

HEALTHCARE MANAGEMENT SYSTEM

Documented by: Mayur Landge

To develop a prototype of a healthcare management system for managing patient appointments and medical information using Python, we use Django web framework. Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.

1. First, make sure you have Django installed. You can install it via pip:

```
pip install django
```

2. Now, let's create a Django project and application for our healthcare management system:

create a Django project by using following command:

```
django-admin startproject Healthcare_Managemment
```

Enter inside the Project directory by using “**cd Healthcare_Managemment**” then create an application for our project:

```
py manage.py startapp appointments_records
```

3. Now let's register our Application into “**settings.py**” -> “**Installed App**” Sections:

```
INSTALLED_APPS = [  
    "django.contrib.admin",  
    "django.contrib.auth",  
    "django.contrib.contenttypes",  
    "django.contrib.sessions",  
    "django.contrib.messages",  
    "django.contrib.staticfiles",  
    "appointments_records",  
    "rest_framework",  
]
```

4. Now, let's define our **models, serializers, views, and templates**.

1. In “appointments_records/models.py”

```
from django.db import models

# Create your models here.

class Patient(models.Model):
    id = models.IntegerField(primary_key=True)
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    gender = models.CharField(max_length=6)
    contact = models.BigIntegerField(default=0)
    email = models.EmailField(unique=True)

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

Here we mention some required data to fill.

If you have a model with a field named “**name**”, defining the “**__str__**” method to return the value of that field “(**self.name**)” provides a readable representation of the object. when you view a list of “**Patient**” objects, you'll see their names displayed instead of something less meaningful like object IDs or memory addresses.

```
class Appointment(models.Model):
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE)
    date_time = models.DateTimeField()
    reason = models.TextField()

    def __str__(self):
        return "Appointment for {} at {}".format(self.patient, self.date_time)
```

If you set “**on_delete=models.CASCADE**” on this ForeignKey field in the “**Patient**” model and then you delete an “**Name**”, Django will automatically delete all associated “**Patient**” objects belonging to that NAME.

```
class MedicalRecord(models.Model):
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE)
    # Add fields for medical records
    # Example: condition, diagnosis, medication, etc.

    def __str__(self):
        return f"Medical record for {self.patient}"
```

Here we only maintaining the Patient record with **"Patient_ID"** in this ForeignKey field.

2. Serializers:

Django REST Framework is a powerful toolkit for building Web APIs in Django. The **"serializers"** module within DRF provides a way to convert complex data types, such as query sets and model instances, into native Python datatypes that can then be easily rendered into JSON, XML, or other content types.

Here's what you can do with serializers in DRF:

"Serialize Data", "Deserialize Data", "Validation" and "Relationships Handling"

Here, first we have to install **"DJANGO REST_FRAMEWORK"** via **pip**:

```
pip install djangorestframework
```

3. In **"appointments_records/serializers.py"**

First of all, we have to create a **"serializer.py"** file inside our application **"appointments_records"**, then here we import all class from **"appointments_records/models.py"**.

```
from rest_framework import serializers
from .models import Patient, Appointment, MedicalRecord

class PatientSerializer(serializers.ModelSerializer):
    class Meta:
        model = Patient
        fields = '__all__'

class AppointmentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Appointment
        fields = '__all__'

class MedicalRecordSerializer(serializers.ModelSerializer):
    class Meta:
        model = MedicalRecord
        fields = '__all__'
```

Then we have to serialize the all model class by using meta class, in it we have to define **"model = its class"** and **"fields = "__all__"**, which means we have applied serializer on all parameters present in **"models.py"**.

4. In “appointments_records/views.py”

```
from django.shortcuts import render
from rest_framework import generics
from rest_framework.permissions import IsAuthenticated
from django.contrib.auth.decorators import login_required
from .models import Patient, Appointment, MedicalRecord
from .serializers import PatientSerializer, AppointmentSerializer, MedicalRecordSerializer
```

In “**views.py**” we have imported all the required modules with their class

Here we imported “**generics**” module to ““**create**”, “**update**”, “**delete**” **APIViews**”:

```
class PatientListCreate(generics.ListCreateAPIView):
    queryset = Patient.objects.all() #ORM Query
    serializer_class = PatientSerializer
    permission_classes = [IsAuthenticated]

class PatientRetrieveUpdateDestroy(generics.RetrieveUpdateDestroyAPIView):
    queryset = Patient.objects.all() #ORM Query
    serializer_class = PatientSerializer
    permission_classes = [IsAuthenticated]
```

In “**queryset**” variable we have used the **Object relationship mapping** (ORM query) to filter the data.

Then from **serializers.py** we imported **serializers** for each class

Then to protect all the Healthcare data we have applied some **authentication** to give “**permission**” to **admin/authorized person**.

```
class AppointmentListCreate(generics.ListCreateAPIView):
    queryset = Appointment.objects.all() #ORM Query
    serializer_class = AppointmentSerializer
    permission_classes = [IsAuthenticated]

class AppointmentRetrieveUpdateDestroy(generics.RetrieveUpdateDestroyAPIView):
    queryset = Appointment.objects.all() #ORM Query
    serializer_class = AppointmentSerializer
    permission_classes = [IsAuthenticated]

class MedicalRecordListCreate(generics.ListCreateAPIView):
    queryset = MedicalRecord.objects.all() #ORM Query
    serializer_class = MedicalRecordSerializer
    permission_classes = [IsAuthenticated]

class MedicalRecordRetrieveUpdateDestroy(generics.RetrieveUpdateDestroyAPIView):
    queryset = MedicalRecord.objects.all() #ORM Query
    serializer_class = MedicalRecordSerializer
    permission_classes = [IsAuthenticated]
```

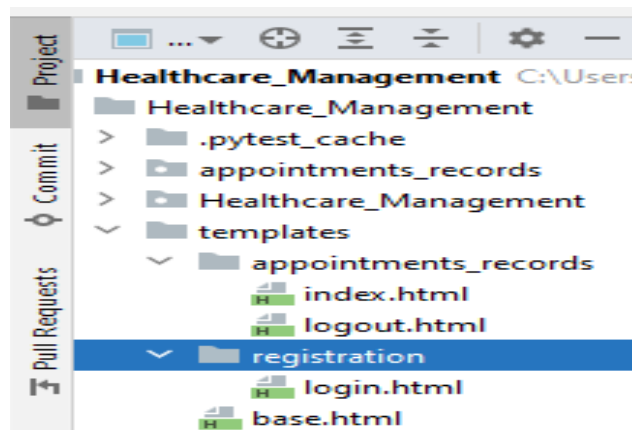
This is all about API views.

Now views for **dashboard:**

```
def Index(request):  
    return render(request, 'appointments_records/index.html')  
  
def Logout(r):  
    return render(r, 'appointments_records/logout.html')
```

Now here we have mentioned some **html pages**, firstly we have to create templates.

5. In “**Healthcare_Management**” we have to create a “**templates**” directory in that we have to create application directory “**appointments_records**” in it “**index.html**” & “**logout.html**” and “**base.html**” page.

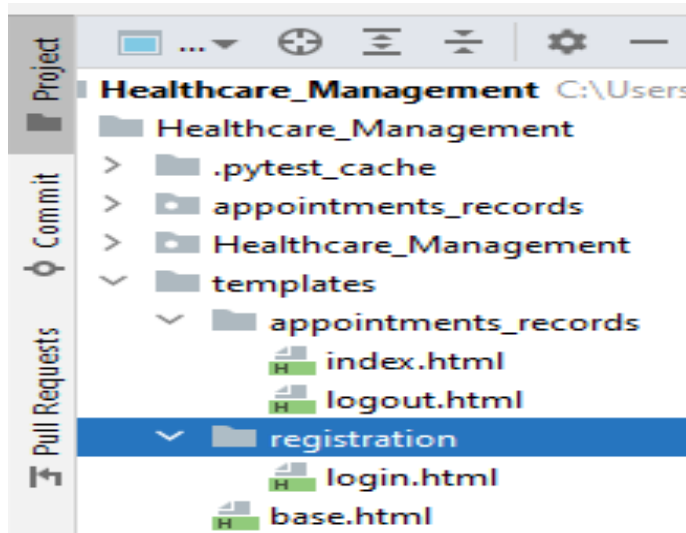


6. Also, we have to **register the templates path** inside “**Healthcare_Management/settings.py**”,

```
# Build paths inside the project like this: BASE_DIR / 'subdir'.  
BASE_DIR = Path(__file__).resolve().parent.parent  
TEMPLATE_DIR = Path.joinpath(BASE_DIR, 'templates')
```

```
TEMPLATES = [  
    {  
        "BACKEND": "django.template.backends.django.DjangoTemplates",  
        "DIRS": [TEMPLATE_DIR],  
        "APP_DIRS": True,  
        "OPTIONS": {
```

7. Now as we have applied authentication so we have to make “registration” directory inside it “login” page in “templates” directory:



8. Now in “views.py” we have applied “login required” decorator to “login” and “logout”:

```
@login_required
def Index(request):
    return render(request, 'appointments_records/index.html')

@login_required
def Logout(r):
    return render(r, 'appointments_records/logout.html')
```

9. Here we have to created “base.html”, “index.html”, “login.html” and “logout.html”

“base.html”:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Healthcare Management system</title><link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
<script
src="https://cdn.jsdelivrivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js
" integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikTlwxGysOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>
</head>
```

```

<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <div class="container-fluid">
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-
expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavDropdown">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page"
href="/appointments">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/appointments/patients"> View Patient </a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/appointments/appointments"> Schedule
Appointment </a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/appointments/records"> Manage Medical
Records </a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/accounts/logout"> Logout </a>
        </li>
      </ul>
    </div>
  </div>
</nav>
</body>
{% block baseblock %}
{% endblock %}
</html>

```

"index.html":

```

<!DOCTYPE html>
{% extends 'base.html' %}
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Index</title>
</head>
{% block baseblock %}
<body>
  <h1> Welcome to Healthcare Management DASHBOARD </h1>
</body>
{% endblock %}
</html>

```

“login.html”:

```
<!DOCTYPE html>
{% extends 'base.html' %}
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Login</title>
</head>
{% block baseblock %}
<body>
<div class="container" align="center">
  <h1> Login </h1>
  <form method="post">
    {{form.as_p}}
    {% csrf_token %}
    <button type="submit"> Login </button>
  </form>
</div>
</body>
{% endblock %}
</html>
```

“logout.html”:

```
<!DOCTYPE html>
{% extends 'base.html' %}
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
{% block baseblock %}
<body>
<div class="container">
  <h1> Thanks for Visiting</h1>
  <a href="/accounts/login" class="btn btn-primary btn-lg btn-success">
    Login
  </a>
</div>
</body>
{% endblock %}
</html>
```

5. Now we have to create a Data base and register in
“Healthcare_Management/settings.py”,

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'Healthcare_Management',
        'USER': 'root',
        'PASSWORD': 'Root*@1234@*'
    }
}
```


6. Now we have to convert our **models** into **SQL format** and **create a table** into the **database** using following commands:

convert our **models** into **SQL format**

```
Py manage.py makemigrations
```

create a table into the **database**

```
Py manage.py migrate
```

7. As we have given the authentications, we have to write logic and register models it in “**appointments_records/admin.py**”:

```
from django.contrib import admin
from .models import Patient, Appointment, MedicalRecord

# Register your models here.

class PatientAdmin(admin.ModelAdmin):
    list_display = ['id', 'name', 'age', 'gender', 'email']

class AppointmentAdmin(admin.ModelAdmin):
    list_display = ['patient', 'date_time', 'reason']

class MedicalRecordAdmin(admin.ModelAdmin):
    list_display = ['patient']

admin.site.register(Patient, PatientAdmin)
admin.site.register(Appointment, AppointmentAdmin)
admin.site.register(MedicalRecord, MedicalRecordAdmin)
```

8. Now we have to “**create superuser**” to handle the application by **authorised person** using following command:

```
Py manage.py createsuperuser
```

```
Username:
Email:
Password:
Confirm_password:
```

9. Now we have to create the “**urls.py**” file into application level i.e., “**appointments_records/urls.py**”:

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('', views.Index, name='index'),
    path('patients/', views.PatientListCreate.as_view(), name='patient-list-create'),
    path('patients/<int:pk>', views.PatientRetrieveUpdateDestroy.as_view(), name='patient-retrieve-update-destroy'),
    path('appointments/', views.AppointmentListCreate.as_view(), name='appointment-list-create'),
    path('appointments/<int:pk>', views.AppointmentRetrieveUpdateDestroy.as_view(), name='appointment-retrieve-update-destroy'),
    path('records/', views.MedicalRecordListCreate.as_view(), name='medical-record-list-create'),
    path('records/<int:pk>', views.MedicalRecordRetrieveUpdateDestroy.as_view(), name='medical-record-retrieve-update-destroy'),
    path('logout/', views.Logout),
]
```

Here, we have **patient**, **appointments**, **records**:

'patients/': This part of the URL specifies a base path. It means that any URL that starts with **'patients/'** will be matched by this pattern.

Similarly, with the “**appointments/**” and “**records/**”.

“patient/<int:pk>”: This converter expects an integer value and assigns it to a variable named **pk**. **pk** is a common abbreviation for “primary key”.

Putting it all together, **'patients/int:pk/'** matches URLs that start with **'patients/'** followed by an integer value, which is captured and stored as “**pk**”, and ends with a trailing slash.

when you have a Django application for managing patient records, **'patients/int:pk/'** might be used to display details of a specific patient based on their primary key.

10. Now we are going to create **urls** in project level i.e., “**Healthcare_Management/urls.py**”,

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path("appointments/", include('appointments_records.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
]
```

`path('accounts/', ...)`: This defines a URL pattern that matches any URL starting with 'accounts/'. It serves as a base URL for authentication-related views.

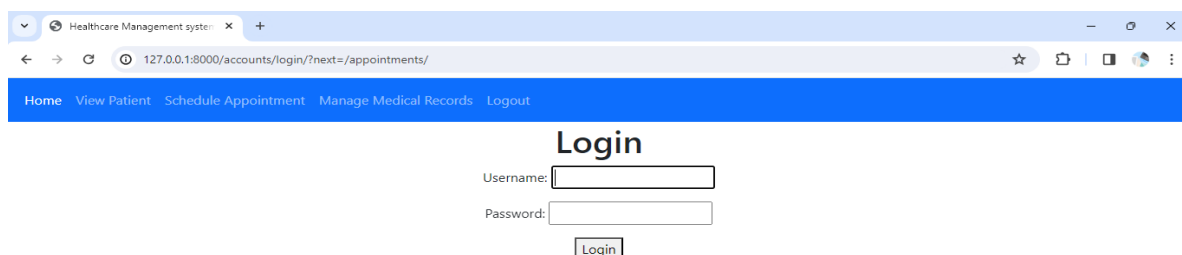
This function tells Django to include the URL patterns defined in the `django.contrib.auth.urls` module.

if you have 'accounts/login/' defined in `django.contrib.auth.urls`, it will handle user authentication, and you don't need to define a separate view for the login functionality. Similarly, other authentication-related URLs like 'accounts/logout/', 'accounts/password_reset/', etc., will be handled by Django's built-in authentication views.

11. Run the Project using following command:

```
Py manage.py runserver
```

Frontend Interface of application asking for **Login ID** to handle Healthcare Management System using “URL”: <http://127.0.0.1:8000/appointments>



The screenshot shows a web browser window with the title 'Healthcare Management system'. The address bar displays the URL '127.0.0.1:8000/accounts/login/?next=/appointments/'. A blue navigation bar at the top contains links: 'Home', 'View Patient', 'Schedule Appointment', 'Manage Medical Records', and 'Logout'. The main content area is titled 'Login' and features a form with two input fields: 'Username:' and 'Password:'. Below these fields is a 'Login' button.