

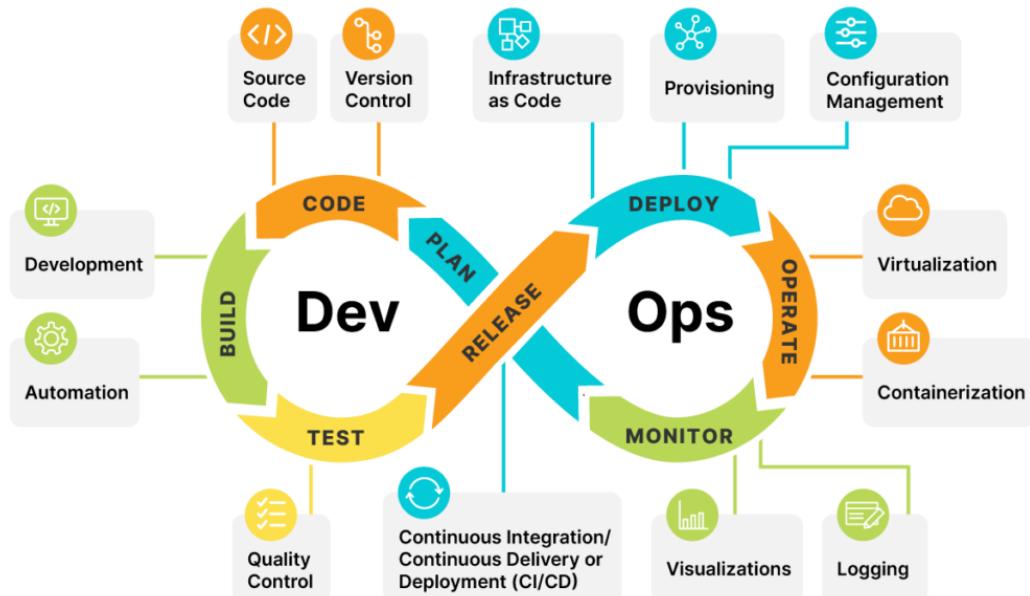


Pioneering Excellence in Education,
Research & Innovation

INTERNATIONAL INSTITUTE OF INFORMATION & TECHNOLOGY, BANGALORE

SOFTWARE PRODUCTION ENGINEERING, MAJOR PROJECT

REAL ESTATE APP WITH DEVOPS



SUBHAM BASU ROY CHOWDHURY MT2022118
VISHAL SINGH MT2022133

Table of Content:-

- Project Summary
- Tech Stack Used
 - Development
 - DevOps
- Project Links
- DevOps
 - What is DevOps
 - Why DevOps
- Application Screenshots
- Backend Descriptions Of URL
- Database
- Testing
- Logging
- API Table
- DevOps
 - Version Controlling System
 - CI/CD Pipeline using Jenkins
 - Jenkinsfile for Frontend
 - Jenkinsfile for Backend
 - Containerization (Docker)
 - Dockerfile for Backend
 - Dockerfile for Frontend
 - Docker Hub Repository for Frontend/Backend
 - Docker Compose
 - Continuous Deployment (Ansible)
 - Continuous Monitoring (ELK)
 - Ngrok
 - API Testing (Postman)
- Problem Faced
- References

Project Summary for Realest Estate:

This is a large project that shows how to implement a real estate application with a variety of different features.

Features:

- Log in and log out for a common user. Both routes are protected by JSON Web Token.
- New User creation.
- Uploading images and storing them inside a folder segregated by currency date and month and year.
- Creation of admin using the superuser command in Django to update the details of the real estate by the respective agents.
- Implementation of Pagination and usage of custom Django Admin panel.
- Sending email with the help of google Gmail API and SMTP Port.

Tech Stack Used here:

1. Development:

- **Language:** Python + Javascript
- **Backend:** Django, Django Rest Framework
- **Database:** MySQL
- **Frontend:** React

2. DevOps:

- **Version Control:** Git and GitHub
- **CI/CD:** Jenkins
- **Build:** Pip(python) + npm(react)
- **Containerization:** Docker + Docker-Compose
- **Configuration Management:** Ansible
- **Monitoring:** ELK Stack
- **Ngrok:** For tunneling the Jenkins from localhost port to the public IP

- **Testing:** Postman
- **Logging:** Python logging library

Important Project Links:

Backend:

- GitHub:
https://github.com/testusername190/Realest_Estate_Backend.git
- DockerHub (Image):
https://hub.docker.com/repository/docker/sbrc1996/realest_estate/general

Frontend:

- GitHub:
https://github.com/testusername190/Realest_Estate_Frontend.git
- DockerHub(Image):
<https://hub.docker.com/r/vishalsin25/spe-major-front>

DevOps:

What is DevOps?

DevOps is a group of concepts emerging from the collision of two trends. It combines software development with IT ops. It aims to decrease SDLC time and provide continuous delivery and integration functionality to produce high-quality software.

Why DevOps?

It has now become important to include DevOps in software development as it enables faster development of products, faster bug fixes, and easier maintenance as compared to traditional methods

Application Screenshots:

Home Page

The screenshot shows the Realest Estate website's home page. At the top, there is a dark header bar with the title "Realest Estate". On the right side of the header are "Login" and "Sign Up" buttons. Below the header is a navigation bar with four items: "Home", "Listings", "About", and "Contact". Underneath the navigation bar is a search form with various filters. The filters include dropdowns for "Sale or Rent" (For Sale), "Days Listed" (1 of less), "Bedrooms" (0+), "Home Type" (House), and "Baths" (0+). There are also dropdowns for "Sqft" (1000+), "Minimum Price" (\$0+), and "Has Photos" (1+). Other filter options include "Keywords" (text input field), "Open Houses" (checkbox), and a "Save" button.

Listings Page

The screenshot shows the Realest Estate website's listings page. At the top, there is a browser toolbar with various icons and a status bar indicating the date and time. Below the toolbar is a header bar with the title "Realest Estate". On the right side of the header are "Login" and "Sign Up" buttons. Below the header is a navigation bar with four items: "Home", "Listings", "About", and "Contact". The main content area displays three property listings in cards:

- Assami Bari**: A modern white building with a swimming pool. Address: 22, Panbazar Road, Guwahati, Assam 781001, Guwahati, Assam. Price: \$350,000. For Sale. Bedrooms: 3. Bathrooms: 3.0. [View Listing](#)
- Namma Bengaluru**: A modern white building with a balcony. Address: 14, Jeevan Bhima Nagar, Bengaluru, Karnataka 560075, Bengaluru, Karnataka. Price: \$800,000. For Sale. Condo. Sqft: 2000. [View Listing](#)
- Chennai Homes**: A yellow house with a porch. Address: 63, Nungambakkam High Road, Chennai, Tamil Nadu 600034, Chennai, Tamil Nadu. Price: \$500,000. For Rent. House. Sqft: 1600. [View Listing](#)

At the bottom of the page, there is a navigation bar with links for "Previous", "1", "2", "3", and "Next".

About Page

Activities Brave Web Browser 29 °C Tue May 9 10:50:59 PM 62 %
Strivers A2Z DSA Course How the INTERNET Works Realest_Estate_Pipeline Realest Estate Report Realest Estate - About +
Support for... IRCTC Next... Oracle Dat... Medicine e... 10 Amazon... Create Res... CS 97SI: In... UC Berkeley... General Kn... Rice's The... WOEE | Fr... Book your...
http://localhost:3000/about
2121212121
jason@gmail.com
sdhfbdbhfrerfgye

Meet our awesome team!



Vipul Ahuja
8013421582
vipul@gmail.com

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially un-



Rahul Sinha
6291511603
rahul@gmail.com

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially un-



Murali Vijay
6398745100
Murali@gmail.com
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially un-

Contact Page

Activities Brave Web Browser 29 °C Tue May 9 10:51:47 PM 62 %
Strivers A2Z DSA Course How the INTERNET Works Realest_Estate_Pipeline Realest Estate Report Realest Estate - Contact +
Support for... IRCTC Next... Oracle Dat... Medicine e... 10 Amazon... Create Res... CS 97SI: In... UC Berkeley... General Kn... Rice's The... WOEE | Fr... Book your...
http://localhost:3000/contact
Realest Estate Login Sign Up

Home Listings About Contact

Name*

Email*

Subject*

Message

Login Page

Sign In

Sign into your Account

Login

Don't have an account? [Sign Up](#)

SignUp

Sign Up

Create your Account

Register

Backend:



In **Django**, the **code flow** is:

- We first create our app that will handle a particular functionality and add it to the settings.py file.
- We define our Model/Entity which will eventually form the table in our database.
- We register the given model with the Django Admin panel.
- We create the URL which will serve as the API Endpoints.
- We create our SERIALIZER file that will help us in doing the serialization and deserialization for the request and response coming and going to the client side.
- We write our business logic inside the VIEWS file.

Apps in our Project:

- **Accounts:** Stores the details of the Users visiting our website, also handles the login/logout and the signup features. JWT is used here to provide maximum security to the routes.
- **Contacts:** Handles the people who have queries regarding the website and can email us the detail of their query. Google API is used here to send emails.
- **Listings:** The houses along with their details and the photos uploaded by the Agents are handled here.
- **Realtors:** Handles the details of the real estate agents.

URLs defined in our Project:

- Default URL:

```
36 lines (31 sloc) | 2.06 KB
1 """realest_estate URL Configuration
2
3 The `urlpatterns` list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/4.1/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import: from my_app import views
8         2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
0     1. Add an import: from other_app.views import Home
1     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
2 Including another URLconf
3     1. Import the include() function: from django.urls import include, path
4     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
5 """
6 from django.contrib import admin
7 from django.urls import path, include, re_path
8 from django.views.generic import TemplateView
9 from django.conf import settings
0 from django.conf.urls.static import static
1 from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView
2
3 urlpatterns = [
4     path('api-auth/', include('rest_framework.urls')),
5     path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),      # API for the obtaining JWT token during login session.
6     path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),      # API for the Account/User creation, SignUp Page
7     path('api/accounts/', include('accounts.urls')),                                    # API for the realtors page (Real Estate Agents)
8     path('api/realtors/', include('realtors.urls')),                                 # API for the listings page (Listings posted by Agents)
9     path('api/listings/', include('listings.urls')),                                # API for the contacts page (Contacts posted by Users)
0     path('api/contacts/', include('contacts.urls')),                               # API for the admin page (Super User)
1     path('admin/', admin.site.urls),                                              # For accessing the media folder through the API
2 ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
3
4 urlpatterns += [re_path(r'^.*', TemplateView.as_view(template_name='index.html'))]
5
```

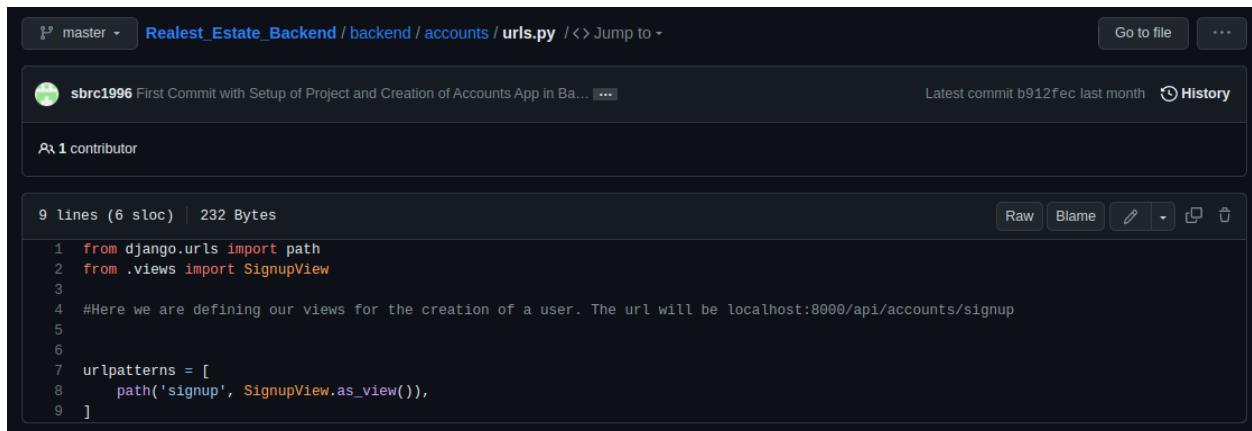
This is the URL configuration file for a Django project named "realest_estate". It defines the URLs for the APIs and admin pages of the project, as well as a catch-all route for serving a single-page application. The above code defines the URL patterns for the API endpoints of the project. It starts by importing necessary modules, including the admin module for the Django admin site, the path, include, and re_path modules for defining URL patterns, and the TemplateView module for serving templates. It also imports the TokenObtainPairView and TokenRefreshView classes from the rest_framework_simplejwt.views module for handling authentication with JSON Web Tokens (JWTs). The urlpatterns variable is a list of URL patterns for the project. The first seven URL patterns define the API endpoints for the project. Each path maps to a different API endpoint. The include function is used to include the URLs from the respective app, which are defined in separate URL configuration files.

For example, the api/accounts/ path maps to the URLs defined in the accounts.urls file. The path('admin/', admin.site.urls) URL pattern maps to the Django admin site, which provides an interface for managing the project's data and settings.

The + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT) code is used to serve static media files during development. It adds a URL pattern that maps to the project's MEDIA_ROOT directory, which is defined in the project's settings file. This allows the API to serve media files, such as images or videos, during development.

This code defines a catch-all URL pattern that serves a single-page application. The re_path function is used to define a regular expression pattern that matches any URL. The TemplateView.as_view() function is used to create a view that serves the index.html template, which is the entry point for the single-page application. By defining a catch-all URL pattern, any request that doesn't match one of the other URL patterns will be handled by the single-page application.

- **Accounts URL:**



A screenshot of a GitHub repository page for 'Realest_Estate_Backend'. The 'backend/accounts/urls.py' file is displayed. The code defines a URL pattern for a SignupView:

```
from django.urls import path
from .views import SignupView

urlpatterns = [
    path('signup', SignupView.as_view()),
]
```

This is a Django URL configuration file that defines a single URL pattern for the creation of a user. The pattern maps the signup URL path to the SignupView view.

```
from django.urls import path
from .views import SignupView
```

This code imports the path function from the django.urls module and the SignupView view from the views.py file in the current directory.

```
urlpatterns = [ path('signup', SignupView.as_view()),]
```

This code defines the URL pattern for the signup endpoint. The path function takes two arguments: the URL path as a string, and the view that should handle requests to that path.

In this case, the URL pattern is simply signup, which means that the endpoint will be accessed at localhost:8000/api/accounts/signup. The SignupView.as_view() method is used to create an instance of the SignupView class, which is responsible for handling requests to this endpoint.

The SignupView view should handle the logic for creating a new user account. This typically involves validating user input, creating a new User object, and storing it in the database. The exact implementation of this view will depend on the specific requirements of the project.

- **Contacts URL:**



A screenshot of a GitHub repository interface. The repository is named 'Realest_Estate_Backend'. The file being viewed is 'backend/contacts/urls.py'. The code in the file is:

```
from django.urls import path
from .views import ContactCreateView

urlpatterns = [
    path('', ContactCreateView.as_view()),
]
```

This is a Django URL configuration file that defines a single URL pattern for creating a new contact. The pattern maps the root URL path to the ContactCreateView view. This code imports the path function from the django.urls module and the ContactCreateView view from the views.py file in the current directory.

```
urlpatterns = [ path("", ContactCreateView.as_view()),]
```

This code defines the URL pattern for the root endpoint. The path function takes two arguments: the URL path as a string, and the view that should handle requests to that path.

In this case, the URL pattern is an empty string "", which means that the endpoint will be accessed at localhost:8000/api/contacts/. The ContactCreateView.as_view() method is used to create an instance of the ContactCreateView class, which is responsible for handling requests to this endpoint.

The ContactCreateView view should handle the logic for creating a new contact object. This typically involves validating user input, creating a new Contact object, and storing it in the database. The exact implementation of this view will depend on the specific requirements of the project.

- **Listings URL:**



A screenshot of a GitHub repository interface. The repository is named 'Realestate_Backend'. The current branch is 'master'. The file being viewed is 'backend/listings/urls.py'. The commit history shows a single commit by 'testusername190' titled 'Commit after Adding Model Testing'. The code in the file is as follows:

```
1 from django.urls import path
2 from .views import ListingsView, ListingView, SearchView
3
4 # Declare the URL for the listings app here.
5
6 urlpatterns = [
7     path('', ListingsView.as_view(), name="ListALL"),
8     path('search', SearchView.as_view()),
9     path('<slug>', ListingView.as_view()),      # Used for listing a particular view, not by PK(id) but by Slug field.
10 ]
```

This is a Django URL configuration file that defines the URL patterns for the listings app.

```
from django.urls import path
```

```
from .views import ListingsView, ListingView, SearchView
```

This code imports the path function from the django.urls module and the three views (ListingsView, ListingView, and SearchView) from the views.py file in the current directory.

```
urlpatterns = [
    path("", ListingsView.as_view(), name="ListALL"),
    path('search', SearchView.as_view()),
    path('<slug>', ListingView.as_view()),
]
```

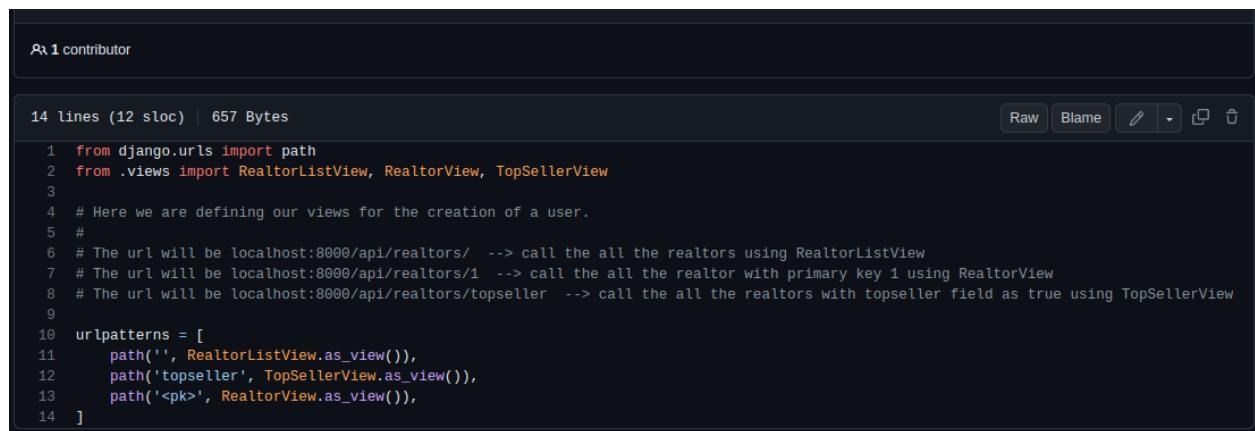
This code defines the URL patterns for the listings app. The first URL pattern is for the root endpoint ". It maps to the ListingsView view and specifies the name ListALL for this URL pattern. This means that this endpoint can be accessed using localhost:8000/api/listings/.

The second URL pattern is for the search endpoint. It maps to the SearchView view and can be accessed using localhost:8000/api/listings/search.

The third URL pattern is a dynamic URL pattern that includes a slug parameter. It maps to the ListingView view and can be accessed using localhost:8000/api/listings/<slug>, where <slug> is a dynamic value. This URL pattern is used for accessing a particular listing, not by primary key (id), but by the slug field.

All three URL patterns use the as_view() method to create an instance of the respective view classes. The name parameter is used to specify a name for the URL pattern, which can be used in reverse URL resolution (i.e., generating URLs based on the name of the pattern).

- **Realtors URL:**



A screenshot of a GitHub code review interface showing a file named 'urls.py'. The file contains 14 lines of Python code defining URL patterns for a 'realtors' application. The code imports views from 'views.py' and uses the 'path' function from 'django.urls' to define three URL patterns: an empty string for 'RealtorListView', 'topseller' for 'TopSellerView', and '' for 'RealtorView'. Each pattern uses the 'as_view()' method to create a view instance.

```
14 lines (12 sloc) | 657 Bytes
1 from django.urls import path
2 from .views import RealtorListView, RealtorView, TopSellerView
3
4 # Here we are defining our views for the creation of a user.
5 #
6 # The url will be localhost:8000/api/realtors/ --> call the all the realtors using RealtorListView
7 # The url will be localhost:8000/api/realtors/1 --> call the all the realtor with primary key 1 using RealtorView
8 # The url will be localhost:8000/api/realtors/topseller --> call the all the realtors with topseller field as true using TopSellerView
9
10 urlpatterns = [
11     path('', RealtorListView.as_view()),
12     path('topseller', TopSellerView.as_view()),
13     path('<pk>', RealtorView.as_view()),
14 ]
```

The given code defines the URL patterns for the realtors app. It imports the required views from the views.py module. The first URL pattern " is associated with the RealtorListView view, which will return a list of all the realtors. The second URL pattern topseller is associated with the TopSellerView view, which will return a list of all the realtors whose topseller field is set to True. The third URL pattern '<pk>' is associated with the

RealtorView view, which will return the details of a single realtor with the given primary key.

Here's a breakdown of each of the URL patterns:

": This matches the base URL for the realtors app (/api/realtors/), and is associated with the RealtorListView view.

'topseller': This matches the URL /api/realtors/topseller/, and is associated with the TopSellerView view.

'<pk>': This matches any URL that has an integer after /api/realtors/. The integer is captured as a named parameter pk and is passed to the RealtorView view to retrieve the details of the realtor with that primary key.

Database: MySQL:

```
mysql> use realest_estate;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_realest_estate |
+-----+
| accounts_useraccount
| accounts_useraccount_groups
| accounts_useraccount_user_permissions
| auth_group
| auth_group_permissions
| auth_permission
| contacts_contact
| django_admin_log
| django_content_type
| django_migrations
| django_session
| listings_listing
| realtors_realtor
+-----+
13 rows in set (0.00 sec)

mysql> █
```

The tables above are the ones present in our database realest_estate.

Settings.py (Configurations):

```
27 # SECURITY WARNING: don't run with debug turned on in production!
28 DEBUG = True
29
30 ALLOWED_HOSTS = ['*']
31
32
33 # Application definition
34
35 INSTALLED_APPS = [
36     'django.contrib.admin',
37     'django.contrib.auth',
38     'django.contrib.contenttypes',
39     'django.contrib.sessions',
40     'django.contrib.messages',
41     'django.contrib.staticfiles',
42     'corsheaders',
43     'rest_framework',
44     'accounts',
45     'realtors',
46     'listings',
47     'contacts',
48 ]
```

Debug= True helps in debugging the code.

ALLOWED_HOSTS = ["*"] means code can run in any host.

```
81
82 # Database
83 # https://docs.djangoproject.com/en/4.1/ref/settings/#databases
84
85 DATABASES = {
86     # 'default': {
87         #     'ENGINE': 'django.db.backends.mysql',
88         #     'NAME': 'realest_estate',
89         #     'HOST': 'mydb',           # '127.0.0.1', Change this to the IP
90         #     'PORT': '3306',
91         #     'USER': 'root',
92         #     'PASSWORD': 'subham123',
93     # }
94     'default': {
95         'ENGINE': 'django.db.backends.mysql',
96         'NAME': os.environ.get('DB_NAME'),
97         'HOST': os.environ.get('DB_HOST'),
98         'PORT': os.environ.get('DB_PORT'),
99         'USER': os.environ.get('DB_USER'),
100        'PASSWORD': os.environ.get('DB_PASSWORD'),
101    }
102 }
103 }
```

The settings for the Database connection of the backend code with the MYSQL database. Initially, we had directly given the information in the Dictionary to do the connection but later we changed and provided the

information through environment variables so that it can run in any environment.

```
103
104 EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
105 EMAIL_HOST = 'smtp.gmail.com'
106 EMAIL_PORT = 587
107 EMAIL_HOST_USER = 'testusername190@gmail.com'
108 EMAIL_HOST_PASSWORD = 'nkzqsqaqfgivtrhg'
109 EMAIL_USE_TLS = True
110
111
112 # Password validation
113 # https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validation
```

Email configuration using Google API and SMTP Port 587 with its API key.

```
155 REST_FRAMEWORK = {
156     'DEFAULT_PERMISSION_CLASSES': [
157         'rest_framework.permissions.IsAuthenticated'
158     ],
159     'DEFAULT_AUTHENTICATION_CLASSES': [
160         'rest_framework_simplejwt.authentication.JWTAuthentication'
161     ],
162     'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
163     'PAGE_SIZE': 3
164 }
165
166 CORS_ORIGIN_ALLOW_ALL = True
167
168 FILE_UPLOAD_PERMISSIONS=0o640
169
170 AUTH_USER_MODEL = 'accounts.UserAccount'
171
```

Library addition to including the JWT Refresh and Access Tokens and adding CORS-Header to interact with our React Frontend.

Testing:

For testing, we have utilized the Django inbuild testing library **django.test**

```
1 from django.test import TestCase, Client
2 from django.urls import reverse
3 from .models import Realtor
4
5 # Create your tests here.
6
7 class TestRealtorsModels(TestCase):
8     def setUp(self):
9         Realtor.objects.create(
10             name='Test Realtor',
11             phone='555-555-5555',
12             email='testrealtor@example.com',
13             top_seller=True,
14         )
15     def test_realtor_model(self):
16         realtor = Realtor.objects.get(name='Test Realtor')
17         self.assertEqual(realtor.phone, '555-555-5555')
18         self.assertEqual(realtor.email, 'testrealtor@example.com')
19         self.assertTrue(realtor.top_seller)
20
21 # python3 manage.py test realtors.tests
```

Testing File for our Realtors App.

```
subham@subham-HP-Laptop-15s-fq2xxx:~/Desktop/RealEstate/backend$ python3 manage.py test realtors.tests
Found 1 test(s).
Creating test database for alias 'default'...
INFO 2023-05-08 11:43:24,732 django views views.py 20 SignupView
Inside the SignupView API of accounts APP. Here we are creating an account of a user. JWT token not needed.
INFO 2023-05-08 11:43:24,733 django views views.py 16 RealtorListView
Inside the RealtorListView API of realtor App. Here we are getting all the realtors from DB..
INFO 2023-05-08 11:43:24,733 django views views.py 24 RealtorView
Inside the RealtorView API of realtor App. Here we are a particular realtors from DB based on its PK..
INFO 2023-05-08 11:43:24,733 django views views.py 30 TopSellerView
Inside the TopSellerView API of realtor App. Here we are getting all the realtors who are topsellers..
INFO 2023-05-08 11:43:24,734 django views views.py 17 ListingsView
Inside the ListingsView API of listings App. Here we are getting all the data filtered by list date and ispublished value..
INFO 2023-05-08 11:43:24,734 django views views.py 27 ListingView
Inside the ListingView API of listings App. Here we are getting the info of a particular View using id and JWT token is mandatory here..
INFO 2023-05-08 11:43:24,735 django views views.py 35 SearchView
Inside the SearchView API of listings App. Here we are getting the search field values from the frontend Form and using it to retrieve data from the backend. This is done in the home page itself.
System check identified some issues:

WARNINGS:
?: (staticfiles.W004) The directory '/home/subham/Desktop/RealEstate/backend/build/static' in the STATICFILES_DIRS setting does not exist.

System check identified 1 issue (0 silenced).
/home/subham/.local/lib/python3.10/site-packages/django/db/models/fields/__init__.py:1564: RuntimeWarning: DateTimeField Realtor.date_hired received a naive datetime (2023-05-08 11:43:24.769016) while time zone support is active.
    warnings.warn(
...
-----
Ran 1 test in 0.006s

OK
Destroying test database for alias 'default'...
subham@subham-HP-Laptop-15s-fq2xxx:~/Desktop/RealEstate/backend$
```

The output of Testing of Realtors App.

```

45 lines (38 sloc) | 1.49 KB
1 from django.test import TestCase, Client
2 from django.urls import reverse
3 from .models import Listing
4 from realtors.models import Realtor
5
6 # Create your tests here.
7
8 class TestListings(TestCase):
9     def setUp(self):
10         self.realtor = Realtor.objects.create(
11             name='Test Realtor',
12             phone='555-555-5555',
13             email='testrealtor@example.com',
14             top_seller=True,
15         )
16
17         Listing.objects.create(
18             title = 'Mera Ghar',
19             address = '433, Jadav Ghosh Road',
20             city = 'Kolkata',
21             state = 'West Bengal',
22             description = 'A 3 storey house in Behala Sarsuna Kolkata about 3 km from metro station',
23             sale_type = 'For Sale',
24             price = 10000000,
25             bedrooms = 5,
26             bathrooms = 5.0,
27             home_type = 'House',
28             sqft = 1000,
29             realtor = self.realtor
30         )
31
32     def test_listings_model(self):
33         listing = Listing.objects.get(title='Mera Ghar')
34         self.assertEqual(listing.address, '433, Jadav Ghosh Road')
35         self.assertEqual(listing.city, 'Kolkata')
36         self.assertEqual(listing.state, 'West Bengal')
37         self.assertNotEqual(listing.description, '433, Jadav Ghosh Road')
38         self.assertEqual(listing.price, 10000000)
39         self.assertEqual(listing.bedrooms, 5)
40         self.assertEqual(listing.sqft, 1000)
41         self.assertEqual(listing.address, '433, Jadav Ghosh Road')
42
43
44
45 # python3 manage.py test listings.tests

```

Testing File for Listings App

```

subham@subham-HP-Laptop-15s-fq2xxx:~/Desktop/RealEstate/backend$ python3 manage.py test listings.tests
Found 1 test(s).
Creating test database for alias 'default'...
INFO 2023-05-08 11:44:42,752 django views.views.py 20 SignupView
Inside the SignupView API of accounts APP. Here we are creating an account of a user. JWT token not needed.
INFO 2023-05-08 11:44:42,753 django views.views.py 16 RealtorListView
Inside the RealtorListView API of realtor App. Here we are getting all the realtors from DB..
INFO 2023-05-08 11:44:42,753 django views.views.py 24 RealtorView
Inside the RealtorView API of realtor App. Here we are a particular realtors from DB based on its PK..
INFO 2023-05-08 11:44:42,753 django views.views.py 30 TopSellerView
Inside the TopSellerView API of realtor App. Here we are getting all the realtors who are topsellers..
INFO 2023-05-08 11:44:42,754 django views.views.py 17 ListingsView
Inside the ListingsView API of listings App. Here we are getting all the data filtered by list date and ispublished value..
INFO 2023-05-08 11:44:42,754 django views.views.py 27 ListingView
Inside the ListingView API of listings App. Here we are getting the info of a particular View using id and JWT token is mandatory here..
INFO 2023-05-08 11:44:42,754 django views.views.py 35 SearchView
Inside the SearchView API of listings App. Here we are getting the search field values from the frontend Form and using it to retrieve data from the backend. This is done in the home page itself.
System check identified some issues:

WARNINGS:
?: (staticfiles.W004) The directory '/home/subham/Desktop/RealEstate/backend/build/static' in the STATICFILES_DIRS setting does not exist.

System check identified 1 issue (0 silenced).
/home/subham/.local/lib/python3.10/site-packages/django/db/models/fields/__init__.py:1564: RuntimeWarning: DateTimeField Realtor.date_hired received a naive datetime (2023-05-08 11:44:42.7778686) while time zone support is active.
    warnings.warn(
...
Ran 1 test in 0.008s

OK
Destroying test database for alias 'default'...
subham@subham-HP-Laptop-15s-fq2xxx:~/Desktop/RealEstate/backend$ 

```

The output of Testing of Listings App.

```

1 from django.test import TestCase, Client
2 from django.urls import reverse
3 from .models import Contact
4
5 # Create your tests here.
6
7 class TestContactsModel(TestCase):
8     def setUp(self):
9         Contact.objects.create(
10             name= 'Test User',
11             email= 'testuser@example.com',
12             subject = 'This is a test query',
13             message = 'Yo Yo!! Whats Up Dude??'
14         )
15     def test_Contacts_model(self):
16         contact = Contact.objects.get(name= 'Test User')
17         self.assertEqual(contact.email , 'testuser@example.com')
18         self.assertNotEqual(contact.subject, 'dskjfbdsf')
19         self.assertEqual(contact.message , 'Yo Yo!! Whats Up Dude??')
20
21 #command to execute the test script : python3 manage.py test contacts.tests

```

Testing File for Contacts App.

```

subham@subham-HP-Laptop-15s-fq2xxx:~/Desktop/RealEstate/backend$ python3 manage.py test contacts.tests
Found 1 test(s).
Creating test database for alias 'default'...
INFO 2023-05-08 11:45:35,167 django views views.py 20 SignupView
    Inside the SignupView API of accounts APP. Here we are creating an account of a user. JWT token not needed.
INFO 2023-05-08 11:45:35,168 django views views.py 10 RealtorListView
    Inside the RealtorListView API of realtor App. Here we are getting all the realtors from DB..
INFO 2023-05-08 11:45:35,168 django views views.py 24 RealtorView
    Inside the RealtorView API of realtor App. Here we are a particular realtors from DB based on its PK.
INFO 2023-05-08 11:45:35,168 django views views.py 30 TopsellerView
    Inside the TopsellerView API of realtor App. Here we are getting all the realtors who are topsellers..
INFO 2023-05-08 11:45:35,169 django views views.py 17 ListingsView
    Inside the ListingsView API of listings App. Here we are getting all the data filtered by list date and ispublished value..
INFO 2023-05-08 11:45:35,169 django views views.py 27 ListingView
    inside the ListingView API of listings App. Here we are getting the info of a particular View using id and JNT token is mandatory here..
INFO 2023-05-08 11:45:35,169 django views views.py 35 SearchView
    Inside the SearchView API of listings App. Here we are getting the search field values from the frontend Form and using it to retrieve data from the backend. This is done in the home page itself.
System check identified some issues:

WARNINGs:
?: (staticfiles.W004) The directory '/home/subham/Desktop/RealEstate/backend/build/static' in the STATICFILES_DIRS setting does not exist.

System check identified 1 issue (0 silenced).
/home/subham/.local/lib/python3.10/site-packages/django/db/models/fields/__init__.py:1564: RuntimeWarning: DateTimeField Contact.contact_date received a naive datetime (2023-05-08 11:45:35.195182) while time zone support is active.
    warnings.warn(
.
-----
Ran 1 test in 0.007s

OK
Destroying test database for alias 'default'...
subham@subham-HP-Laptop-15s-fq2xxx:~/Desktop/RealEstate/backend$
```

The output of Testing of Contacts App.

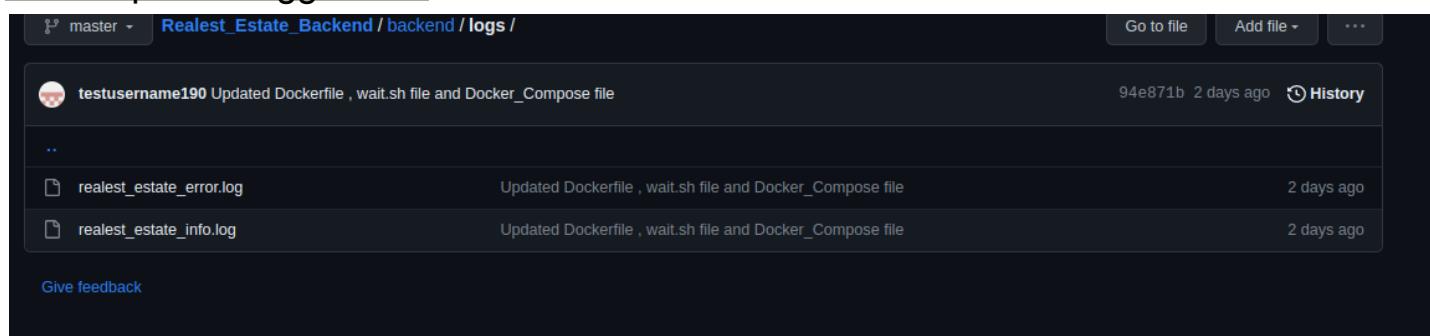
Logging:

Configurations in settings.py for our Logger:

```
221 LOGGERS = (
222     {
223         "django": {
224             "handlers": ["console_handler", "info_handler"],
225             "level": "INFO",
226         },
227         "django.request": {
228             "handlers": ["error_handler"],
229             "level": "INFO",
230             "propagate": True,
231         },
232         "django.template": {
233             "handlers": ["error_handler"],
234             "level": "DEBUG",
235             "propagate": True,
236         },
237         "django.server": {
238             "handlers": ["error_handler"],
239             "level": "INFO",
240             "propagate": True,
241         },
242     },
243 )
244
245
246 LOGGING = {
247     "version": 1,
248     "disable_existing_loggers": False,
249     "formatters": FORMATTERS[0],
250     "handlers": HANDLERS,
251     "loggers": LOGGERS[0],
252 }
```

For logging, we have used Python's inbuild logging library **logging** and we are doing persistent storage of log file through our docker-compose file using Volumes.

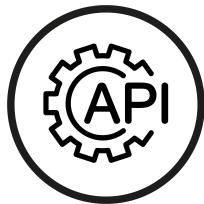
The output of Loggin File:



A screenshot of a GitHub commit history page. The commit by 'testusername190' is shown, along with two log files: 'realest_estate_error.log' and 'realest_estate_info.log', both updated 2 days ago.

File	Description	Last Updated
realest_estate_error.log	Updated Dockerfile , wait.sh file and Docker_Compose file	2 days ago
realest_estate_info.log	Updated Dockerfile , wait.sh file and Docker_Compose file	2 days ago

API Table:



<u>API Endpoint</u>	<u>Request Type</u>	<u>Details of the API</u>
http://127.0.0.1:8000/api/accounts/signup	POST	Creation of a new User Account
http://127.0.0.1:8000/api/token/	POST	Validating the credentials of the existing user and generating a JWT token on successful verification
localhost:8000/api/realtors/	GET	Get all the realtors
localhost:8000/api/realtors/1	GET	Get a particular realtor by id/pk
localhost:8000/api/realtors/topseller	GET	Get only the top-selling realtors
http://127.0.0.1:8000/api/listings/	GET	Get all the listings
http://127.0.0.1:8000/api/listings/Shyama-Prasad-Mukherjee-Rd	GET	Get a particular listing by its slug field
http://127.0.0.1:8000/api/listings/search	POST	Get the data from the client used for searching the listings data
http://127.0.0.1:8000/api/contacts/	POST	Send a request email with the client email id and query

DEVOPS:



Version Controlling System:

We have used Github as VCS. GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code.

Git hub repo for Backend Code:

testusername190 / Realest_Estate_Backend (Public)

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

testusername190 Updated Dockerfile , wait.sh file and Docker_Compose file 94e871b 2 days ago 10 commits

backend Updated Dockerfile , wait.sh file and Docker_Compose file 2 days ago

db Updated Dockerfile , wait.sh file and Docker_Compose file 2 days ago

Dockerfile Updated Dockerfile , wait.sh file and Docker_Compose file 2 days ago

Google Account Password.txt Backend Push 3 weeks ago

SuperUserDetails.txt Updated Dockerfile , wait.sh file and Docker_Compose file 2 days ago

docker_compose_spe.yaml Updated Dockerfile , wait.sh file and Docker_Compose file 2 days ago

link for Git Setup Modified Dockerfile 3 weeks ago

About SPE Major Project Property Listing App Backend using Django and DjangoRestFramework

0 stars 1 watching 0 forks

Releases No releases published Create a new release

Packages No packages published Publish your first package

Git hub repo for Frontend Code:

testusername190 / Realest_Estate_Frontend (Public)

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

testusername190 about and contact f9ca075 3 weeks ago 9 commits

public Initialize project using Create React App 3 weeks ago

src about and contact 3 weeks ago

.gitignore Initialize project using Create React App 3 weeks ago

Github Token.txt Frontend 3 weeks ago

README.md Initialize project using Create React App 3 weeks ago

package-lock.json frontend with listing form 3 weeks ago

package.json frontend with listing form 3 weeks ago

About SPE major project, front end part using React

Readme 0 stars 1 watching 0 forks

Releases No releases published Create a new release

Packages No packages published Publish your first package



CI/CD Pipeline using Jenkins: Jenkins

A CI/CD pipeline is a series of steps that must be performed in order to deliver a new version of the software. We have used Jenkins for our pipeline. Jenkins is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

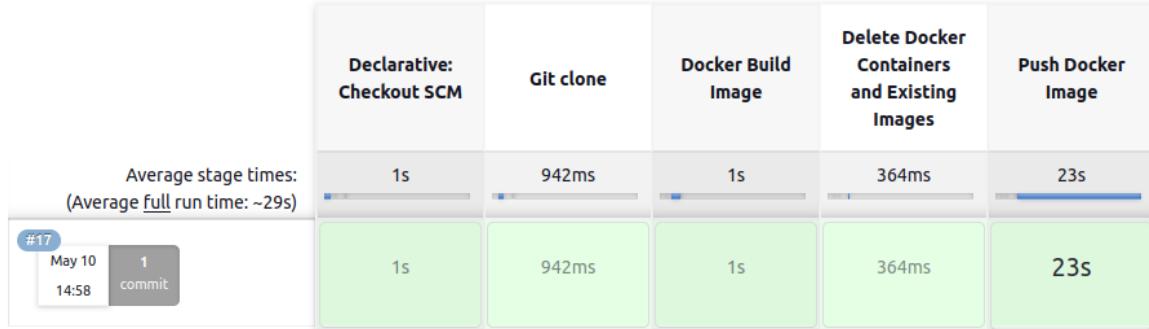
Pipeline run for Backend:

Stage View

	Declarative: Checkout SCM	Git Pull	Change Permissions and grant permission to create the log files..	Install dependencies in the pipeline folder..	Build + Test Cases	Docker Build Image for Django Backend	Push Django Python Docker Image	Delete Docker Containers and Existing Images	Pull Frontend Docker Image from the Docker Hub	Ansible Deploy Stage
Average stage times: (Average full run time: ~1min)	1s	930ms	321ms	1s	8s	20s	30s	527ms	3s	6s
#70 May 09 22:47 22:47	1s	976ms	314ms	1s	8s	20s	30s	642ms	6s	12s
#69 May 09 22:44	1s	884ms	328ms	1s	8s	19s	30s	412ms Failed	123ms Failed	44ms Failed

Permalinks

Pipeline run for Frontend:



JenkinsFile for Frontend:

This Jenkinsfile for frontend is pipeline scripts that will automate the build and will create the docker image for the front-end and will push the image file to the Docker hub.

Below is the script for the Jenkinsfile for automation:

```
1  pipeline {  
2      environment{  
3          dockerimage=""  
4      }  
5      agent any  
6      stages {  
7          stage('Git clone') {  
8              steps {  
9                  git 'https://github.com/testusername190/Realest_Estate_Frontend.git'  
10             }  
11         }  
12     }  
13 }
```

Git clone: Clones the source code from a GitHub repository for the front-end.

```
    }  
    stage('Docker Build Image') {  
        steps {  
            script{  
                dockerimage=docker.build "vishalsin25/spe-major-front:latest"  
            }  
        }  
    }  
}
```

Docker Build Image: Builds the latest Docker image for the front-end development code.

```

        stage('Delete Docker Containers and Existing Images') {
    steps {
        script{
            // here we are checking if there are any containers running in our system if so then delete them.
            // def running_containers = sh (returnStdout: true, script: 'docker ps -q').trim()
            // if (running_containers) {
            //     sh 'docker rm -f $(docker ps -aq)'
            // }
            //
            // sh 'docker image rm -f sbrc1996/speminiproject'
            sh '''
                # Remove all images with the tag <none>
                docker rmi --force $(docker images | grep "<none>" | awk '{print $3}')
                '''
            }
        }
    }
}

```

Delete Existing Image: Clear the previous/old existing images so that only the latest image can be pushed to the dockerhub.

```

        ]
    stage('Push Docker Image') {
        steps {
            script{
                docker.withRegistry('', 'dockerhub'){
                    dockerimage.push()
                }
            }
        }
    }
}

```

Push Docker Image: Pushing the docker image to the docker hub which will be further pulled and then can be used with backend and database containers.

JenkinsFile for Backend and Database and Docker Compose:

This Jenkinsfile is a pipeline script that automates the build, testing, and deployment of a Django backend application using Docker. Here's a stage-wise explanation of the code:

```
1 pipeline {
2     environment{
3         dockerImage = ''
4         PYTHONPATH = "/home/subham/.local/lib/python3.10/site-packages:/var/lib/jenkins/workspace/Realest_Estate_Pipeline_Django/backend"
5         DB_NAME = "realest_estate"
6         DB_HOST = "127.0.0.1"
7         DB_PORT = "3306"
8         DB_USER = "root"
9         DB_PASSWORD = "subham123"
10    }
11    agent any
12 }
```

- environment: Sets up environment variables for the pipeline, including the Docker image name, PYTHONPATH, and database credentials. As Jenkins is unable to fetch the system env variables.
- agent any: Specifies that the pipeline can run on any available agent.

```
stages {
    stage('Git Pull') {
        steps {
            git 'https://github.com/testusername190/Realest_Estate_Backend.git'
        }
    }
    stage('Change Permissions and grant permission to create the log files..') {
        steps {
            sh 'chmod -R 777 backend/logs'
        }
    }
    stage('Install dependencies in the pipeline folder..') {
        steps {
            sh 'pip install django djangorestframework django-cors-headers djangorestframework-simplejwt Pillow mysqlclient'
        }
    }
    stage('Build + Test Cases') {
        steps {
            echo 'Build not required for our django project, we just need to install pip in our Docker file'
            echo 'Test cases written for Model Testing in Django!!'
            sh 'cd backend/ && python3 manage.py test contacts.tests'
            sh 'cd backend/ && python3 manage.py test listings.tests'
            sh 'cd backend/ && python3 manage.py test realtors.tests'
        }
    }
}
```

- stages: Divides the pipeline into different stages to perform specific tasks in sequence.
- Git Pull: Clones the source code from a GitHub repository.
- Change Permissions and grant permission to create the log files..: Changes the permissions of the log files to allow write access.
- Install dependencies in the pipeline folder..: Installs the required dependencies in the pipeline folder.
- Build + Test Cases: Runs the test cases for the Django models.

```

stage('Docker Build Image for Django Backend') {
    steps {
        script{
            // sh 'cd backend/'
            dockerImg=docker.build("sbrcl996/realest_estate:latest")
        }
    }
}
stage('Push Django Python Docker Image') {
    steps {
        script{
            docker.withRegistry('', 'dockerhub'){
                dockerImg.push()
            }
        }
    }
}
stage('Delete Docker Image') {
    steps {
        script{
            // sh 'docker image rm -f sbrcl996/speminiproject'
            sh '''
                # Remove all images with the tag <none>
                docker rmi --force $(docker images | grep "<none>" | awk '{print $3}')
            '''
        }
    }
}

```

- Docker Build Image for Django Backend: Builds a Docker image for the Django backend.
- Push Django Python Docker Image: Pushes the Docker image to a Docker registry (in this case, Docker Hub).
- Delete Docker Image: Removes any dangling Docker images with the tag <none>

```

stage('Pull Frontend Docker Image from the Docker Hub')
{
    steps
    {
        sh 'docker pull vishalsin25/spe-major-front'
    }
}

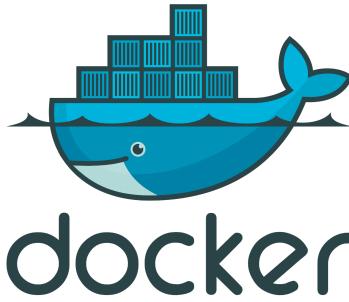
```

Pulling the frontend Image from the Dockerhub of Vishal that contains the image of node used to run our frontend react application.

```
1 stage('Ansible Deploy Stage') {
2     steps {
3         //Ansible Deploy to remote server
4         ansiblePlaybook colorized: true,
5                         disableHostKeyChecking: true,
6                         inventory: 'inventory',
7                         installation:'Ansible',
8                         playbook: 'playbook.yml',
9                         sudoUser: null
10    }
11 }
```

- The stage is called "Ansible Deploy Stage" and is defined using the stage block in the Jenkins pipeline syntax.
- The steps block defines the steps that should be executed as a part of this stage.
- The ansiblePlaybook step is used to execute an Ansible playbook on the remote server. The arguments passed to this step are:
- colorized: true - enables colorized output for the playbook execution.
- disableHostKeyChecking: true - disables the host key checking for SSH connections.
- inventory: 'inventory' - specifies the inventory file containing the list of hosts to which the playbook will be executed.
- installation:'Ansible' - specifies the installation directory for Ansible on the remote server.
- playbook: 'playbook.yml' - specifies the name of the Ansible playbook that should be executed on the remote server.
- sudoUser: null - specifies that the playbook should be executed as the root user on the remote server.

Overall, this Jenkinsfile sets up a continuous integration and continuous deployment (CI/CD) pipeline for a Django backend application, automating the build, testing, and deployment processes.



Containerization:

Docker is a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker.

Dockerfile for Backend:

```
1 # Creating the Base Image
2 FROM python:3.10.6-alpine3.15
3
4 # setup environment variable
5 ENV DockerHOME=/app
6
7 # set work directory
8 RUN mkdir -p $DockerHOME
9
10 # where your code lives
11 WORKDIR $DockerHOME
12
13 # set environment variables
14 ENV PYTHONDONTWRITETBYTECODE 1
15 ENV PYTHONUNBUFFERED 1
16
17
18 # install dependencies mysqlclient using mariadb connector and delete it after installation.
19 RUN apk update && \
20     apk add --no-cache mariadb-connector-c-dev && \
21     apk add --no-cache python3 python3-dev mariadb-dev build-base && \
22     pip3 install --no-cache-dir mysqlclient && \
23     apk del python3-dev mariadb-dev build-base
24
25 # checking the network adapters inside the docker container
26 RUN apk add netcat-openbsd
27
28 # copy whole project to your docker app directory.
29 COPY ./backend $DockerHOME
30
31 # run this command to install all dependencies
32 RUN pip3 install --no-cache-dir django djangorestframework django-cors-headers djangorestframework-simplejwt Pillow
33
34 # we create a new wait.sh file who makes our django app wait for the db to startup.
35
36 COPY ./backend/wait.sh $DockerHOME/wait.sh
37 RUN chmod +x $DockerHOME/wait.sh
38
39
```

[Give feedback](#)

1. The Dockerfile is used to create a Docker image of the Django project.
2. FROM python:3.10.6-alpine3.15: This line specifies the base image to be used, in this case, Python 3.10.6 with Alpine Linux version 3.15. Alpine Linux is a lightweight distribution that is popular in Docker containers.
3. ENV DockerHOME=/app: This line sets the environment variable DockerHOME to "/app". This will be the root directory for our Docker container.
4. RUN mkdir -p \$DockerHOME: This line creates a directory named DockerHOME, with the -p flag used to create the parent directories if they do not exist.
5. WORKDIR \$DockerHOME: This line sets the working directory for our container to DockerHOME.
6. ENV PYTHONDONTWRITEBYTECODE 1: This line sets the environment variable PYTHONDONTWRITEBYTECODE to 1, which prevents Python from writing .pyc files.
7. ENV PYTHONUNBUFFERED 1: This line sets the environment variable PYTHONUNBUFFERED to 1, which ensures that Python output is not buffered.
8. RUN apk update && \ apk add --no-cache mariadb-connector-c-dev && \ apk add --no-cache python3 python3-dev mariadb-dev build-base && \ pip3 install --no-cache-dir mysqlclient && \ apk del python3-dev mariadb-dev build-base: This section installs the dependencies required for our Django project, including MariaDB connector, Python 3, and the development libraries for MariaDB and Python. After the dependencies are installed, it removes the packages that are no longer needed.
9. RUN apk add netcat-openbsd: This line installs netcat-openbsd, which will be used to check network adapters inside the Docker container.
10. COPY ./backend \$DockerHOME: This line copies the entire backend directory to the Docker container's app directory.
11. RUN pip3 install --no-cache-dir django djangorestframework django-cors-headers djangorestframework-simplejwt Pillow: This line installs the required Python packages using pip.
12. COPY ./backend/wait.sh \$DockerHOME/wait.sh: This line copies the wait.sh script to the Docker container's app directory.
13. RUN chmod +x \$DockerHOME/wait.sh: This line makes the wait.sh script executable.
14. Finally, once the Dockerfile is built, we can use this image to create a Docker container that can run our Django project.

Dockerfile for Frontend:

```
#Telling Docker to use the node:17-alpine image as the base image
FROM node:12-alpine
# Layer 2: Telling Docker to create a directory called `/usr/src/app` in the container and set it as the working directory.
WORKDIR /frontend

# Layer 3: Copying the package.json file from the root of the project to the `app` directory in the container.
COPY ./package.json ./
COPY ./package-lock.json ./

# Layer 4: Installing the dependencies listed in the package.json file.
RUN npm i --force
# RUN NODE_ENV=development npm i --force

# Layer 5: Copying all the files from the root of the project to the `app` directory in the container.
COPY . .

# Layer 6: Telling Docker that the container will listen on port 3000.
EXPOSE 3000

# Layer 7: Telling Docker to run the `npm start` command when the container is started.
CMD [ "npm", "start" ]
```

1. FROM node:12-alpine: This instruction tells Docker to use the node:12-alpine image as the base image for our new image. This means that our new image will have all the features of the node:12-alpine image.
2. WORKDIR /frontend: This instruction tells Docker to create a new directory called frontend in the container and set it as the working directory.
3. COPY ./package.json ./ and COPY ./package-lock.json ./: These instructions copy the package.json and package-lock.json files from the root directory of the project into the frontend directory in the container.
4. RUN npm i --force: This instruction runs the npm i command in the frontend directory in the container, which installs all the dependencies listed in the package.json file.
5. COPY . .: This instruction copies all the files from the root directory of the project into the frontend directory in the container.
6. EXPOSE 3000: This instruction tells Docker that the container will listen on port 3000.
7. CMD ["npm", "start"]: This instruction tells Docker to run the npm start command when the container is started. This command will start the application on port 3000.

DockerHub Repo For Backend:

 sbrc1996 / realest_estate

Description
This repository does not have a description 

 Last pushed: a day ago

Docker commands
To push a new tag to this repository,
`docker push sbrc1996/realest_estate:tagname`

Tags
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	5 hours ago	a day ago

[See all](#) [Go to Advanced Image Management](#)

Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.
Available with Pro, Team and Business subscriptions. [Read more about automated builds](#).

DockerHub For Frontend:

 vishalsin25/spe-major-front 

By [vishalsin25](#) • Updated an hour ago
Docker image for SPE-major project frontend
[Image](#)

[Pulls 11](#)

[Overview](#) **Tags**

Sort by [Newest](#) [Filter Tags](#) 

TAG	DIGEST	OS/ARCH	SCANNED	COMPRESSED SIZE
latest	33e5f03c9afb	linux/amd64	---	107.34 MB

`docker pull vishalsin25/spe... `

Docker-Compose File:

```
1  version: "3"
2
3  services:
4    realest_estate_backend:
5      image: sbrcc1996/realest_estate
6      container_name: realest_estate_backend
7      environment:
8        - DB_HOST=172.16.128.119
9        - DB_PORT=30000
10       - DB_NAME=realest_estate
11       - DB_USER=root
12       - DB_PASSWORD=root
13     ports:
14       - "8000:8000"
15     depends_on:
16       - mydb
17     entrypoint: ["../wait.sh"]
18     volumes:
19       - realest_estate_logs:/app/logs    # Persistent Log storing.
20       - realest_estate_media:/app/media  # Persistent Media storage.
21
22 mydb:
23   image: mysql:8.0.21
24   ports:
25     - "30000:3306"
26   environment:
27     #These environment variables are to be not included in the Settings.py file of Django.
28     MYSQL_ROOT_PASSWORD: root
29     MYSQL_DATABASE: realest_estate
30   # This will create the database realest_estate here if it not exists.
31   volumes:
32     - mydb_data:/var/lib/mysql
33
34 realest_estate_frontend:
35   image: vishalsin25/spe-major-front
36   container_name: realest_estate_frontend
37   stdin_open: true
38   ports:
39     - "3000:3000"
40   depends_on:
41     - realest_estate_backend
42
43
44 volumes:
45   mydb_data:          # Creating a volume named mydb_data and store the data of the MYSQL running in the container by doing the mapping.
46   realest_estate_logs: # Creating a volume to store the logs persistently.
47   realest_estate_media: # Creating a volume to store the photos and other media files persistently.
```

wait.sh file:

```
15 lines (9 sloc) | 376 Bytes
1  #!/bin/sh
2
3  while ! nc -z mydb 3306 ; do
4    echo "Waiting for the MySQL Server"
5    sleep 3
6  done
7
8
9  python3 manage.py makemigrations
10 python3 manage.py migrate
11
12 DJANGO_SUPERUSER_USERNAME=adminbrc DJANGO_SUPERUSER_EMAIL=adminbrc@example.com DJANGO_SUPERUSER_PASSWORD=password123 python3 manage.py createsuperuser
13
14
15 python3 manage.py runserver 0.0.0.0:8000
```

Brief Explanation for the DockerCompose file and Wait.sh file:

1. **version: "3"**: This instruction specifies the version of Docker Compose being used in this file.

2. services: This is a list of the services (containers) that will be created by Docker Compose.
3. realest_estate_backend: This service is the backend container for the real estate application. It uses an existing Docker image from Docker Hub (sbrcc1996/realest_estate), sets the container name to realest_estate_backend, and configures environment variables for the container's database connection settings (DB_HOST, DB_PORT, DB_NAME, DB_USER, and DB_PASSWORD). It also maps the container's port 8000 to the host's port 8000, sets a dependency on the mydb service, runs a script before starting the container (entrypoint: ["./wait.sh"]), and defines two volumes for the container's logs and media files.
4. mydb: This service is the database container for MySQL. It uses the official MySQL Docker image (mysql:8.0.21), maps the container's port 3306 to the host's port 30000, sets environment variables for the MySQL root password and database name (MYSQL_ROOT_PASSWORD and MYSQL_DATABASE), and defines a volume for the container's data.
5. realest_estate_frontend: This service is the frontend container for the real estate application. It uses an existing Docker image from Docker Hub (vishalsin25/spe-major-front), sets the container name to realest_estate_frontend, maps the container's port 3000 to the host's port 3000, and sets a dependency on the realest_estate_backend service.
6. volumes: This is a list of the volumes that will be created by Docker Compose.
7. mydb_data: This volume is created to store the data of the MySQL container persistently.
8. realest_estate_logs and realest_estate_media: These volumes are created to store the logs and media files of the backend container persistently.
9. The wait.sh is a shell script. It checks if the MySQL Server is up and running on the mydb host at port 3306 using the nc command.
10. If the server is not up, it waits for 3 seconds and then checks again.

11. Once the server is up, it runs the Django migration commands to create the database schema.
12. It creates a superuser with the username adminbrc, email adminbrc@example.com, and password password123.
13. Finally, it starts the Django development server on 0.0.0.0:8000 which listens to all incoming requests.

Continuous Deployment:



Ansible is a suite of software tools that enables infrastructure as code. It is open-source and the suite includes software provisioning, configuration management, and application deployment functionality.

Inventory file:

```
ybook.yml  inventory  X  
inventory  
[ubuntu18]  
172.16.128.119 ansible_user="ansible_usr"
```

[ubuntu18] is a group name used to represent a group of hosts in Ansible. Here, it is used to group a single host with the specified IP address 172.16.128.119 (which is my IP Address)

172.16.128.119 is the IP address of the host that Ansible will operate on.

ansible_user="ansible_usr" is an optional parameter that specifies the username that Ansible should use to connect to the host via SSH. Here, the username is set to ansible_usr.

Playbook file:

```
playbook.yml ×
- playbook.yml > {} o > [ ]tasks > {} 2 > command
  Ansible Playbook - Ansible playbook files (ansible.json)
1  ---
2  - name: copy docker-compose file
3    hosts: all
4    tasks:
5      - name: copy docker-compose file
6        copy:
7          src: ./docker_compose_spe.yaml
8          dest: ./
9      - name: docker-compose down
10        command: docker-compose -f docker_compose_spe.yaml down
11      - name: run docker-compose file
12        command: docker-compose -f docker_compose_spe.yaml up -d
```

This ansible playbook.yml contains three tasks that will be executed on all hosts specified in the inventory file.

- The first task is to copy the docker_compose_spe.yaml from the local machine to the target host. The copy module is used for this task, which takes src and dest arguments to specify the source and destination file paths.
- The second task is to stop any running containers and remove the network, volumes, and images associated with the containers using the docker-compose down command. This command is executed using the command module.

- The third task is to start the containers defined in the docker_compose_spe.yaml file in detached mode using the docker-compose up -d command. Again, this command is executed using the command module.

The application will start running but as a background process, to access the application we need to run localhost:8000 or localhost:3000. But to run the individual container with their assigned ip we need to do the following:

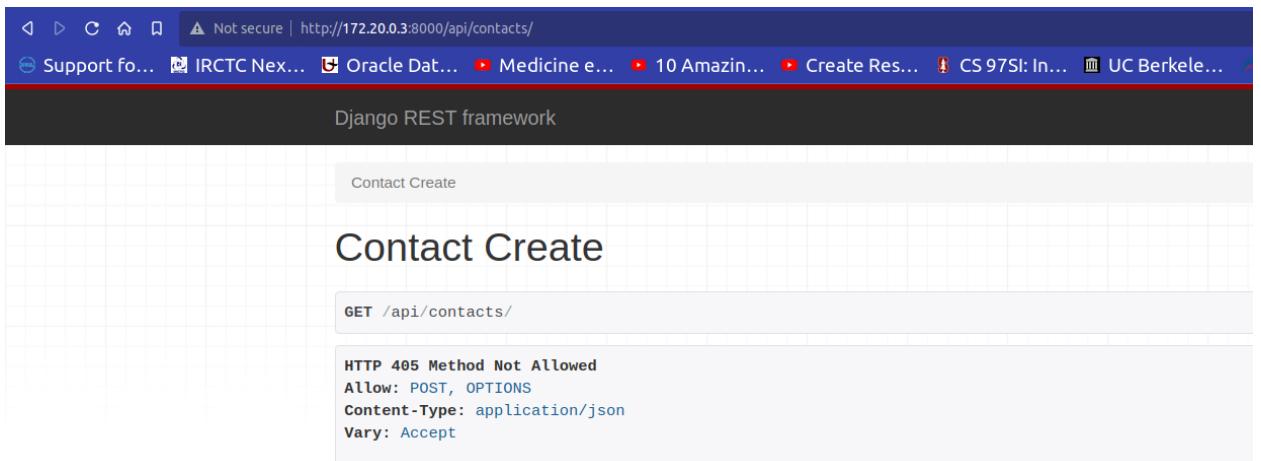
- To access the application through localhost:8000, you need to first identify the IP address of the Docker container.
- You can use the **docker ps** command to get a list of running containers. Look for the container that is running your Django application and note its container ID.
- Then run this command:

```
docker inspect <container ID> | grep IPAddress
```

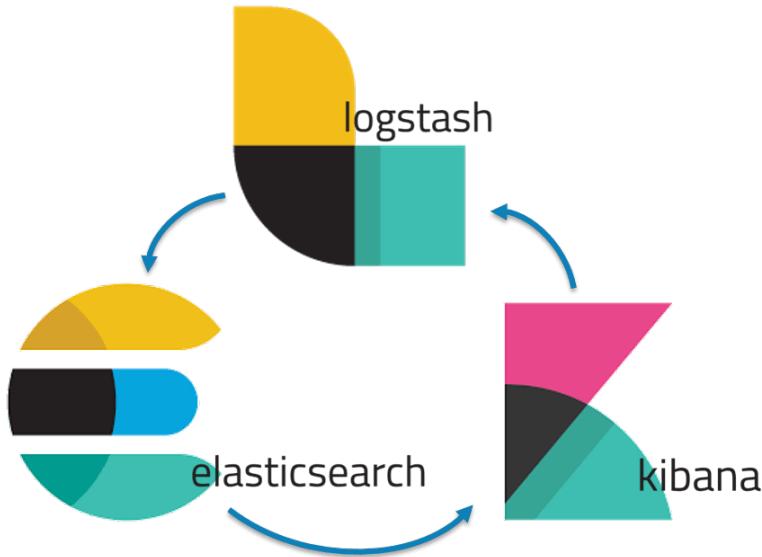
```
subham@subham-HP-Laptop-15s-fq2xxx:~$ docker inspect 0c | grep IPAddress
    "SecondaryIPAddresses": null,
    "IPAddress": "",
        "IPAddress": "172.19.0.3",

```

- Now run the application in the Browser using the command <http://<container IP address>:8000> ,
For example : <http://172.20.0.3:8000>



CONTINOUS MONITORING:



First we fetch our log files from inside the container where our backend application is running:

```
docker exec -it c1 /bin/sh
```

```
subham@subham-HP-Laptop-15s-fq2xxx:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
c1d967b36cb9 sbrc1996/realest_estate "/watt.sh" 16 minutes ago Up 16 minutes 0.0.0.0:8000->8000/tcp, :::8000->8000/
b5be7e4703dd mysql:8.0.21 "docker-entrypoint.s..." 16 minutes ago Up 16 minutes 3306/tcp, 0.0.0.0:30000->3306/tcp, ::

subham@subham-HP-Laptop-15s-fq2xxx:~$ docker exec -it c1 /bin/sh
/app # ls
accounts      contacts      listings      logs      manage.py      media      realest_estate      realtors      wait.sh
/app # cd logs
/app/logs # ls
```

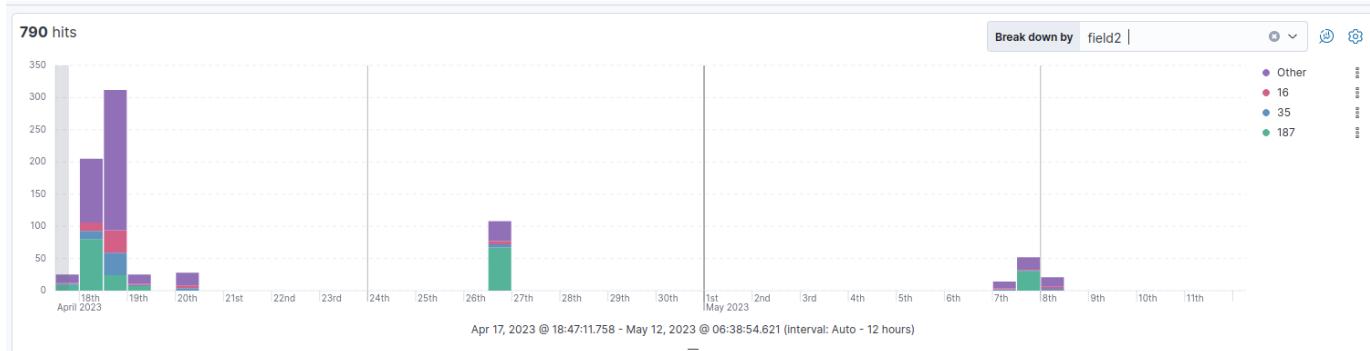
Now see where the log files are then fetch it and place it in your local using this command.

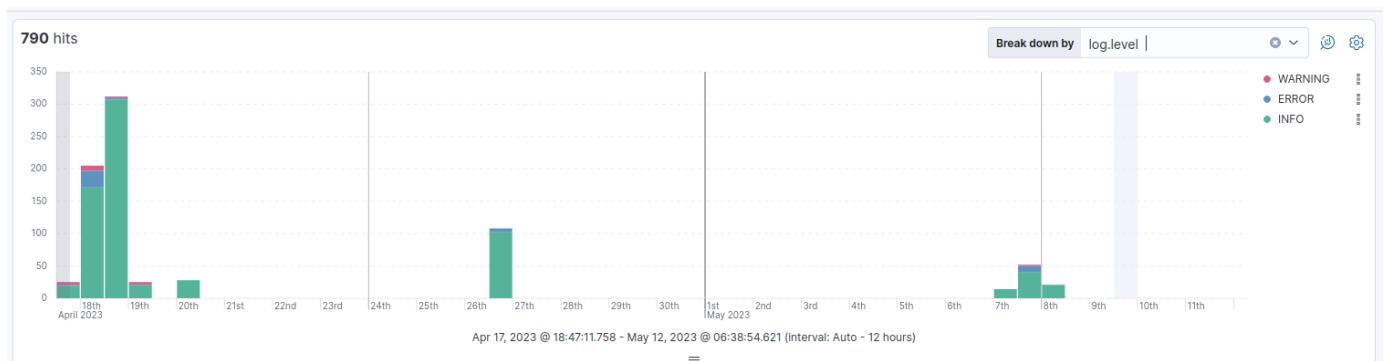
```
docker cp <container_name>:<path_to_file> <host_path>
```

```
docker cp realest_estate_backend:/app/logs/realest_estate_info.log /home/subham/Desktop/
```

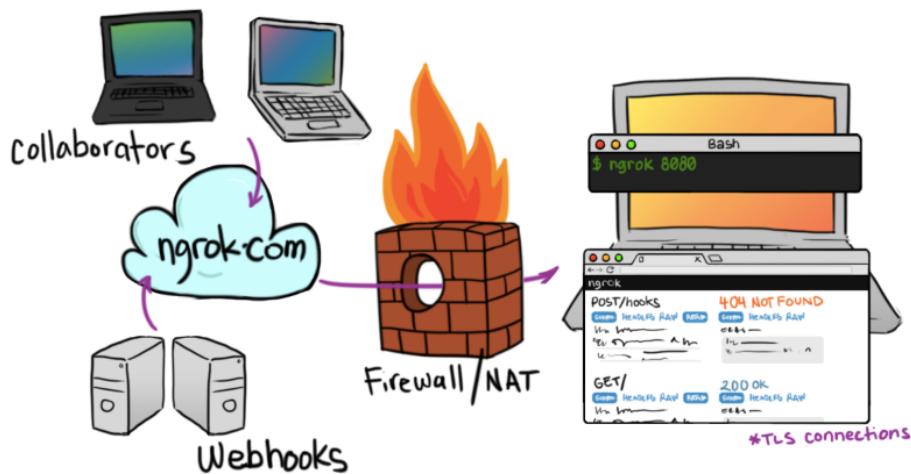
Now we have our log with us.. We need to upload it to the ELK stack. But for the sake of simplicity, we will upload it manually in Kibana or we could use the Filebeat to upload it.

Below are the Kibana visualizations for the log file, based on different fields:-





Ngrok:-



Ngrok is a tool that allows you to expose a local server to the internet by creating a secure tunnel. It provides a public URL that redirects incoming

requests to your local server, enabling you to test and share your web applications without deploying them to a public server.

Ngrok can be used to test webhooks locally by creating a tunnel to your local development server, allowing you to receive webhook payloads from third-party services like GitHub.

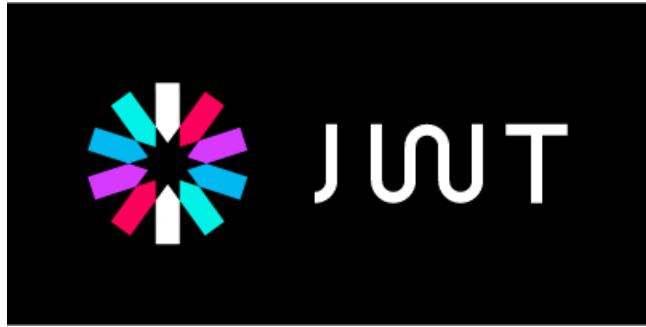
To start ngrok in terminal give command: `ngrok http 8080`

Then ngrok will provide a public URL to receive payload from Github. So, as soon as any changes are made in github it will deliver payload to Jenkins and Jenkins will trigger the new build for the payload received.

```
ngrok by @inconshreveable
Session Status           online
Account                  Vishal Singh (Plan: Free)
Version                 2.3.41
Region                  United States (us)
Web Interface            http://127.0.0.1:4040
Forwarding              http://72a2-119-161-98-68.ngrok-free.app -> http://localhost:8080
                         https://72a2-119-161-98-68.ngrok-free.app -> http://localhost:8080

Connections             ttl     opn      rti      rt5      p50      p90
                         0       1       0.00    0.00    0.00    0.00

HTTP Requests
-----
POST /github-webhook/      200 OK
```



SECURITY:

For security we have used JWT or JSON Web Tokens. JWT (JSON Web Token) is a form of transmitting a JSON object as information between parties. Let's learn more about what JWTs are and how they work.

We use JWTs to exchange information in cases where they are signed – for example using public-private key pairs to make sure that the integrity of the information is not compromised since the payload and header are used to compute signatures.

Working of JWT:

You can get a JWT through logging in with a username and password. In exchange the server returns an access and refresh token in the form of a JWT. The tokens access resources on the server.

The lifetimes of the access and refresh tokens vary since access tokens last for five minutes or less while the refresh tokens can last for 24 hours. But you can customize the timelines of both types of tokens.

Parts of JWT:

```
header.payload.signature
```

Illustration of a JWT in simple English

```
header = eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9F5  
payload = eyJ0b2tlbl90eXBlijoiYWNjZXNzIiwiZXhwIjoxNTQzODI4NDMxLCJqdGkiOiI3ZjU5OTdiNzE1MGQ0NjU3OWRjMm  
signature = Ju70kdcaHKn1Qaz8H42zr0Yk0Jx9kIciuhkTn9Xx7vhikY
```

More details here:

<https://www.freecodecamp.org/news/how-to-use-jwt-and-django-rest-framework-to-get-tokens/>



POSTMAN

API TESTING:

We have a total of 9 api in our project, during the development our backend and our frontend code was developed separately.

Hence while developing the backend to test if the APIs were indeed working we used POSTMAN. Here are some of the screenshots to understand it better:

The first folder was for developing the application in my locally.

The second folder was for testing if the APIs are working in the docker container.

My Workspace

- Collections
- +
- > Realest Estate
- > Realest Estate Docker

APIs

Environments

Mock Servers

Monitors

Flows

Realest Estate

- Accounts
 - POST Signup
 - POST Signin
- Realtors
 - GET Get All Realtors
 - GET Get a particular Realtor by ID
 - GET Top Selling Realtors
- Listings
 - GET Get All Listings
 - GET Get Listing By Slug
 - POST Get Listings By Search
- Contacts
 - POST Send Email

Realest Estate Docker

- Accounts_Docker
 - POST SignUp_Docker
 - POST SignIn_Docker
- Contacts_Docker
 - POST Send Email Docker

PROBLEMS FACED:

Problems faced by me while creating this project were many.

1. The first major issue faced by me was the library `djangorestframework-simplejwt` was not compatible with the django virtual environment, hence I had to proceed without using the virtual environment.
2. Sending email through Google API and using the SMTP port 527 was very much new to me hence I had to do a lot of research in coming up with the solution.
3. The docker daemon process does not have the sufficient permissions hence we need to manually add these permissions manually on every restart using the cmd: **`sudo chmod 666 /var/run/docker.sock`**
4. The django settings.py was provided the required environment variables in docker container to get the required details for a successful connection to the MYSQL database running in another container. But when I ran that same code in local it was failed due to absence of those env variables. Hence I created those variables in my local `bashrc` file. In these steps:

- **`nano ~/.bashrc`**

```
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This

# Creating Database Environment Variable for our Realeast_estate Project.
export DB_NAME="realeast_estate"
export DB_HOST="127.0.0.1"
export DB_PORT=3306
export DB_USER="root"
export DB_PASSWORD="subham123"
```

- **`source ~/.bashrc`**
- To apply the changes, either open a new terminal window or run the following command in the current terminal:

5. Even after creating these variables our Jenkinsfile was unable to fetch the environment variables from the `bashrc` file, hence we had to manually declare the environment variables in the Jenkinsfile to solve this issue.

6. Jenkins was also unable to read the environment variable of the python path, strangely on running the test scripts manually in the directory /var/lib/jenkins/workspace/<curr_wsk> the test cases ran perfectly but on running them through the Jenkinsfile was not possible hence I had to manually install all the dependencies in the Jenkins workspace to ensure smooth testing.

REFERENCES:

1. Dockerize a Django App:
<https://blog.logrocket.com/dockerizing-django-app/>
2. Dockerfile for Django and React Application and Running it with the DockerCompose: <https://www.honeybadger.io/blog/docker-django-react/>
3. Logging in Django:
<https://www.freecodecamp.org/news/logging-in-python-debug-your-django-projects/>
4. Dockerizing Django and MySQL using Docker Compose:
<https://roytuts.com/docker-compose-dockerizing-django-mysql-app/>
5. GitHub Error resolution:
<https://www.a2hosting.com/kb/developer-corner/version-control-systems1/403-forbidden-error-message-when-you-try-to-push-to-a-github-repository/>
6. JWT:
<https://www.freecodecamp.org/news/how-to-use-jwt-and-django-rest-framework-to-get-tokens/>
7. Git and GitHub:
<https://www.freecodecamp.org/news/git-and-github-for-beginners/>
8. Docker and DockerHub:
<https://www.freecodecamp.org/news/the-docker-handbook/>
9. React:
<https://legacy.reactjs.org/docs/getting-started.html>
10. ChatGPT