**Healthcare Report Documentary Content Management System (CMS)**

Creating a **Healthcare Report Documentary Content Management System (CMS)** requires a structured approach involving multiple components: backend (server-side), frontend (client-side), and static files like CSS for styling.

**Structure of the CMS Application**

1. **Backend (Python/Django)**:
   * Django as the backend framework for content management.
   * Model for healthcare reports and documentary data.
   * Views for managing reports (CRUD operations).
   * Forms for creating/editing reports.
2. **Frontend (HTML/CSS/JavaScript)**:
   * HTML templates for displaying content.
   * CSS for styling the layouts and documents.
   * JavaScript for interactivity and form validation.
3. **Static Files**:
   * CSS for layout, forms, and buttons.
   * Static images or icons if needed.

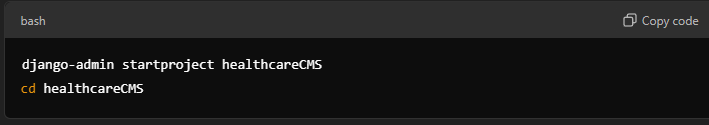
**Step-by-Step Breakdown**

**1. Backend: Django (Python Framework)**

**Install Django**

A black rectangular object with a black stripe

Description automatically generatedpip install Django

Create a Django project: 

django-admin startproject healthcareCMS

cd healthcareCMS

Create a Django App for Report Management:

A black rectangular object with a black stripe

Description automatically generated

python manage.py startapp reports

**Update the settings.py for Static Files**

In healthcareCMS/settings.py, configure the static files settings:

A black rectangular object with a black border

Description automatically generated

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

BASE\_DIR / "static",

]

**Define Models for Reports**

In reports/models.py:

A computer screen shot of a program code

Description automatically generated

**from django.db import models**

**class Report(models.Model):**

**title = models.CharField(max\_length=200)**

**description = models.TextField()**

**document = models.FileField(upload\_to='documents/')**

**date\_published = models.DateTimeField(auto\_now\_add=True)**

**def \_\_str\_\_(self):**

**return self.title**

**Define Forms for Handling Reports**

In reports/forms.py:

A computer screen with white text

Description automatically generated

from django import forms

from .models import Report

class ReportForm(forms.ModelForm):

class Meta:

model = Report

fields = ['title', 'description', 'document']

**Define Views for Managing Reports**

In reports/views.py:

A screenshot of a computer program

Description automatically generated

from django.shortcuts import render, redirect

from .models import Report

from .forms import ReportForm

def report\_list(request):

reports = Report.objects.all().order\_by('-date\_published')

return render(request, 'reports/report\_list.html', {'reports': reports})

def report\_detail(request, pk):

report = Report.objects.get(pk=pk)

return render(request, 'reports/report\_detail.html', {'report': report})

def report\_create(request):

if request.method == 'POST':

form = ReportForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('report\_list')

else:

form = ReportForm()

return render(request, 'reports/report\_form.html', {'form': form})

def report\_edit(request, pk):

report = Report.objects.get(pk=pk)

if request.method == 'POST':

form = ReportForm(request.POST, request.FILES, instance=report)

if form.is\_valid():

form.save()

return redirect('report\_detail', pk=pk)

else:

form = ReportForm(instance=report)

return render(request, 'reports/report\_form.html', {'form': form})

def report\_delete(request, pk):

report = Report.objects.get(pk=pk)

report.delete()

return redirect('report\_list')

**Define URLs for the Application**

In reports/urls.py:

A screenshot of a computer program

Description automatically generated

from django.urls import path

from . import views

urlpatterns = [

path('', views.report\_list, name='report\_list'),

path('report/<int:pk>/', views.report\_detail, name='report\_detail'),

path('report/new/', views.report\_create, name='report\_create'),

path('report/<int:pk>/edit/', views.report\_edit, name='report\_edit'),

path('report/<int:pk>/delete/', views.report\_delete, name='report\_delete'),

]

In healthcareCMS/urls.py:

A screenshot of a computer program

Description automatically generated

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('reports.urls')),

]

**2. Frontend: HTML Templates**

**Base Template: base.html**

A screenshot of a computer program

Description automatically generated

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Healthcare Report CMS</title>

<link rel="stylesheet" href="{% static 'css/styles.css' %}">

</head>

<body>

<header>

<h1>Healthcare Report Documentary CMS</h1>

</header>

<nav>

<ul>

<li><a href="{% url 'report\_list' %}">Home</a></li>

<li><a href="{% url 'report\_create' %}">Create Report</a></li>

</ul>

</nav>

<main>

{% block content %}

{% endblock %}

</main>

<footer>

<p>&copy; 2024 Healthcare CMS</p>

</footer>

</body>

</html>

**Report List Template: report\_list.html**

A screenshot of a computer program

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h2>Reports</h2>

<ul>

{% for report in reports %}

<li>

<a href="{% url 'report\_detail' report.pk %}">{{ report.title }}</a>

- Published on {{ report.date\_published }}

</li>

{% endfor %}

</ul>

{% endblock %}

Report Detail Template: report\_detail.html:

A screenshot of a computer program

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h2>{{ report.title }}</h2>

<p>{{ report.description }}</p>

<a href="{{ report.document.url }}">Download Document</a>

<a href="{% url 'report\_edit' report.pk %}">Edit</a>

<a href="{% url 'report\_delete' report.pk %}">Delete</a>

{% endblock %}

Report Form Template: report\_form.html:

A screenshot of a computer program

Description automatically generated

**3. Static Files: CSS for Styling**

Here is the detailed CSS for static/css/styles.css:

A screenshot of a computer program

Description automatically generated

/\* General Styles \*/

body {

font-family: 'Arial', sans-serif;

margin: 0;

padding: 0;

background-color: #f4f4f4;

color: #333;

}

header {

background-color: #4CAF50;

color: white;

padding: 20px;

text-align: center;

}

nav {

background-color: #333;

color: white;

padding: 15px;

text-align: center;

}

nav ul {

list-style: none;

padding: 0;

}

nav ul li {

display: inline;

margin: 0 10px;

}

nav ul li a {

color: white;

text-decoration: none;

font-size: 18px;

}

nav ul li a:hover {

text-decoration: underline;

}

main {

margin: 20px;

}

h2 {

color: #4CAF50;

}

ul {

list-style: none;

padding: 0;

}

ul li {

background-color: white;

margin-bottom: 10px;

padding: 10px;

border-radius: 5px;

}

button {

background-color: #4CAF50;

color: white;

padding: 10px 20px;

border: none;

cursor: pointer;

}

button:hover {

background-color: #45a049;

}

footer {

background-color: #333;

color: white;

text-align: center;

padding: 10px;

position: absolute;

width: 100%;

bottom: 0;

}

1. **Run Migrations and Start the Server:**

A black rectangular object with a black border

Description automatically generated

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

**Summary**

This Django-based CMS allows healthcare professionals to manage, upload, and edit documentary reports, with a clean and responsive front end. The CSS provides a professional layout, and Django handles all backend CRUD operations.

Creating a **Healthcare Report Documentary Content Management System (CMS)** involves several steps, from backend logic and database setup to frontend design, including CSS for styles. I'll walk you through a simple implementation of such a CMS in Python using the Flask framework for the backend, HTML for the frontend, and CSS for styling.

**Step 1: Project Structure**

Here’s a typical project structure for a CMS:

A screenshot of a computer program

Description automatically generated

**Step 2: Backend (Flask Setup)**

**Install Flask:**

A black rectangular object with a black stripe

Description automatically generated

**app/init.py**:

A screenshot of a computer

Description automatically generated

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///db.sqlite3'

app.config['SECRET\_KEY'] = 'supersecretkey'

db = SQLAlchemy(app)

from app import routes

**app/models.py** (Database models for reports):

A screen shot of a computer

Description automatically generated

from app import db

class Report(db.Model):

id = db.Column(db.Integer, primary\_key=True)

title = db.Column(db.String(100), nullable=False)

description = db.Column(db.Text, nullable=False)

date\_created = db.Column(db.DateTime, default=db.func.now())

**app/routes.py** (Routes/Views):

A screenshot of a computer program

Description automatically generated

from flask import render\_template, redirect, url\_for, request

from app import app, db

from app.models import Report

@app.route('/')

def index():

reports = Report.query.all()

return render\_template('index.html', reports=reports)

@app.route('/report/<int:id>')

def report(id):

report = Report.query.get\_or\_404(id)

return render\_template('report.html', report=report)

@app.route('/add', methods=['GET', 'POST'])

def add\_report():

if request.method == 'POST':

title = request.form['title']

description = request.form['description']

new\_report = Report(title=title, description=description)

db.session.add(new\_report)

db.session.commit()

return redirect(url\_for('index'))

return render\_template('add\_report.html')

**run.py** (Run the app):

A black rectangular object with a black border

Description automatically generated

from app import app, db

db.create\_all()

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 3: Frontend (HTML Templates)**

**app/templates/base.html** (Base Layout):

A screenshot of a computer program

Description automatically generated

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Healthcare Report CMS</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/styles.css') }}">

</head>

<body>

<header>

<h1>Healthcare Report Documentary CMS</h1>

<nav>

<a href="{{ url\_for('index') }}">Home</a>

<a href="{{ url\_for('add\_report') }}">Add Report</a>

</nav>

</header>

<main>

{% block content %}{% endblock %}

</main>

<footer>

<p>© 2024 Healthcare CMS</p>

</footer>

</body>

</html>

**app/templates/index.html** (Home Page):

A screenshot of a computer program

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h2>Reports</h2>

<ul>

{% for report in reports %}

<li><a href="{{ url\_for('report', id=report.id) }}">{{ report.title }}</a> - {{ report.date\_created.strftime('%Y-%m-%d') }}</li>

{% endfor %}

</ul>

{% endblock %}

**app/templates/report.html** (Individual Report Page):

A screenshot of a computer program

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h2>{{ report.title }}</h2>

<p>{{ report.description }}</p>

<p><small>Created on: {{ report.date\_created.strftime('%Y-%m-%d') }}</small></p>

{% endblock %}

**app/templates/add\_report.html** (Add New Report):

A screen shot of a computer program

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h2>Add New Report</h2>

<form action="{{ url\_for('add\_report') }}" method="POST">

<label for="title">Title:</label>

<input type="text" name="title" id="title" required>

<label for="description">Description:</label>

<textarea name="description" id="description" required></textarea>

<button type="submit">Submit</button>

</form>

{% endblock %}

**Step 4: Static Files (CSS)**

**app/static/css/styles.css**:

A screenshot of a computer program

Description automatically generated

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

background-color: #f4f4f4;

}

header {

background-color: #007BFF;

color: white;

padding: 15px 0;

text-align: center;

}

header h1 {

margin: 0;

font-size: 2rem;

}

nav {

margin-top: 10px;

}

nav a {

color: white;

text-decoration: none;

margin: 0 15px;

font-size: 1.2rem;

}

nav a:hover {

text-decoration: underline;

}

main {

padding: 20px;

}

h2 {

color: #333;

font-size: 1.8rem;

}

form {

background-color: white;

padding: 20px;

border-radius: 5px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

form label {

display: block;

margin-bottom: 8px;

font-weight: bold;

}

form input, form textarea {

width: 100%;

padding: 10px;

margin-bottom: 20px;

border: 1px solid #ccc;

border-radius: 4px;

}

form button {

background-color: #007BFF;

color: white;

border: none;

padding: 10px 15px;

cursor: pointer;

}

form button:hover {

background-color: #0056b3;

}

footer {

text-align: center;

padding: 20px;

background-color: #007BFF;

color: white;

position: fixed;

bottom: 0;

width: 100%;

}

**Step 5: Running the Application**

1. **Initialize the database**: Run the following to initialize the SQLite database and tables:

A black rectangle with a black stripe

Description automatically generatedpython run.py

1. **Run the application**: Open a terminal and run:

A black and grey rectangular object

Description automatically generatedflask run

1. **Access the application**:  
   Open your browser and go to http://127.0.0.1:5000/ to access your healthcare report CMS.

This is a basic CMS with a simple structure, database models, routes, HTML templates, and styling using CSS. You can further extend it by adding authentication, authorization, more fields for reports, and other features like file uploads or advanced search functionality.