When we add the complete address, it takes the Geolocation, that is, longitude and latitude, to display the exact location of the listed property. This two feature in the admin section makes it possible to view the google map location on the single listing page of the website.

Google API is used to integrate the google map feature on the Housepedia website. In response to an HTTP request via a URL, the Maps Static API returns an image (GIF, PNG, or JPEG). You can specify the map's location, the size of the image, the zoom levels, the type of map, and the placement of optional markers for each request. You can also use alphanumeric characters to label your markers. A Maps Static API image is embedded within the source src attribute of a <img> tag, or it is similar in other programming languages.

8 Database Management

Our database is PostgreSQL, which we manage with the pgAdmin management tool, as shown in the screenshot below. pgAdmin is a web-based GUI tool for interacting with Postgres database sessions on local and remote servers. pgAdmin can perform any database administration tasks for a Postgres database.

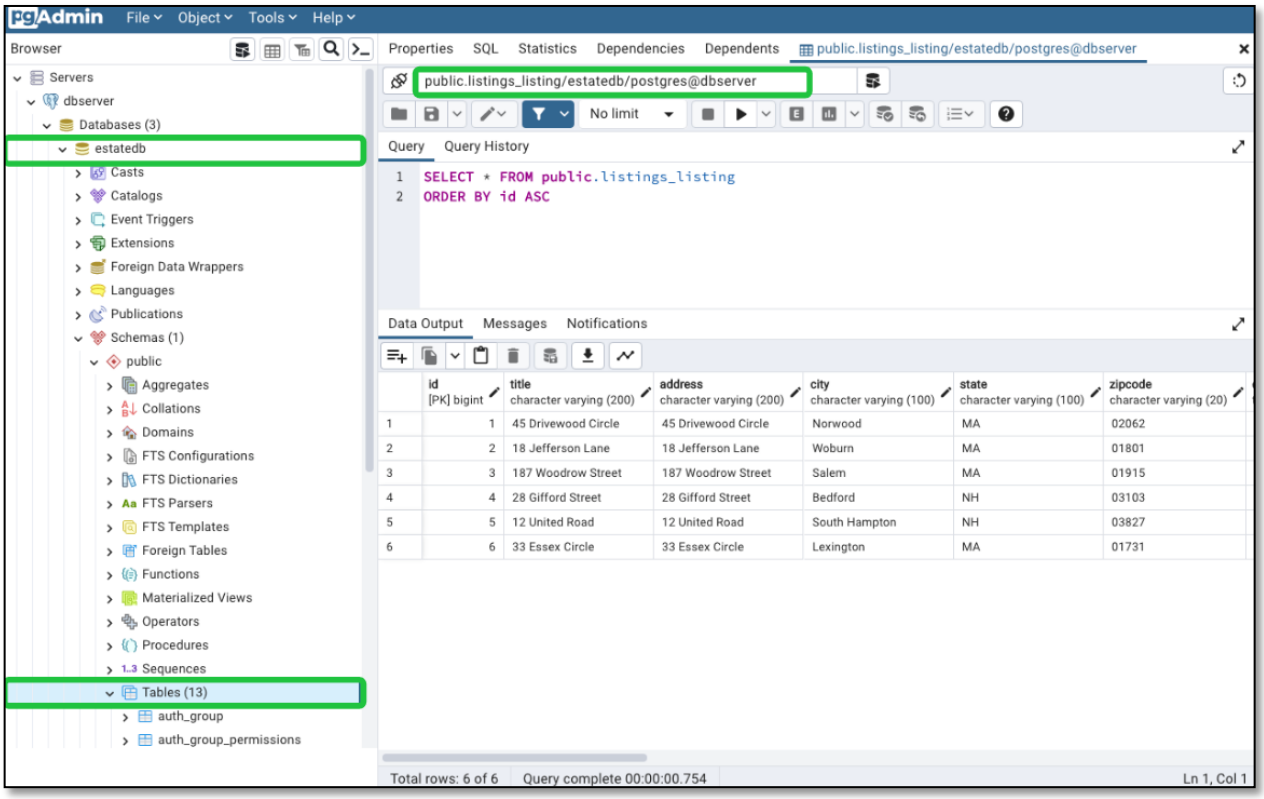

Figure 21 pgAdmin page with our listing table

9 Deployment

Figure 22 Logos

The deployment of the Housepedia is done with the cloud computing provider GitHub and two web servers**. DigitalOcean** is a cloud hosting provider that offers cloud computing services and Infrastructure as a Service (IaaS). **Gunicorn** is a server that runs applications. It converts HTTP requests into Python-friendly syntax. The Web Server Gateway Interface (WSGI) is a standard interface between web server software and web applications that Gunicorn implements. **Nginx** is a type of web server. It is the public handler, also known as the reverse proxy, for incoming requests and can handle thousands of concurrent connections.

9.1 Digital Ocean

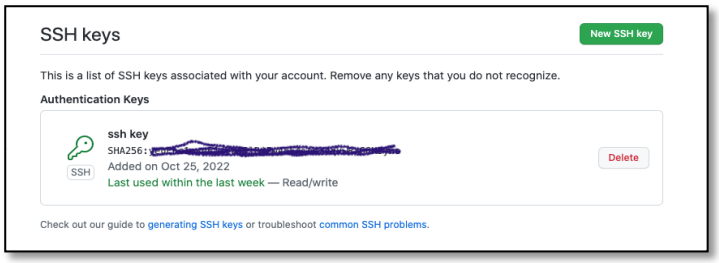

Figure 23 SSH key

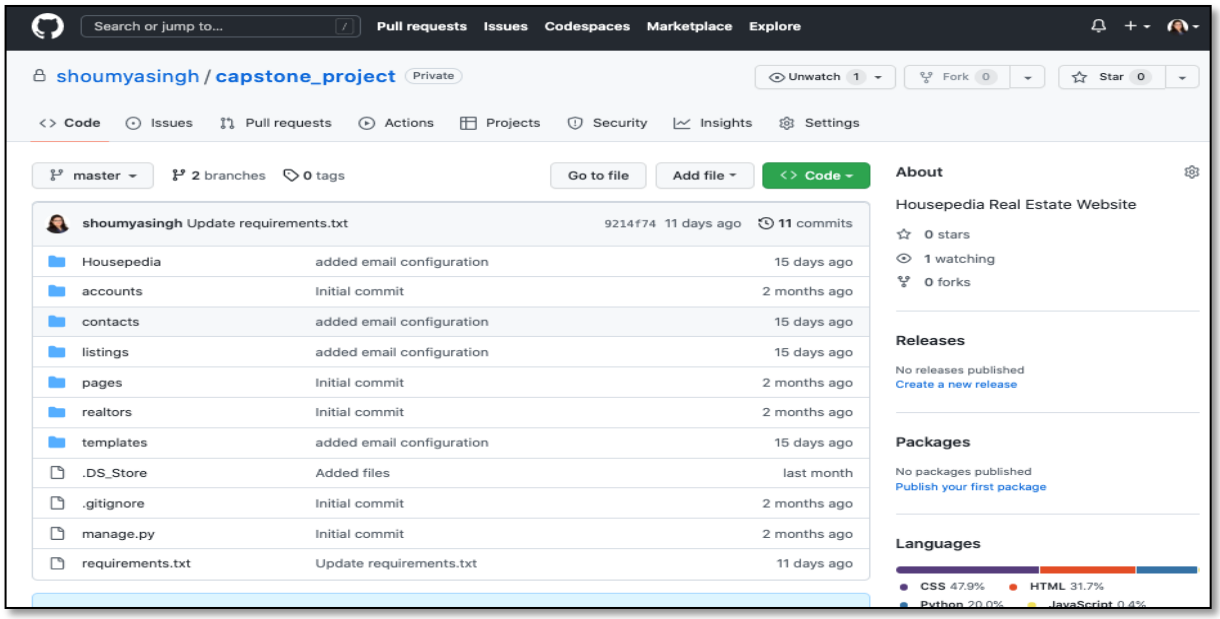

Figure 24 Github Page

DigitalOcean is a cloud hosting provider that offers cloud computing services and Infrastructure as a Service (IaaS). Personal access tokens are a substitute for using passwords to identify GitHub when using the GitHub API or the command-line interface. Personal access tokens are designed to access GitHub services on behalf of our account. GitHub now offers two types of personal access tokens: fine-grained personal access tokens and personal access tokens (traditional) (classic). GitHub recommends employing fine-grained personal access tokens rather than just personal access tokens (traditional) wherever possible. For this deployment, We have generated the fine-grained personal access tokens to pull the modified code in the master. SSH, generally called Secure Shell or Secure Socket Shell, is a network protocol that offers a secure way to access a system over an unprotected network. Ultimately, with the support of Git activities over SSH, we were all to retrieve the latest code on the DigitalOcean cloud system.

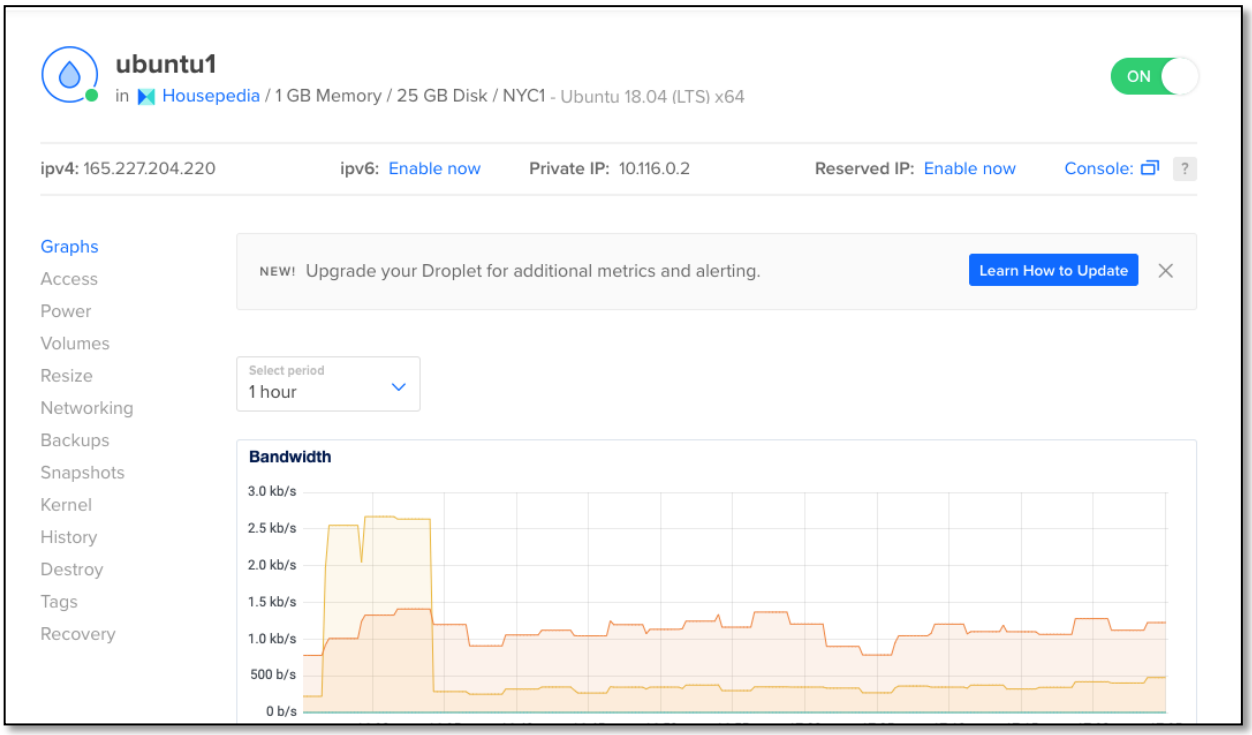

Figure 25 Digital Ocean

9.2 Gunicorn & Nginx

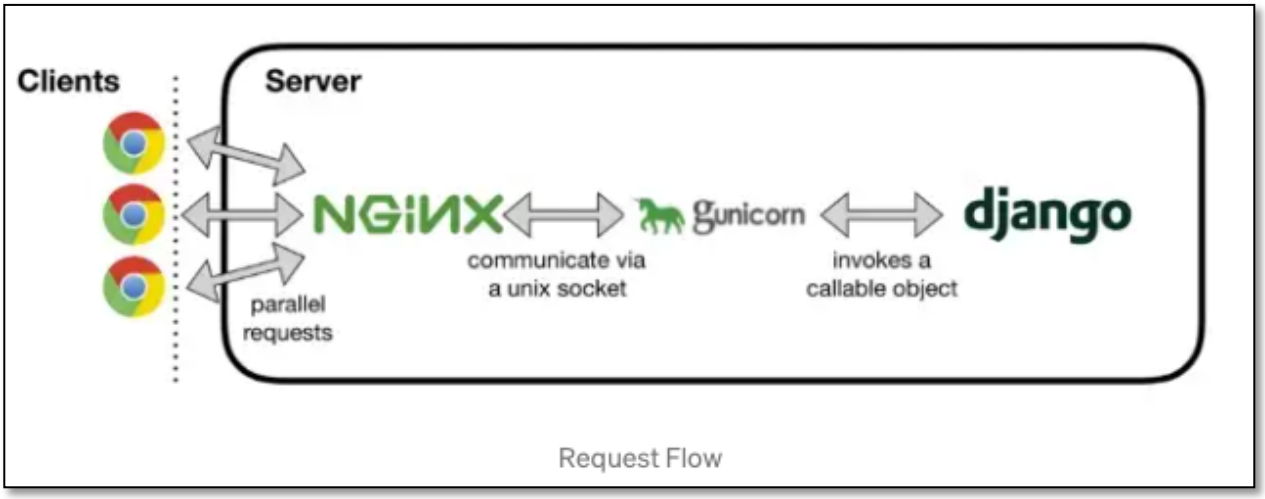

Figure 26 Request flow with Gunicorn & Nginx

Gunicorn is a server that runs programs. It translates HTTP requests into Python syntax. Gunicorn integrates the Web Server Gateway Interface (WSGI), a primary interface between web server software and web applications. Gunicorn 'Green Unicorn' is a UNIX Python WSGI HTTP Server. It is a fork worker model. The Gunicorn server is widely compatible with various web frameworks, is easily implemented, uses few server resources, and is relatively fast. **NGINX** is open-source software that can be used for web serving, reverse proxying, caching, load balancing, media streaming, and other purposes. It is a web server optimized for speed and stability. NGINX can also serve as an email proxy server (IMAP, POP3, and SMTP) as well as a reverse proxy and load balancer for HTTP, TCP, and UDP servers.

10 Coding Snipnets

1. **models.py from listings app** - It is a Python module that represents web application models as classes. It is regarded as the most crucial aspect of the App file structure. Models

define the database's structure. It describes the overall design, data set relationships, and attribute constraints.

```
from django.db import models
from datetime import datetime
from realtors.models import Realtor
from django_google_maps import fields as map_fields


# Listing Property Models.

class Listing(models.Model):
    realtor = models.ForeignKey(Realtor, on_delete=models.DO_NOTHING)
    title = models.CharField(max_length=200)
    # address = models.CharField(max_length=200)
    city = models.CharField(max_length=100)
    state = models.CharField(max_length=100)
    zipcode = models.CharField(max_length=20)
    address = map_fields.AddressField(max_length=200)
    geolocation = map_fields.GeoLocationField(max_length=100, blank=True, default='')
    description = models.TextField(blank=True)
    price = models.IntegerField()
    bedrooms = models.IntegerField()
    bathrooms = models.DecimalField(max_digits=2, decimal_places=1)
    garage = models.IntegerField(default=0)
    sqft = models.IntegerField()
    lot_size = models.DecimalField(max_digits=5, decimal_places=1)
    photo_main = models.ImageField(upload_to='photos/%Y/%m/%d/')
    photo_1 = models.ImageField(upload_to='photos/%Y/%m/%d/', blank=True)
    photo_2 = models.ImageField(upload_to='photos/%Y/%m/%d/', blank=True)
    photo_3 = models.ImageField(upload_to='photos/%Y/%m/%d/', blank=True)
    photo_4 = models.ImageField(upload_to='photos/%Y/%m/%d/', blank=True)
    photo_5 = models.ImageField(upload_to='photos/%Y/%m/%d/', blank=True)
    photo_6 = models.ImageField(upload_to='photos/%Y/%m/%d/', blank=True)
    is_published = models.BooleanField(default=True)
    list_date = models.DateTimeField(default=datetime.now, blank=True)

    def __str__(self):
        return self.title
```

2. **view.py from listings app** - In Django app structure, views play an important role. Views provide an interface for interacting with a Django web application. It contains all of the views as classes.

```python
from django.shortcuts import get_object_or_404, render
from django.core.paginator import EmptyPage, PageNotAnInteger, Paginator
from .choices import price_choices, bedroom_choices, state_choices

# Create your views here.

from .models import Listing

def index(request):
  listings = Listing.objects.order_by('-list_date').filter(is_published=True)

  paginator = Paginator(listings, 6)
  page = request.GET.get('page')
  paged_listings = paginator.get_page(page)

  context = {
    'listings': paged_listings
  }

  return render(request, 'listings/listings.html', context)

def listing(request, listing_id):
  listing = get_object_or_404(Listing, pk=listing_id)

  context = {
    'listing': listing
  }

  return render(request, 'listings/listing.html', context)

def search(request):
  queryset_list = Listing.objects.order_by('-list_date')

  # Keywords
  if 'keywords' in request.GET:
    keywords = request.GET['keywords']
    if keywords:
      queryset_list = queryset_list.filter(description__icontains=keywords)

  # City
  if 'city' in request.GET:
    city = request.GET['city']
    if city:
      queryset_list = queryset_list.filter(city__iexact=city)

  # State
  if 'state' in request.GET:
    state = request.GET['state']
```

```
    if state:
      queryset_list = queryset_list.filter(state__iexact=state)

  # Bedrooms
  if 'bedrooms' in request.GET:
    bedrooms = request.GET['bedrooms']
    if bedrooms:
      queryset_list = queryset_list.filter(bedrooms__lte=bedrooms)

  # Price
  if 'price' in request.GET:
    price = request.GET['price']
    if price:
      queryset_list = queryset_list.filter(price__lte=price)

  context = {
    'state_choices': state_choices,
    'bedroom_choices': bedroom_choices,
    'price_choices': price_choices,
    'listings': queryset_list,
    'values': request.GET
  }

  return render(request, 'listings/search.html', context)

def default_map(request):
      context = {
        'listing': listing
        }
      return render(request, 'listing.html', context)
```

3. view.py from Contacts app

```
from django.shortcuts import render, redirect
from django.contrib import messages
from django.core.mail import send_mail
from .models import Contact

def contact(request):
  if request.method == 'POST':
    listing_id = request.POST['listing_id']
    listing = request.POST['listing']
    name = request.POST['name']
    email = request.POST['email']
    phone = request.POST['phone']
    message = request.POST['message']
    user_id = request.POST['user_id']
```

```
    realtor_email = request.POST['realtor_email']

    #  Check if user has made inquiry already
    if request.user.is_authenticated:
      user_id = request.user.id
      has_contacted = Contact.objects.all().filter(listing_id=listing_id,
user_id=user_id)
      if has_contacted:
        messages.error(request, 'You have already made an inquiry for this listing')
        return redirect('/listings/'+listing_id)

    contact = Contact(listing=listing, listing_id=listing_id, name=name, email=email,
phone=phone, message=message, user_id=user_id )

    contact.save()

    # Send email
    send_mail(
      'Property Listing Inquiry',
      'There has been an inquiry for ' + listing + '. Sign into the admin panel for
more info',
      'dkranchii@outlook.com',
      [realtor_email, 'shoumya25@gmail.com'],
      fail_silently=False
    )

    messages.success(request, 'Your request has been submitted, a realtor will get
back to you soon')
    return redirect('/listings/'+listing_id)
```

Conclusion

Our Housepedia, a real estate website, is currently deployed and functioning on " http://165.227.204.220:8000/ ". We have achieved our goals as planned till week 14. The purpose and objective of the Housepedia project are achieved. The User can search for a house as per their requirements. The functionality specification like Customer inquiry email to the realtor, Google map location of the listed property, and Search of the listed property with keywords like pool, city, and number of bedrooms and bathrooms are achieved. It had been deployed and can be accessed from anywhere.

Table of Figures

FIGURE 1:FULL STACK TECHNIQUES USED 4
FIGURE 2 HOUSEPEDIA DJANGO ARCHITECTURE 6
FIGURE 3 FOLDER STRUCTURE 7
FIGURE 4 HOME PAGE OF HOUSEPEDIA 11
FIGURE 5 HOMEPAGE CONTINUE 12
FIGURE 6 ABOUT PAGE 13
FIGURE 7 LISTING PAGE 13
FIGURE 8 SINGLE LISTING PAGE 14
FIGURE 9 SEARCH PAGE 15
FIGURE 10 REGISTER & LOGIN 15
FIGURE 11 LOGIN & DASHBOARD 16
FIGURE 12 INQUIRY DIALOGUE BOX 17
FIGURE 13 EMAIL TO REALTOR 17
FIGURE 14 GOOGLE MAP IN SINGLE LISTING PAGE 18
FIGURE 15 GOOGLE MAP IN ZOOMED VIEW 19
FIGURE 16 ADMIN LISTING PAGE 20
FIGURE 17 USERS IN THE ADMIN 20
FIGURE 18 ADMIN PAGE OF RECEIVED MESSEAGE FOR INQUIRY 21
FIGURE 19 REALTOR ADMIN PAGE 21
FIGURE 20 GOOGLE MAP ON ADMIN SIDE 22
FIGURE 21 PGADMIN PAGE WITH OUR LISTING TABLE 23
FIGURE 22 LOGOS 23
FIGURE 23 SSH KEY 24
FIGURE 24 GITHUB PAGE 24
FIGURE 25 DIGITAL OCEAN 25
FIGURE 26 REQUEST FLOW WITH GUNICORN & NGINX 26

References

Vincent, W. S. (2021). *Django for Beginners: Build websites with Python and Django.* WelcomeToCode.

Gore, H., Singh, R. K., Singh, A., Singh, A. P., Shabaz, M., Singh, B. K., & Jagota, V. (2021). *Django: Web Development Simple & Fast. Annals of the Romanian Society for Cell Biology,* 25(6), 4576-4585.

Shaw, B., Bird, A., S, B. C. K., Badhwar, S., Guest, C. (2021). Web Development with Django: Learn to Build Modern Web Applications with a Python-based Framework. United Kingdom: Packt Publishing.

https://developers.google.com/maps/documentation/maps-static/start#Markers

https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-centos-7