Project Report

DOCTOR APPOINTMENT SYSTEM USING DJANGO, ANGULAR AND PYTHON

MODULE TITLE: WEB DEVELOPMENT FOR INFORMATION SYSTEMS

MODULE CODE: B9IS109

GROUP MEMBERS:

SINGUPURAM NAVIN KUMAR | 10612366

MODULE LEADER: DR OBINNA IZIMA

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

A doctor appointment booking system is an online tool that enables patients to conveniently schedule an appointment at a specific clinic for a health concern. Examples include dental, injury, mental health, muscle strain, obesity, arthritis, and asthma. Around 95% of the world's population suffers from multiple diseases. Several of those schedule visits with a neighboring doctor each week in order to receive the treatment. Also, you can benefit greatly from a doctor appointment booking system on your clinic's or doctor's website. It's a booking form with various fields that the patient must complete and submit in order for it to be considered an appointment. The patient and you both save a ton of time over this entire procedure. Also, you can quite effectively manage the many components of the firm using this.

The following list of advantages of a doctor booking system contains a variety of them.

- Time and money savings.
- Saves resources.
- Makes it easy to manage all the appointments and bookings.
- Proper following of the guidelines in times of health crisis.
- No rush hours and better sessions with the doctor.
- Better flow of income.
- Income from pre-bookings.
- Convenient for patients to easily book appointments.
- And much more.

HOW TO PROPERLY USE THE DOCTOR APPOINTMENT BOOKING SYSTEM

The website's front page is the best place for a medical appointment booking tool. At the top of the time is the form. So that the sick person may quickly arrange a doctor appointment without having to look through the internet once they arrive there.

Other health services pages, the footer, and the contact page on the doctor's website are additional areas you would want to take into account for adding an online doctor booking form.

Since many users have a tendency to open the contact us page right away after entering a website. Hence, failing to consider that would be a grave error in site design.

2. RESEARCH AND PLANNING

A system that will be user-friendly and solve a user's immediate problem was designed and built throughout a vital phase of research and planning. So, it was determined to create a system for scheduling medical appointments. Once the application was selected, the database design, framework selection, and entity design were all made by the requirements. I chose to use the open-source, simple-to-configure **SQL Lite 3 database**, which can be scaled as necessary. The framework was finished after the database was constructed. The **Django Python framework** was used to begin the development process because it offers the greatest degree of flexibility. Additionally, the **Angular Web Development Framework** was used to simplify the development of an application while gaining the advantages of using **Bootstrap** and other front-end development technologies. To streamline the development process, the app's screens, features, and routes were chosen before installation, setup, and actual work began.

3. CHOICE OF FRAMEWORK AND TECHNOLOGIES

3.1. PYTHON

The parts of a website or program that visitors don't see—the back end—are typically made with Python. For transferring data to and from servers, processing data, interacting with databases, routing URLs, and guaranteeing security, Python can be used in web development. Python offers several frameworks for web development. The application was created using the Python Django framework for the back end and the Angular Framework for the front end, both of which use the most recent version of Python, 3.11.2. For transferring data to and from servers, processing data, interacting with databases, routing URLs, and guaranteeing security, Python can be used in web development.

3.2. DJANGO FRAMEWORK

Model-View-Controller (MVC) architecture is used by the high-level Python web framework Django. By offering built-in components and tools for typical web development activities, it is intended to assist developers in creating online applications more rapidly and with less code.

Django has several important features, including:

Database interaction with ORM (Object-Relational Mapping)

- Admin interface built-in for data management and user authentication
- HTTP requests and responses are handled via URL routing and view management.
- engine for rendering HTML pages templates
- processing and validation of forms
- Cross-site scripting (XSS) and cross-site request forgery (CSRF) protection are examples of security features

3.3. SQL LITE 3

SQLite is one of the most well-liked and approachable relational database systems. It excels above other relational databases in many ways. The benefit of using SQLite3 is that it is simpler to set up and use, and the completed database is only one file that can be sent through email or kept on a USB memory stick.

The feature that makes it desirable are:

- The program is open-source.
- It works well on all platforms.
- Simple work with several sessions.

3.4. Angular

Utilizing Angular, one of the most widely used frameworks, for the frontend application. Angular gives developers the resources they need to create and organize complex JavaScript applications. In addition, Angular has several significant benefits over some rivals. For instance, Google employees created and maintain Angular. Along with these engineers, there is a sizable community available to assist you with problems as they arise. Angular is a code library that aims to make it easier to pair Django with AngularJS on the front end.

3.5. Bootstrap Templates

The most widely used CSS framework for creating responsive and mobile-first websites is Bootstrap. The templates are designed to allow for easy

customization to meet the needs. It is very easy to combine and use many templates.

3.6. AWS

An online service called Amazon Elastic Compute Cloud (Amazon EC2) offers safe, scalable processing capability in the cloud. It is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes.

4. UX DESIGN

4.1 FLOW DIAGRAM

This Doctor Appointment System involves two parties: the admin, which makes use of the Django Framework, and the user. Admin will have the right to interact with the functionalities to add data to the application such as specialties accounted.

Let's understand the flow:

User flow:

- Register to the Doctor Appointment Management System by adding a username and password.
- Log in with the Credentials
- Create a new appointment using the application by adding Full name, email date, phone, timing, and the specialist.
- Newly created appointments would be there in the view status and also an option to delete when needed.
- View Contact information on the website
- View About us, our services, and about team information on the website

Admin Django Flow:

- Sign up for the Django administration as a super user to manage appointments, authentication, and authorization.

- Enter the credentials to log in.
- Access appointments, physician, and specialty histories.
- View the history of groups and users as part of the authentication, and authorization.
- Add doctors and their specialties via the admin panel.

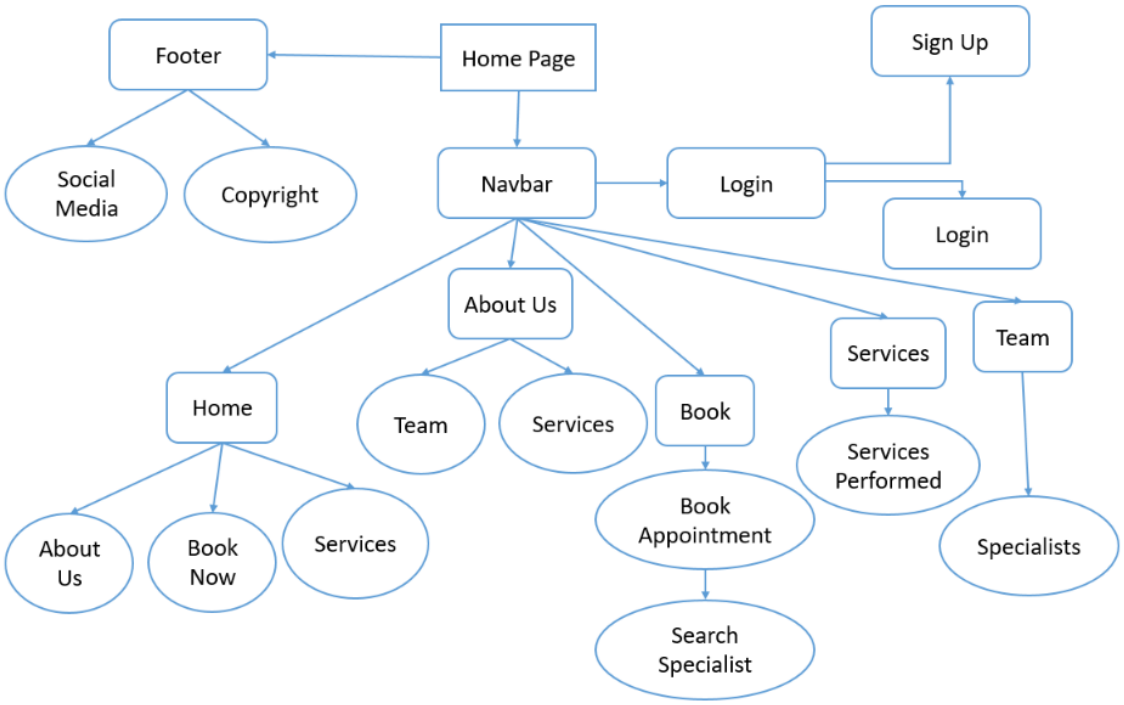


Figure 1: Doctor Appointment Flow Diagram

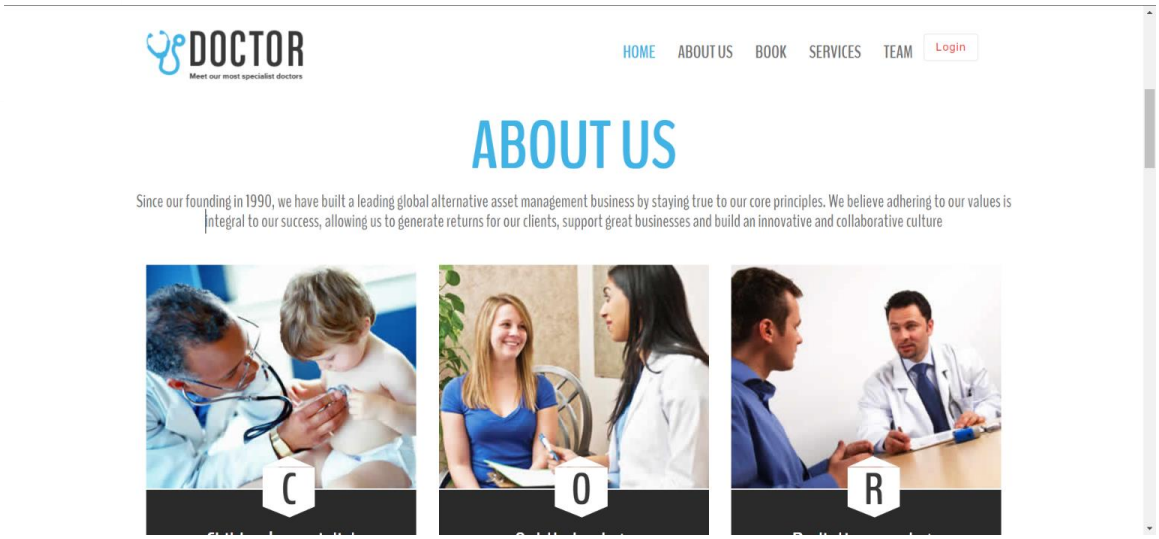
4.2 HOME PAGE

The homepage consists of six sections as it is a single-page application. The user has options for navigating around the home page, about us, book, services, and team using the Navbar. The user can also book an appointment from the book now section by providing details such as name, details, contact number, time slot needed, and specialist required.

Figure 2: Home Page

4.3 ABOUT US PAGE

This section consists of the details of the doctors and the specialists provided in the application such as the Children's specialist, ophthalmology, radiation oncology, etc. Also, the description of the different specialists providing services.



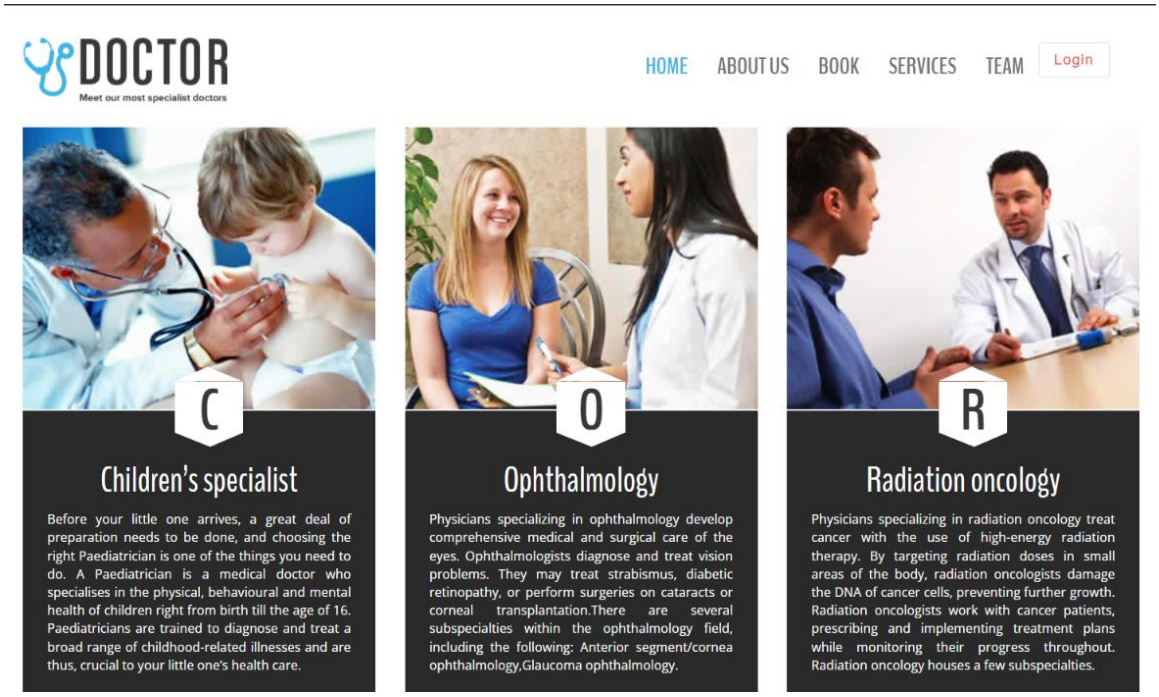


Figure 3: About Us

4.4 BOOK AN APPOINTMENT

This section consists of a few fields to punch in and book an appointment on a specific day and time with a specialist.

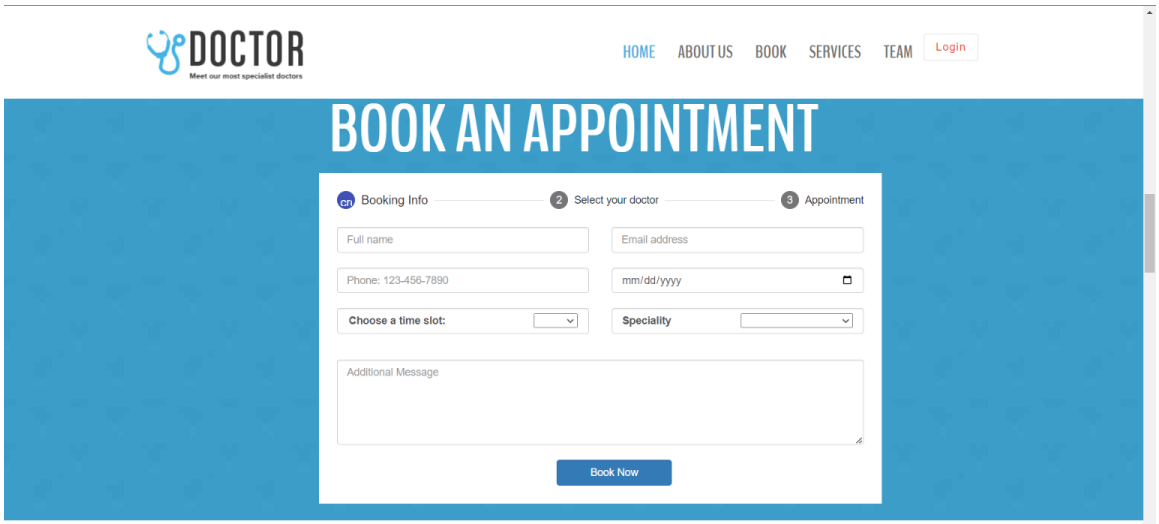


Figure 4: Appointment Booking

4.5 SERVICES

This section consists of different services provided by the particular hospital and can use the services wisely with an appointment.

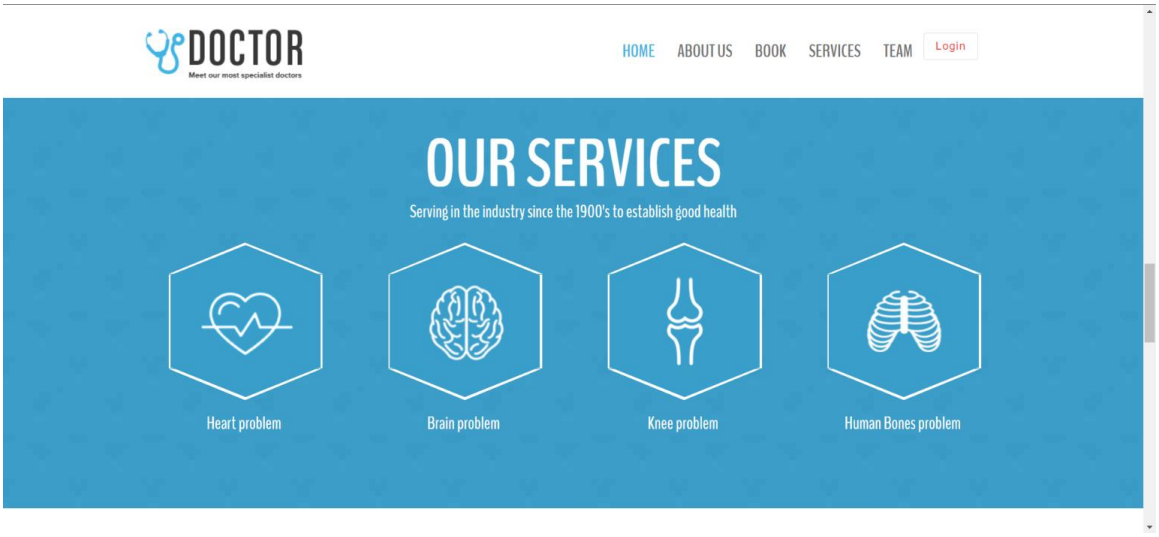


Figure 5: Services

4.6 TEAM

This section consists of a team of doctors and specialists working as part of the hospitality and on different diseases.

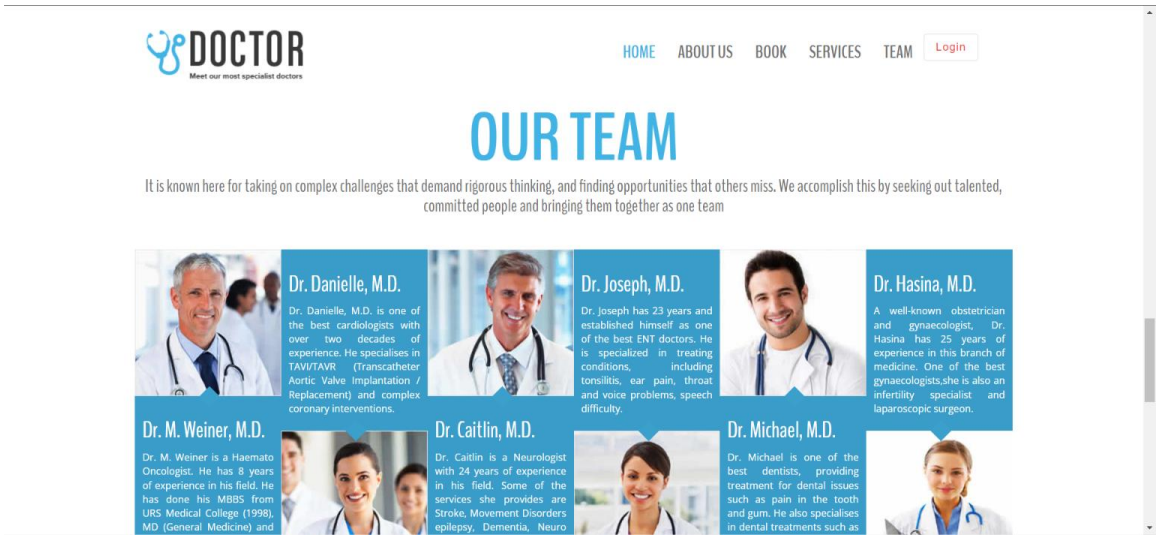


Figure 6: Team

4.7 CONTACT US

This section has the contact details for any inquiry and other details

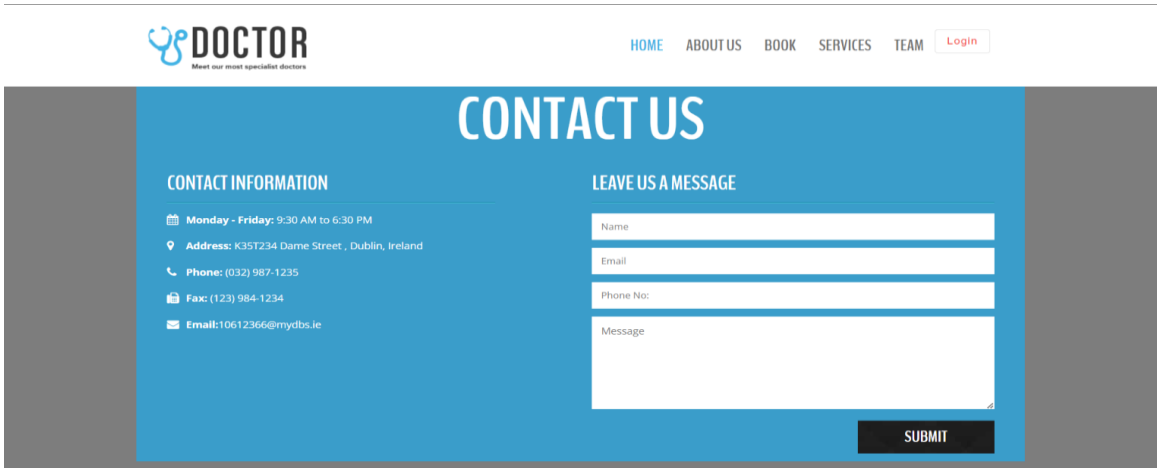


Figure 7: Contact Us

4.8 FOOTER

The footer section consists of the social media and the copyrights reserved.

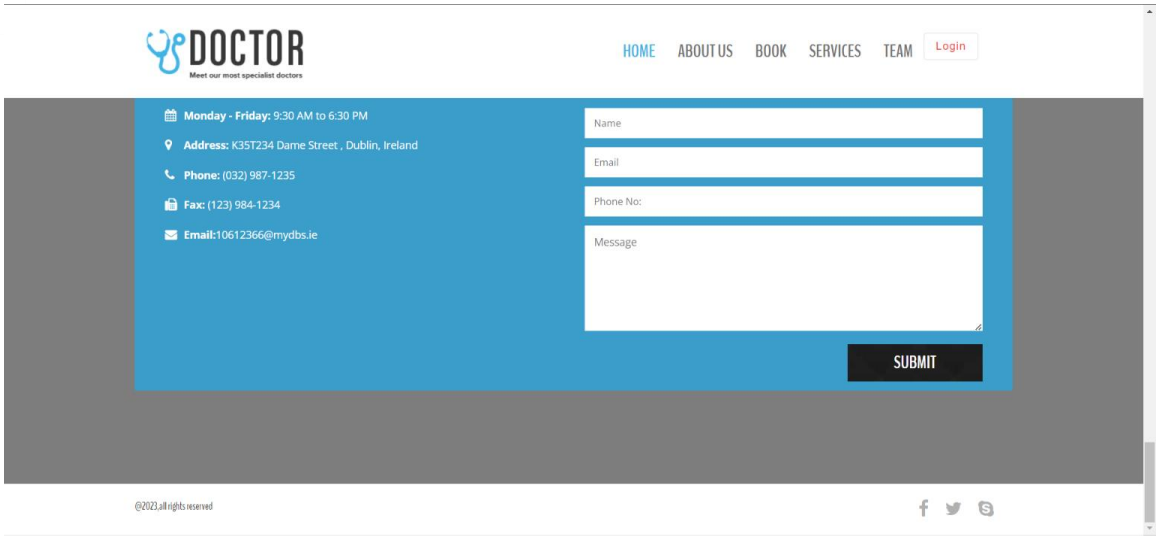


Figure 8: Footer

4.9 SIGNUP

The footer section consists of the signup username and password to get into the application.

Figure 9: SignUp Page

4.10 LOGIN

Users can be able to login with the registered details in the below screen.

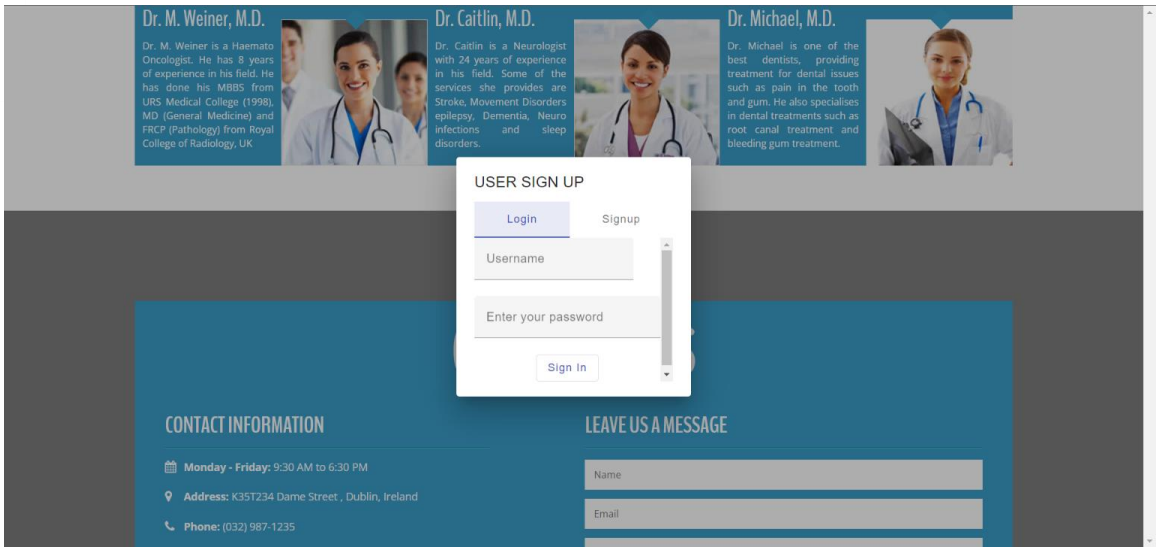


Figure 10: Login Modal Popup

4.11 SEARCH PAGE

The section consists of the search option to select any doctor to search on the availability.

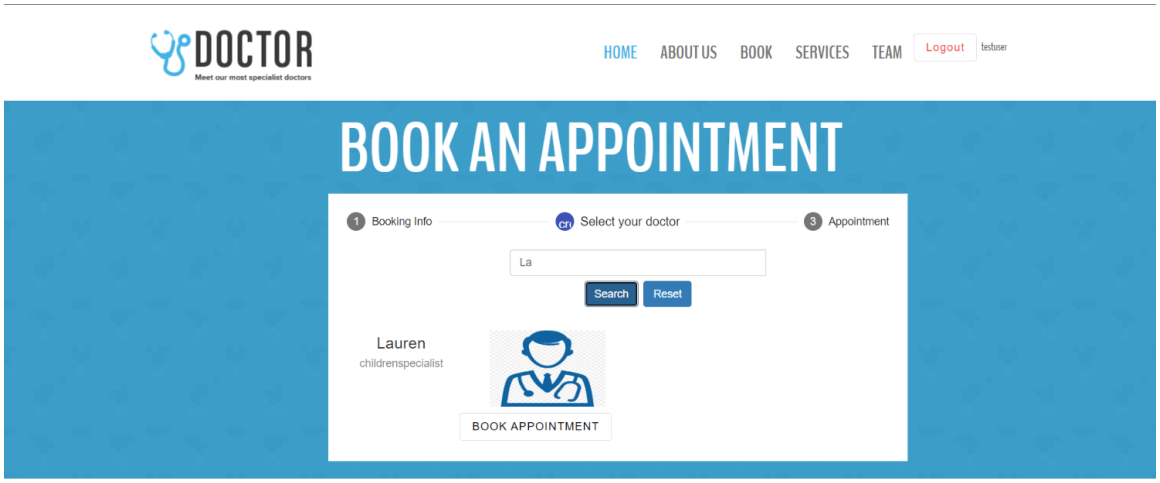


Figure 11: Search

5. BACKEND

5.1 DATABASE SETTING

As the below screenshot shows the engine used was a sqllite3 for the database. In the settings.py file, the database configuration has been developed.

```
# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Figure 12: SQL Lite 3 setting in the py setting file

Below is the XML file of the connection of the database which is the sql lite 3

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="dataSourceStorageLocal">
    <data-source name="db.sqlite3" uuid="568d9c2b-72a2-4ea8-a9d3-47234d342c48">
      <database-info product="SQLite" version="3.31.1" jdbc-version="2.1" driver-name="SQLite JDBC" driver-version="3.31.1"
      dbms="SQLITE" exact-version="3.31.1" exact-driver-version="3.31">
        <identifier-quote-string>&quot;</identifier-quote-string>
      </database-info>
      <case-sensitivity plain-identifiers="mixed" quoted-identifiers="mixed" />
      <secret-storage>master_key</secret-storage>
      <auth-required>false</auth-required>
      <schema-mapping>
        <introspection-scope>
          <node kind="schema" qname="@" />
        </introspection-scope>
      </schema-mapping>
    </data-source>
  </component>
</project>
```

Figure 13: Database XML File

5.2 LINUX SETTING

Created the Ubuntu Linux to run the code through the vagrant file

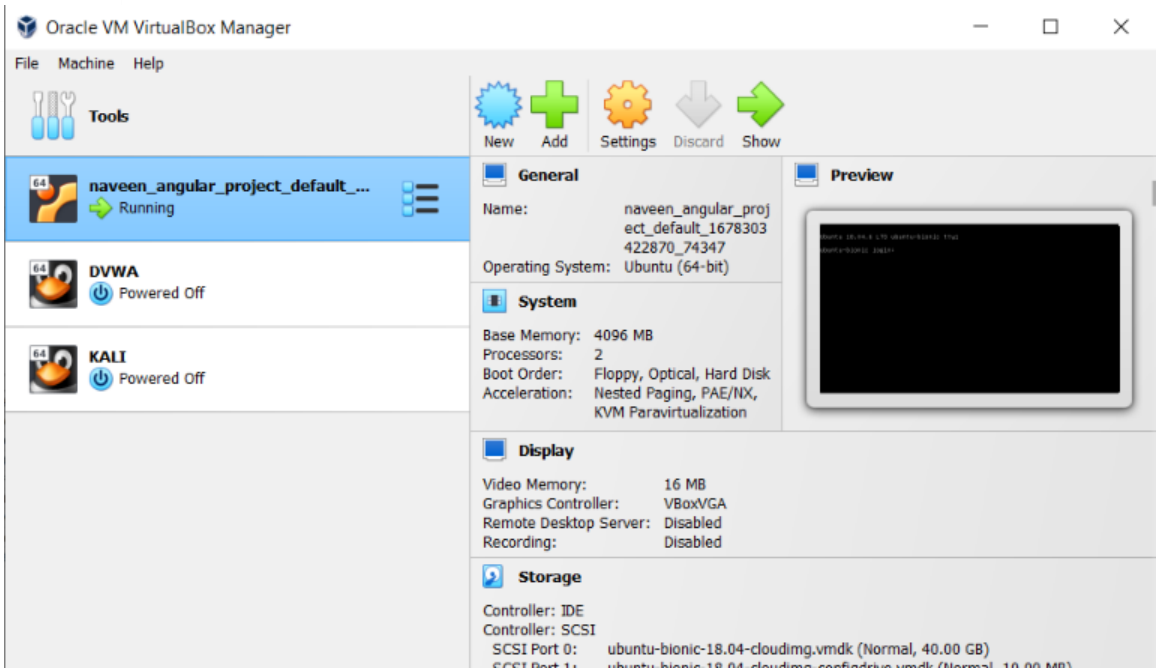


Figure 14: Linux Virtual Box

5.3 VAGRANT FILE

The below vagrant file is used to image the actual Windows angular, and python code to the Linux Ubuntu and then can be used properly during the deployment of the Amazon EC2.

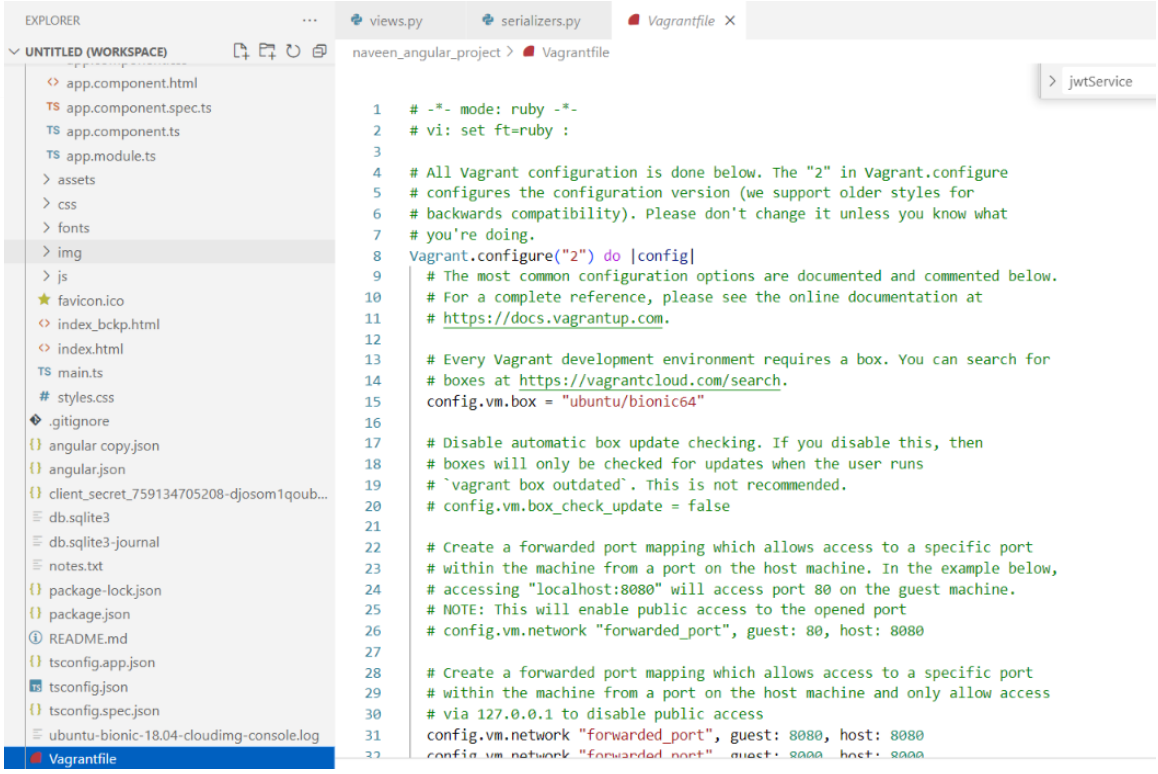


Figure 15: Vagrant File for Imaging

5.4 DJANGO ADMIN PAGE

In this, data such as the users, specialists, and doctors can be viewed, edited, and deleted as necessary.

Below is the screenshot from the deployed version of the Django

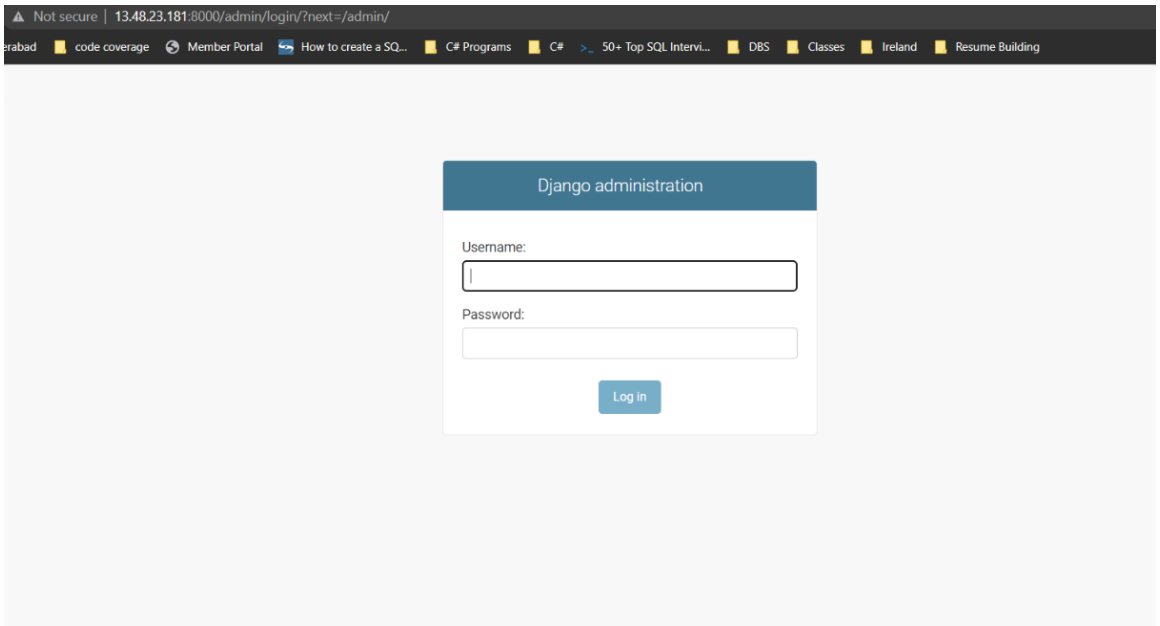


Figure 16: Admin Login Page.

Use the provided link to access the admin page - "http://13.48.23.181:8000/admin/login/?next=/admin/". The login credentials for local SQLLite3 are the same as the username and password.

SuperUser Credentials:

Username: admin, **Password**: password

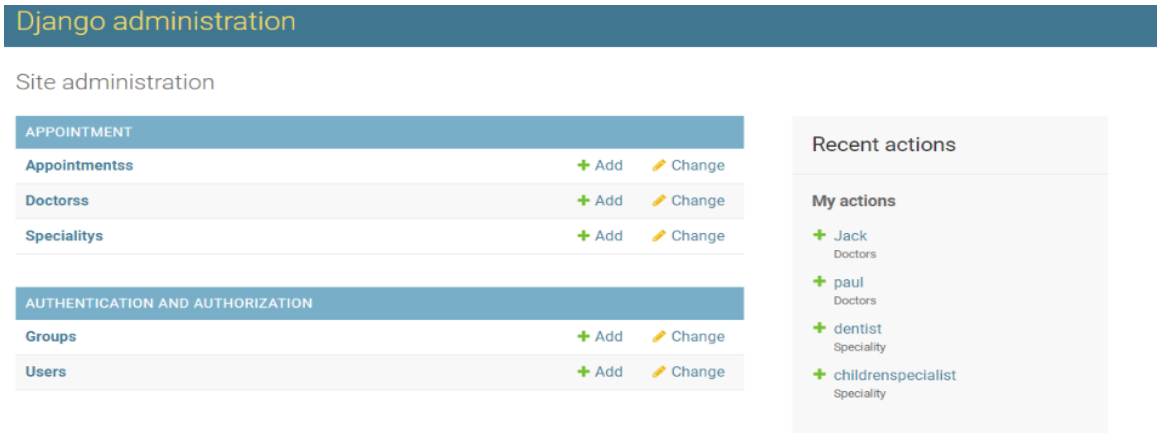


Figure 17: Django Administration

The administrator can view all the data and add any new data that is required for the web application on this page.

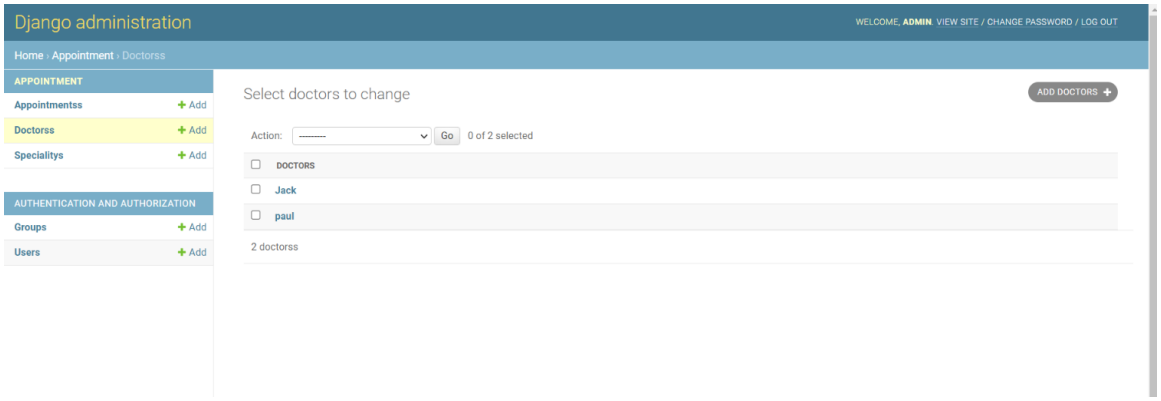


Figure 18: Adding the Doctors

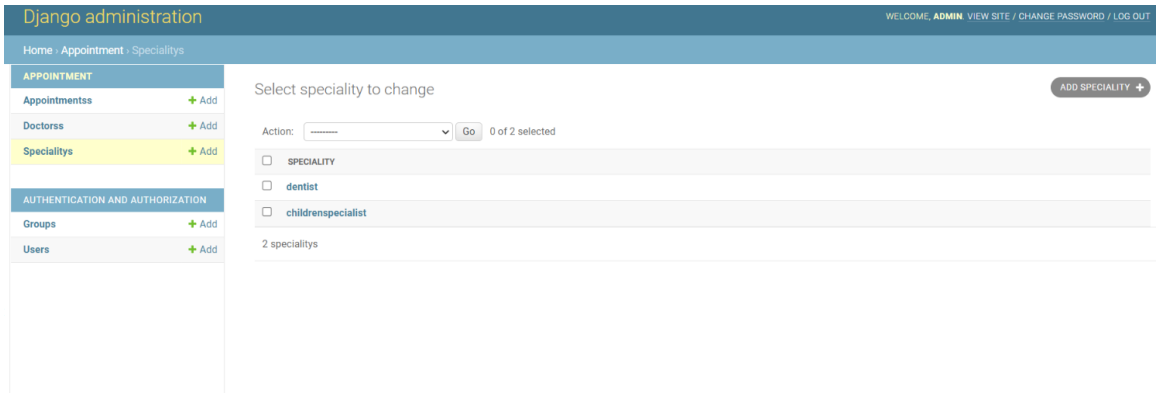


Figure 19: Adding the Specialists

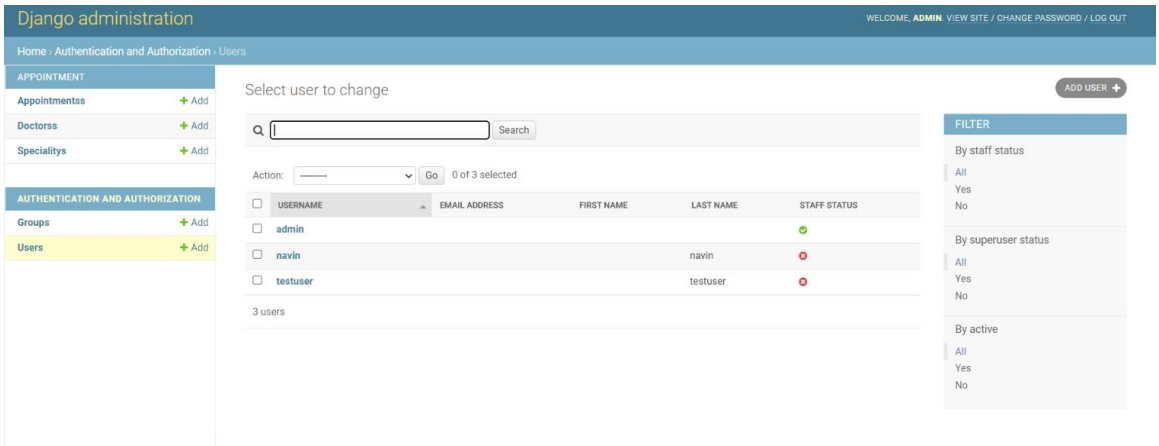


Figure 20: Adding the Users

5.5 DJANGO SETTINGS

The logic below states the Django settings in the manage.py in the visual code

views.py | manage.py | TS login-modal.component.ts 9+ | serializers.py

naveen_angular_project > backend > manage.py

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'doctor_backend.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

views.py | wsgi.py | login-modal.component.ts 9+ | serializers.py

naveen_angular_project > backend > doctor_backend > wsgi.py

```python
"""
WSGI config for doctor_backend project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.1/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'doctor_backend.settings')

application = get_wsgi_application()

```

Figure 21: Django Settings Module

views.py | settings.py × | TS login-modal.component.ts 9+ | serializers.py

naveen_angular_project > backend > doctor_backend > settings.py

```
21  # See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/
22
23  # SECURITY WARNING: keep the secret key used in production secret!
24  SECRET_KEY = 'django-insecure-k6v956he-=)10)#w&s5*19xal)5(7_%a6fh38$xw-&$@_&x^8n'
25
26  # SECURITY WARNING: don't run with debug turned on in production!
27  DEBUG = True
28
29  ALLOWED_HOSTS = ['*']
30
31
32  # Application definition
33
34  INSTALLED_APPS = [
35      'appointment',
36      'corsheaders',
37      'django.contrib.admin',
38      'django.contrib.auth',
39      'django.contrib.contenttypes',
40      'django.contrib.sessions',
41      'django.contrib.messages',
42      'django.contrib.staticfiles',
43      'rest_framework',
44  ]
45
46  MIDDLEWARE = [
47      'corsheaders.middleware.CorsMiddleware',
48      'django.middleware.security.SecurityMiddleware',
49      'django.contrib.sessions.middleware.SessionMiddleware',
50      'django.middleware.common.CommonMiddleware',
51      'django.middleware.csrf.CsrfViewMiddleware',
52      'django.contrib.auth.middleware.AuthenticationMiddleware',
53      'django.contrib.messages.middleware.MessageMiddleware',
```

Figure 22: Django Settings

```
import django.db.models.deletion


class Migration(migrations.Migration):

    initial = True

    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]

    operations = [
        migrations.CreateModel(
            name='speciality',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('speciality_name', models.CharField(max_length=100)),
            ],
        ),
        migrations.CreateModel(
            name='doctors',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('doctor_name', models.CharField(max_length=100)),
                ('doctor_speciality', models.ForeignKey(default='general', on_delete=django.db.models.deletion.SET_DEFAULT, t
            ],
        ),
        migrations.CreateModel(
            name='appointments',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('appointments_date', models.DateField()),
                ('appointments_time', models.CharField(max_length=10)),
```

MS 11 OUTPUT DEBUG CONSOLE TERMINAL

Figure 23: Migrations

6. WEB SERVICES

There is a specific route linked with each screen. Let's examine the APIS list.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/sign_up/', SignUpView.as_view(), name='sign_up'),
    path('api/log_in/', LogInView.as_view(), name='log_in'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/test/', HelloView.as_view()),
    path('api/validatetoken/', ValidateToken.as_view()),
    path('api/createappointment/', CreateAppointment.as_view()),
    path('api/listdoctors/', ListDoctors.as_view()),
    path('api/deleteappointment/', Deleteappointment.as_view()),
    path('api/getappointment/', GetAppontmentDetails.as_view()),
    path('api/google/', GoogleView.as_view()),
]
```

Figure 24: API Services URL

6.1 SIGNUP SCREEN

Users can register themselves on the sign up page so they can access the website for scheduling a doctor's appointment.

The system ensures that the password and valid email id are created when the user creates the credentials. The password is saved in an encrypted format during storage to ensure that the user's security is not compromised and that no one else can see the password.

```
signup(payload: signupModel){
  var formData: any = new FormData();
  formData.append('username', payload['username']);
  formData.append('password1', payload['passwordone']);
  formData.append('password2', payload['passwordtwo']);
  formData.append('frist_name', payload['username']);
  formData.append('last_name', payload['username']);


  this.httpClient.post('http://13.48.23.181:8000/api/sign_up/', formData).subscribe((data) => {
    console.log(data);
    this._snackBar.open( "User has been created successfully", "success",{
      duration: 3000
    });
    this.dialogRef.close();
  })
}
```

Figure 25: SignUp Logic

Instead of keeping the original password, the generate password bearer function encrypts the password and stores the encrypted password in the database.

```
        read_only_fields = ('id',)

class LogInSerializer(TokenObtainPairSerializer):
    @classmethod
    def get_token(cls, user):
        token = super().get_token(user)
        user_data = UserSerializer(user).data
        for key, value in user_data.items():
            if key != 'id':
                token[key] = value
        return token
```

```
ngOnInit(){
  const token_value = localStorage.getItem('tokens')
  console.log("on it token info");
  if (token_value != null) {
    const data = JSON.parse(token_value);
    console.log(data.access)
    let headers = new HttpHeaders();
    headers = headers.append( 'Content-Type', 'application/json'),
    headers = headers.append( "Authorization", 'Bearer ' + data.access )
    this.httpClient.get('http://13.48.23.181/:8000/api/validatetoken/', {headers: headers, observe: "response"}).subscribe(data
      console.log(data);
      if (data.status == 200){
        this.is_logged=true;
        const user_name: string | null = localStorage.getItem('username')
        if (user_name){
          this.username = user_name
        }
      }
```

Figure 26: Token-based Logic

6.2 LOGIN SCREEN

The user must have a working email address and password to access the login page. Since both fields are required, the user must input accurate and legitimate credentials.

```
userLogin(payload: LoginModel){
  console.log(payload);
  console.log(payload['username']);
  var formData: any = new FormData();
  formData.append('username', payload['username']);
  formData.append('password', payload['password']);
  return this.httpClient.post('http://13.48.23.181:8000/api/log_in/', formData).pipe(map((data) => {

      var token = data as TokenModel;
      localStorage.setItem('tokens', JSON.stringify(token));
      var userInfo = this.jwtService.decodeToken( token.access ) as UserProfile;
      this.userProfile.next(userInfo);
      this.data["username"] = userInfo.username;
      localStorage.setItem('username', userInfo.username);
      this.dialogRef.close({data: this.data});
      // this.data = {"first_name": "kunafn", "last_name": "kunln", "username": "kunausr"};
      return true;
    }),
  catchError((error) => {
    console.log(error);
    return of(false);
  })
  );
}
```

Figure 27: Login JWT Authentication

6.3 LIST OF DOCTORS VIEW

```
onSubmit(event: NgForm) {
  console.log("Form Submitted!");
  console.log(event.value);
  this.doctorslist = []
  this.current_session_data = {"date": event.value['date'], "speciality": event.value['specality'], "time":event.value['timings'

  const token_value = localStorage.getItem('tokens')
  console.log("on it token info");
  if (token_value != null) {
    const data = JSON.parse(token_value);
    console.log(data.access)
    let headers = new HttpHeaders();
    headers = headers.append( "Authorization", 'Bearer ' + data.access )

    var formData: any = new FormData();
    formData.append('specality', event.value['specality']);
    formData.append('timings', event.value['timings']);
    formData.append('date', event.value['date']);

    console.log("list_doctors");

    this.httpClient.post<doctor_http>('http://13.48.23.181:8000/api/listdoctors/', formData, {headers: headers}).subscribe((data

      for (let i = 0; i < data_http.data.length ; i++){
        this.doctorslist.push(data_http.data[i]);
      }
      console.log(this.doctorslist)
    })
  } else {
    this._snackBar.open( "User not logged in", "failed",{
```

Figure 28: Form Submission for Doctor Appointment

6.4 Book Appointment Backend

```
bookappointment(event: any){
  const token_value = localStorage.getItem('tokens')
  if (token_value != null) {
    const data = JSON.parse(token_value);
    console.log(data.access)
    let headers = new HttpHeaders();
    headers = headers.append( "Authorization", 'Bearer ' + data.access )

    var formData: any = new FormData();
    formData.append('timings', this.current_session_data.time);
    formData.append('date', this.current_session_data.date);
    formData.append('doctor_id', event);

    this.httpClient.post('http://13.48.23.181:8000/api/createappointment/', formData, {headers: headers}).subscribe((data) =>
      this._snackBar.open( "Appointment has been booked", "success",{
        duration: 3000
      });
      console.log(data);
      this.getappointment();
    })
  } else {
    this._snackBar.open( "User not logged in", "failed",{
      duration: 3000
    });
  }
}
```

Figure 29: Book Appointment Logic

6.5 Get Appointment Backend

```
getappointment(){
  this.appointment_list_status = false;
  const token_value = localStorage.getItem('tokens')
  this.dataSource = [];
  if (token_value != null) {
    const data = JSON.parse(token_value);
    console.log(data.access)
    let headers = new HttpHeaders();
    headers = headers.append( "Authorization", 'Bearer ' + data.access )

    this.httpClient.get<appointment_session_data>('http://13.48.23.181:8000/api/getappointment/', {headers: headers}).sub
      console.log(data.data);
      for (let i = 0; i < data.data.length ; i++){
        this.dataSource.push(data.data[i]);
      }
      this.appointment_list_status = true;
    })
  } else {
    this._snackBar.open( "User not logged in", "failed",{
      duration: 3000
    });
  }
}
```

Figure 30: Get Appointment Logic

6.6 Delete Appointment Backend

```
deleteappointment(event: string){
  const token_value = localStorage.getItem('tokens')
  this.appointment_list_status = false;
  if (token_value != null) {
    const data = JSON.parse(token_value);
    console.log(data.access)
    let headers = new HttpHeaders();
    headers = headers.append( "Authorization", 'Bearer ' + data.access )

    var formData: any = new FormData();
    formData.append('id', event);

    this.httpClient.post('http://13.48.23.181:8000/api/deleteappointment/', formData, {headers: headers}).subscribe((data
      this._snackBar.open( JSON.stringify(data), "success");
      this.getappointment();
      })
  } else {
    this._snackBar.open( "User not logged in", "failed",{
      duration: 3000
    });
  }
}
```

Figure 31: Delete Appointment Logic

7. SECURITY AND MEASURES

7.1 JWT BEARER AUTHENTICATION

An open standard (RFC 7519) called JSON Web Token (JWT) is used to safely transport data between parties in the form of JSON objects. It is small, readable, and digitally signed by the Identity Provider using a private key or a public key pair (IdP). As a result, the token's legitimacy and integrity can be confirmed by other parties. Using JWT serves to assure data authenticity rather than to conceal data. JWT is not encrypted; it is signed and encoded.

JWT is a stateless, token-based authentication method. As the session is client-side based and stateless, the server does not entirely rely on a datastore (database) to store session data.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    )
}

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': datetime.timedelta(minutes=60),
    'REFRESH_TOKEN_LIFETIME': datetime.timedelta(days=1),
    'USER_ID_CLAIM': 'id',
}

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
```

Figure 32: JWT Settings

```
formData.append('username', payload['username']);
formData.append('password', payload['password']);
return this.httpClient.post('http://13.48.23.181:8000/api/log_in/', formData).pipe(map((data) => {

    var token = data as TokenModel;
    localStorage.setItem('tokens', JSON.stringify(token));
    var userInfo = this.jwtService.decodeToken( token.access ) as UserProfile;
    this.userProfile.next(userInfo);
    this.data["username"] = userInfo.username;
    localStorage.setItem('username', userInfo.username);
    this.dialogRef.close({data: this.data});
```

Figure 33: Tokenization

```
class GoogleView(APIView):
    def post(self, request):
        payload = {'access_token': request.data.get("token")}  # validate the token
        r = requests.get('https://www.googleapis.com/oauth2/v2/userinfo', params=payload)
        data = json.loads(r.text)

        if 'error' in data:
            content = {'message': 'wrong google token / this google token is already expired.'}
            return Response(content)

        # create user if not exist
        try:
            user = User.objects.get(email=data['email'])
        except User.DoesNotExist:
            user = User()
            user.username = data['email']
            # provider random default password
            user.password = make_password(BaseUserManager().make_random_password())
            user.email = data['email']
            user.save()

        token = RefreshToken.for_user(user)  # generate token without username & password
        response = {}
        response['username'] = user.username
        response['access_token'] = str(token.access_token)
        response['refresh_token'] = str(token)
        return Response(response)
```

Figure 34: Google API OAuth Tokenisation

Below is the screenshot which indicates the bearer-based authentication that is used for the web services and the code below is for the submission of the form.

```
const token_value = localStorage.getItem('tokens')
console.log("on it token info");
if (token_value != null) {
  const data = JSON.parse(token_value);
  console.log(data.access)
  let headers = new HttpHeaders();
  headers = headers.append( "Authorization", 'Bearer ' + data.access )

  var formData: any = new FormData();
  formData.append('specality', event.value['specality']);
  formData.append('timings', event.value['timings']);
  formData.append('date', event.value['date']);

  console.log("list_doctors");

  this.httpClient.post<doctor_http>('http://13.48.23.181:8000/api/listdoctors/', formData, {headers: headers}).subscribe((data

    for (let i = 0; i < data_http.data.length ; i++){
      this.doctorslist.push(data_http.data[i]);
    }
    console.log(this.doctorslist)
  })
} else {
  this._snackBar.open( "User not logged in", "failed",{
    duration: 3000
```

Figure 35: Tokenisation Used for Api Services

	id	password	last_login	is_superuser	username	last_name	email	is_staff	is_active	date_joined	first
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	pbkdf2_sha256$260000$t0TimR2eioK...	2023-04-02 18:09:58.427278	1	admin			1	1	2023-04-02 18:07:41.604815	
2	2	pbkdf2_sha256$260000$TGp6Hh67qh...	NULL	0	navin	navin		0	1	2023-04-02 18:25:24.556470	
3	3	pbkdf2_sha256$260000$47hl88OywN...	NULL	0	navin1			0	1	2023-04-02 18:49:26.065357	
4	4	pbkdf2_sha256$260000$OG3aSNevO...	NULL	0	abc	abc		0	1	2023-04-03 21:04:11.051321	
5	5	pbkdf2_sha256$260000$nwSwNFb2n...	NULL	0	qwer	qwer		0	1	2023-04-03 21:11:24.821769	
6	6	pbkdf2_sha256$260000$AjG8AzT4tK...	NULL	0	321	321		0	1	2023-04-03 21:28:42.320073	
7	7	pbkdf2_sha256$260000$v6Oca5N0n7...	NULL	0	5432	5432		0	1	2023-04-03 21:53:18.849750	
8	8	pbkdf2_sha256$260000$oVKFNgrdYb...	NULL	0	testuser	testuser		0	1	2023-04-03 21:59:19.279572	
9	9	pbkdf2_sha256$260000$bSJ1htJBIXii...	NULL	0	asd	asd		0	1	2023-04-03 22:08:11.099691	

Figure 36: Login Hash

8. WEB APPLICATION FEATURES

Login and Logout Feature:

Every page of the web application's navigation bar features a login and logout button. By pressing this button, the user can sign up for an account.

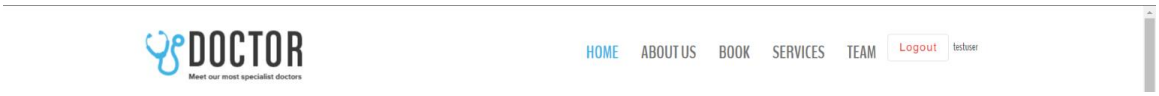


Figure 37: Login and Logout Feature

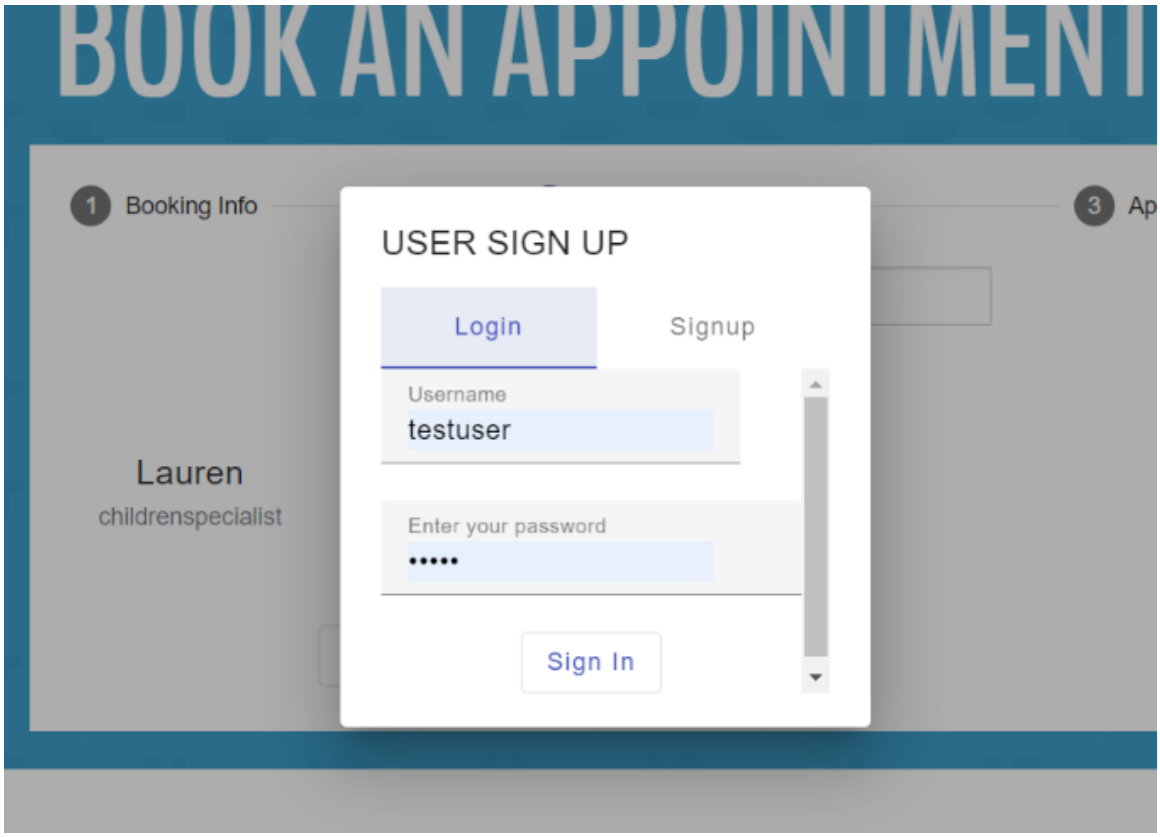


Figure 38: Login and Register popup

Search Feature:

This application consists of search functionality which can be navigated through the book appointment section while filtering the data of the doctors and the user can look through all of the physicians.

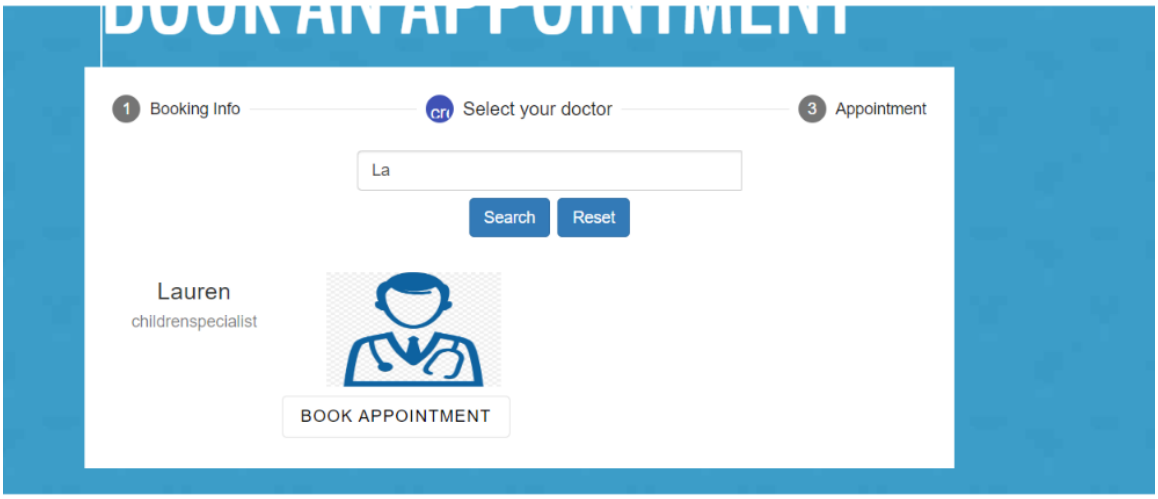


Figure 39: Search Feature

List of Appointments Feature: This is used to view the details using the rest API services

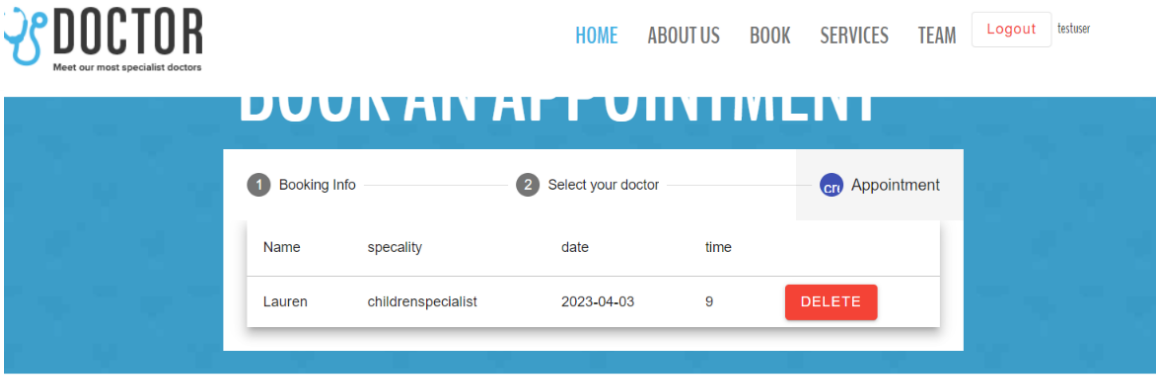


Figure 40: List Appointment Feature

9. AWS Deployment

In this section, we can see the instance created as part of the deployment in the Amazon EC2

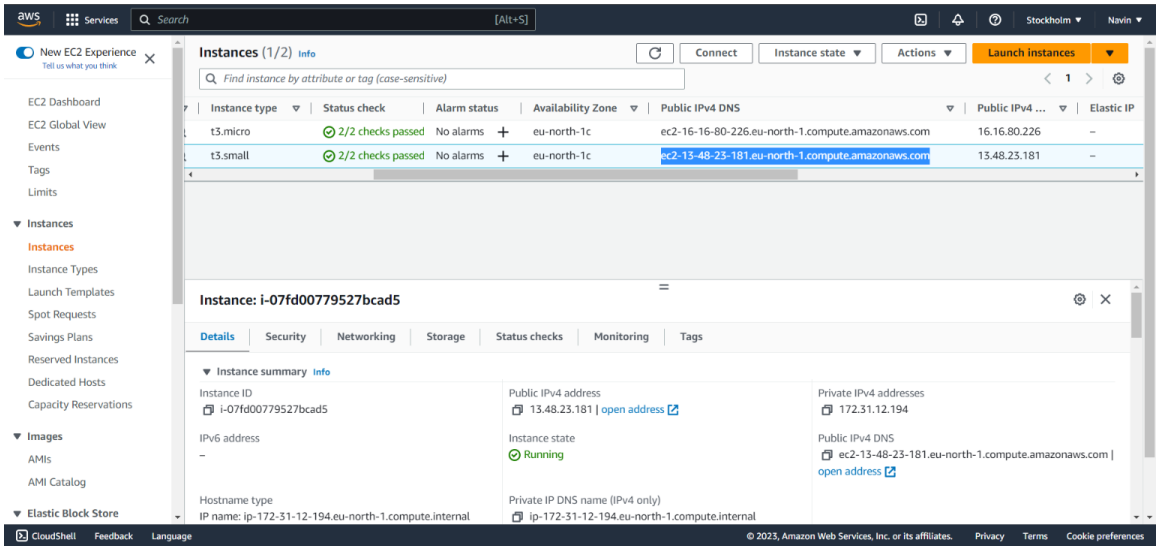


Figure 41: AWS EC2 Deployment

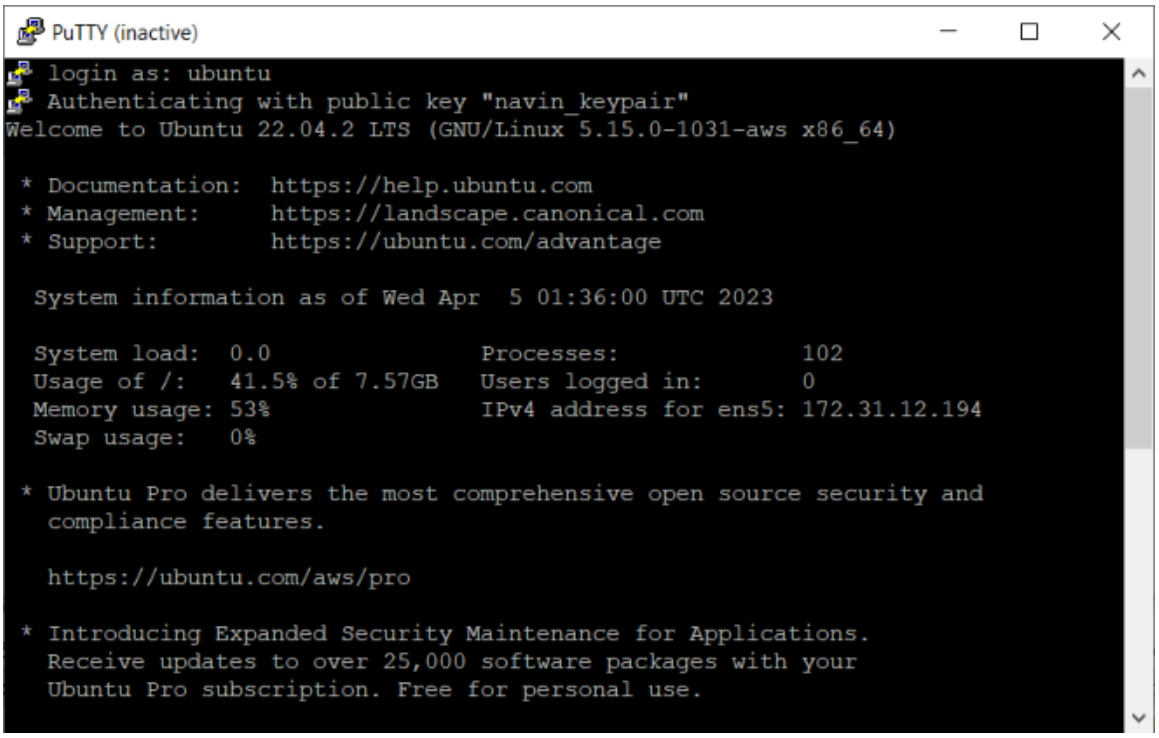


Figure 42: Putty Console deployment

Login as: ubuntu

Authenticating with public key "keypair" added as part of the artifact.

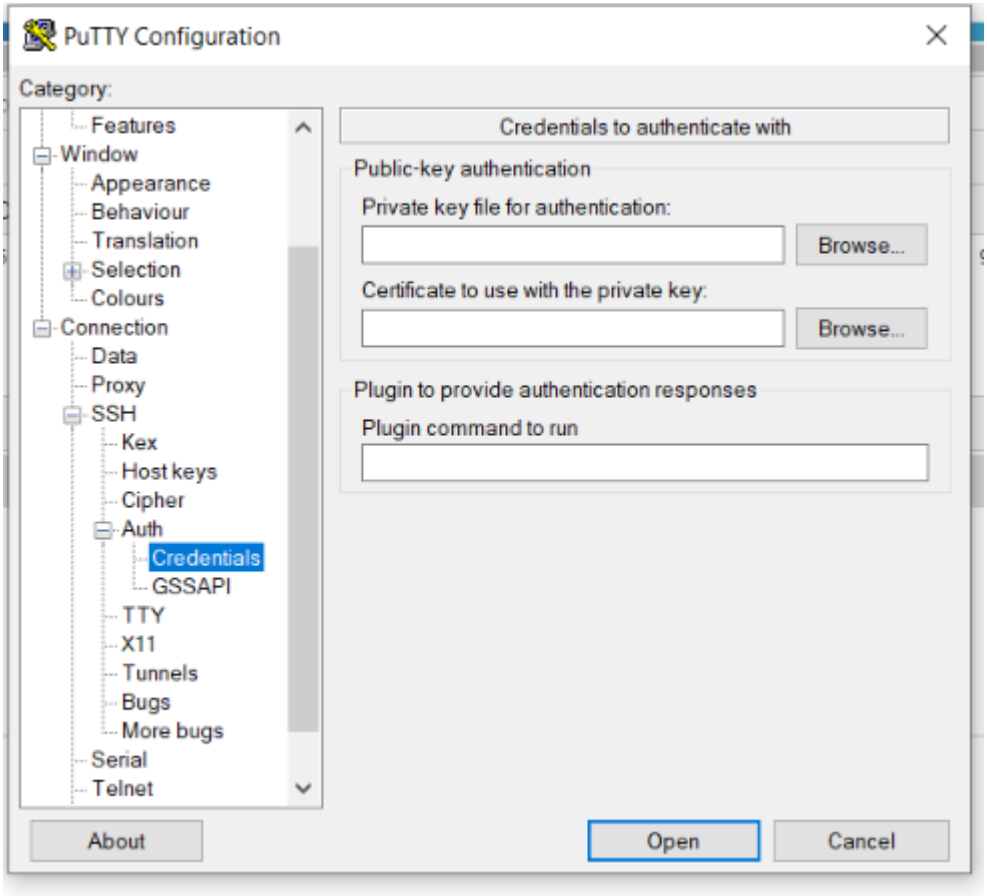
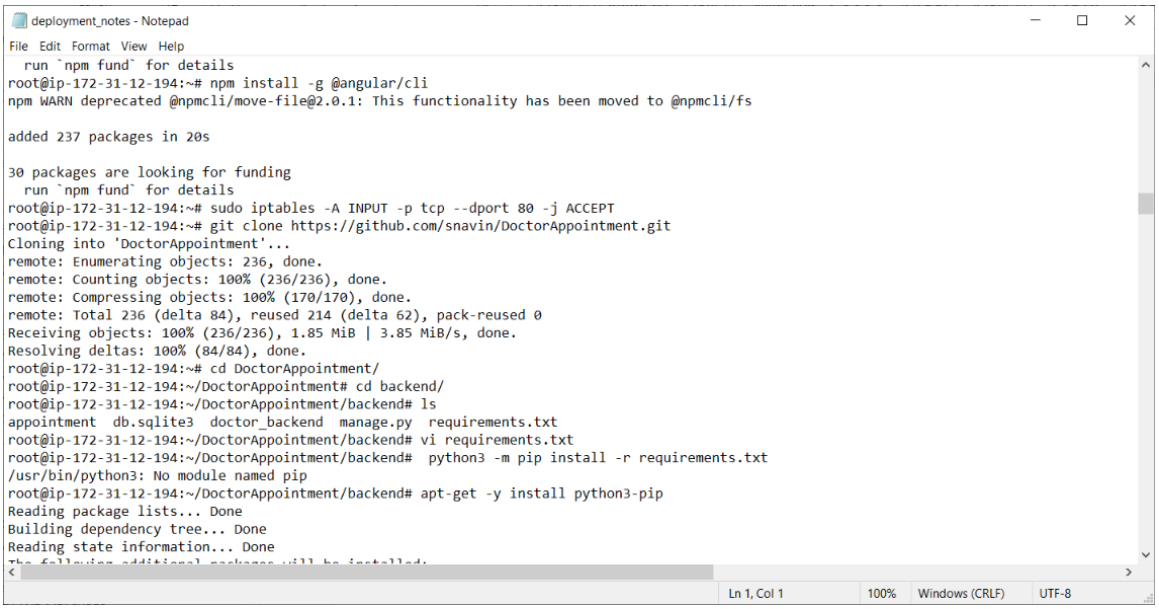


Figure 43: Putty Configuration

The deployment notes are added as part of the artifacts folder



```
deployment_notes - Notepad
File  Edit  Format  View  Help
  run `npm fund` for details
root@ip-172-31-12-194:~# npm install -g @angular/cli
npm WARN deprecated @npmcli/move-file@2.0.1: This functionality has been moved to @npmcli/fs

added 237 packages in 20s

30 packages are looking for funding
  run `npm fund` for details
root@ip-172-31-12-194:~# sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
root@ip-172-31-12-194:~# git clone https://github.com/snavin/DoctorAppointment.git
Cloning into 'DoctorAppointment'...
remote: Enumerating objects: 236, done.
remote: Counting objects: 100% (236/236), done.
remote: Compressing objects: 100% (170/170), done.
remote: Total 236 (delta 84), reused 214 (delta 62), pack-reused 0
Receiving objects: 100% (236/236), 1.85 MiB | 3.85 MiB/s, done.
Resolving deltas: 100% (84/84), done.
root@ip-172-31-12-194:~# cd DoctorAppointment/
root@ip-172-31-12-194:~/DoctorAppointment# cd backend/
root@ip-172-31-12-194:~/DoctorAppointment/backend# ls
appointment  db.sqlite3  doctor_backend  manage.py  requirements.txt
root@ip-172-31-12-194:~/DoctorAppointment/backend# vi requirements.txt
root@ip-172-31-12-194:~/DoctorAppointment/backend#  python3 -m pip install -r requirements.txt
/usr/bin/python3: No module named pip
root@ip-172-31-12-194:~/DoctorAppointment/backend# apt-get -y install python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Ln 1, Col 1   100%   Windows (CRLF)   UTF-8
```

Figure 44: Commands used for Deployment via Putty

10. LINKS

GitHub Link: https://github.com/snavin/DoctorAppointment.git

AWS Link: http://13.48.23.181/

DJANGO Admin Link: http://13.48.23.181:8000/admin/login/?next=/admin/

11. REFERENCES:

1. T Haerder and A Reuter, "Principles of transaction-oriented database recovery", *CSUR*, vol. 15, pp. 287-317, Apr. 1983.

2. F Ramisch and M Rieger, "Recovery of SQLite3 Data Using Expired Indexes", 9th International Conference on IT Security Incident Management & IT Forensics, pp. 147-152, Nov. 2015.

3. MDB - Material Design for Bootstrap 5 & 4 (no date). Available at: https://mdbootstrap.com/ (Accessed: 5 April 2023).

4. Contributors, M.O., Jacob Thornton, and Bootstrap (no date) Accordion. Available at: https://getbootstrap.com/docs/5.2/components/accordion/ (Accessed: 5 April 2023).

5. Youtube.com. 2021. How to use the vagrant box to create AWS EC2 instance? (2021). Available at: https://www.youtube.com/watch?v=im-shBCnP2E (Accessed: 5 April 2023).

6. Download PuTTY - a free SSH and telnet client for Windows (no date). Available at: https://www.putty.org/ (Accessed: 5 April 2023).

7. Python Release Python 3.11.0 (no date) Python.org. Available at: https://www.python.org/downloads/release/python-3110/ (Accessed: 5 April 2023).

8. Secure and resizable cloud computing – Amazon EC2 – Amazon Web Services (no date). Available at: https://aws.amazon.com/ec2/ (Accessed: 5 April 2023).

9. Team, A.C. (no date) Angular Material, Angular Material. Available at: https://material.angular.io/ (Accessed: 5 April 2023).

10. Free CSS | 3472 Free Website Templates, CSS Templates, and Open Source Templates (no date). Available at: https://www.free-css.com/ (Accessed: 5 April 2023).