Healthcare Report Documentary Content Management System Application Development

Developing a Healthcare Report Documentary Content Management System (CMS) involves creating a system where healthcare professionals can manage, upload, edit, and review healthcare reports. I'll outline a basic structure using Python's Flask framework, which includes user authentication, report management, and the ability to upload and store documents.

**Healthcare Report CMS in Python using Flask**

**1. Setup the Flask Project**

Ensure Flask and required packages are installed:

A black and white screen

Description automatically generated

pip install flask flask\_sqlalchemy flask\_wtf werkzeug flask\_login

2. **Project Structure:**

A screenshot of a computer

Description automatically generated

healthcare\_cms/

│

├── app.py

├── models.py

├── forms.py

├── templates/

│ ├── base.html

│ ├── home.html

│ ├── login.html

│ ├── dashboard.html

│ ├── upload\_report.html

│ └── view\_report.html

├── static/

│ └── styles.css

├── uploads/

└── reports.db

3. **models.py - Database Models:**

A computer screen shot of a program

Description automatically generated

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from flask\_login import UserMixin

db = SQLAlchemy()

class User(UserMixin, db.Model):

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

username = db.Column(db.String(150), unique=True, nullable=False)

password = db.Column(db.String(150), nullable=False)

def set\_password(self, password):

self.password = generate\_password\_hash(password)

def check\_password(self, password):

return check\_password\_hash(self.password, password)

class Report(db.Model):

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

title = db.Column(db.String(200), nullable=False)

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

filename = db.Column(db.String(150), nullable=False)

uploaded\_at = db.Column(db.DateTime, nullable=False, default=db.func.current\_timestamp())

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

user = db.relationship('User', backref=db.backref('reports', lazy=True))

4. **forms.py - Flask-WTF Forms**

A computer screen shot of a program

Description automatically generated

from flask\_wtf import FlaskForm

from wtforms import StringField, PasswordField, TextAreaField, FileField, SubmitField

from wtforms.validators import DataRequired, Length

class LoginForm(FlaskForm):

username = StringField('Username', validators=[DataRequired()])

password = PasswordField('Password', validators=[DataRequired()])

submit = SubmitField('Login')

class ReportForm(FlaskForm):

title = StringField('Title', validators=[DataRequired(), Length(min=5, max=200)])

description = TextAreaField('Description', validators=[DataRequired()])

file = FileField('Upload Report', validators=[DataRequired()])

submit = SubmitField('Submit')

**5. app.py - Main Flask Application:**

A screenshot of a computer program

Description automatically generated

from flask import Flask, render\_template, redirect, url\_for, request, flash, send\_from\_directory

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import LoginManager, login\_user, login\_required, logout\_user, current\_user

from werkzeug.utils import secure\_filename

import os

from models import db, User, Report

from forms import LoginForm, ReportForm

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

app.config['SECRET\_KEY'] = 'your\_secret\_key'

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

app.config['UPLOAD\_FOLDER'] = 'uploads/'

db.init\_app(app)

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

@login\_manager.user\_loader

def load\_user(user\_id):

return User.query.get(int(user\_id))

@app.route('/')

def home():

reports = Report.query.all()

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

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

def login():

form = LoginForm()

if form.validate\_on\_submit():

user = User.query.filter\_by(username=form.username.data).first()

if user and user.check\_password(form.password.data):

login\_user(user)

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

else:

flash('Invalid username or password')

return render\_template('login.html', form=form)

@app.route('/logout')

@login\_required

def logout():

logout\_user()

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

@app.route('/dashboard')

@login\_required

def dashboard():

reports = Report.query.filter\_by(user\_id=current\_user.id).all()

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

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

@login\_required

def upload\_report():

form = ReportForm()

if form.validate\_on\_submit():

filename = secure\_filename(form.file.data.filename)

form.file.data.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

report = Report(title=form.title.data, description=form.description.data,

filename=filename, user\_id=current\_user.id)

db.session.add(report)

db.session.commit()

flash('Report uploaded successfully')

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

return render\_template('upload\_report.html', form=form)

@app.route('/view/<int:report\_id>')

def view\_report(report\_id):

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

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

@app.route('/download/<filename>')

def download\_file(filename):

return send\_from\_directory(app.config['UPLOAD\_FOLDER'], filename)

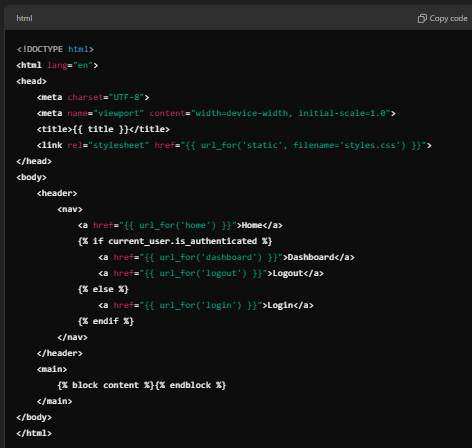
if \_\_name\_\_ == '\_\_main\_\_':

with app.app\_context():

db.create\_all()

app.run(debug=True)

**6. Templates:**

* **base.html:**
* 

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>{{ title }}</title>

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

</head>

<body>

<header>

<nav>

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

{% if current\_user.is\_authenticated %}

<a href="{{ url\_for('dashboard') }}">Dashboard</a>

<a href="{{ url\_for('logout') }}">Logout</a>

{% else %}

<a href="{{ url\_for('login') }}">Login</a>

{% endif %}

</nav>

</header>

<main>

{% block content %}{% endblock %}

</main>

</body>

</html>

* home.html:

A computer screen shot of a program code

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h1>Healthcare Reports</h1>

{% for report in reports %}

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

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

<a href="{{ url\_for('view\_report', report\_id=report.id) }}">View</a> |

<a href="{{ url\_for('download\_file', filename=report.filename) }}">Download</a>

{% endfor %}

{% endblock %}

* login.html:

A computer screen shot of a computer code

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h1>Login</h1>

<form method="POST">

{{ form.hidden\_tag() }}

<label for="username">{{ form.username.label }}</label>

{{ form.username }}

<label for="password">{{ form.password.label }}</label>

{{ form.password }}

{{ form.submit }}

</form>

{% endblock %}

* dashboard.html:

A screenshot of a computer program

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h1>Dashboard</h1>

<a href="{{ url\_for('upload\_report') }}">Upload New Report</a>

{% for report in reports %}

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

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

<a href="{{ url\_for('view\_report', report\_id=report.id) }}">View</a> |

<a href="{{ url\_for('download\_file', filename=report.filename) }}">Download</a>

{% endfor %}

{% endblock %}

* upload\_report.html:

A screen shot of a computer

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h1>Upload Report</h1>

<form method="POST" enctype="multipart/form-data">

{{ form.hidden\_tag() }}

<label for="title">{{ form.title.label }}</label>

{{ form.title }}

<label for="description">{{ form.description.label }}</label>

{{ form.description }}

<label for="file">{{ form.file.label }}</label>

{{ form.file }}

{{ form.submit }}

</form>

{% endblock %}

* view\_report.html:

A screen shot of a computer program

Description automatically generated

{% extends 'base.html' %}

{% block content %}

<h1>{{ report.title }}</h1>

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

<a href="{{ url\_for('download\_file', filename=report.filename) }}">Download</a>

{% endblock %}

**7. Static Files**

You can add your custom styles in styles.css under the static/ directory.It can be optional

For the developer. To create a well-organized healthcare report documentary content management system (CMS), the CSS should handle multiple aspects such as layout, responsiveness, typography, and theme consistency. Here’s an example of a comprehensive styles.css for a healthcare report CMS. This CSS is aimed at creating a clean, professional, and responsive design for a documentary report application.

**styles.css**

A screen shot of a computer program

Description automatically generated

Source code:

/\* General Reset \*/

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

/\* Body Styling \*/

body {

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

background-color: #f4f4f4;

color: #333;

line-height: 1.6;

}

/\* Container \*/

.container {

max-width: 1200px;

margin: 0 auto;

padding: 20px;

}

/\* Header Styles \*/

header {

background-color: #4CAF50;

color: white;

padding: 20px 0;

text-align: center;

border-bottom: 4px solid #333;

}

header h1 {

font-size: 36px;

margin-bottom: 10px;

}

header p {

font-size: 16px;

font-weight: 300;

}

/\* Navigation Bar \*/

nav {

background: #333;

color: white;

padding: 10px 0;

text-align: center;

}

nav ul {

list-style: none;

}

nav ul li {

display: inline;

margin: 0 15px;

}

nav ul li a {

color: white;

text-decoration: none;

font-size: 18px;

}

nav ul li a:hover {

text-decoration: underline;

}

/\* Main Content Styling \*/

main {

display: flex;

flex-direction: row;

gap: 20px;

margin-top: 20px;

}

/\* Sidebar Styling \*/

.sidebar {

width: 25%;

background-color: #f9f9f9;

padding: 20px;

border: 1px solid #ddd;

}

.sidebar h3 {

font-size: 22px;

margin-bottom: 15px;

}

.sidebar ul {

list-style: none;

}

.sidebar ul li {

margin-bottom: 10px;

}

.sidebar ul li a {

color: #333;

text-decoration: none;

}

.sidebar ul li a:hover {

color: #4CAF50;

}

/\* Main Content Area \*/

.content {

width: 75%;

background-color: white;

padding: 20px;

border: 1px solid #ddd;

}

.content h2 {

font-size: 28px;

margin-bottom: 15px;

color: #333;

}

.content p {

font-size: 18px;

margin-bottom: 20px;

line-height: 1.6;

}

.content img {

max-width: 100%;

height: auto;

margin-bottom: 20px;

}

.content blockquote {

font-size: 20px;

font-style: italic;

background-color: #f9f9f9;

padding: 15px;

border-left: 5px solid #4CAF50;

margin-bottom: 20px;

}

/\* Button Styles \*/

button {

background-color: #4CAF50;

color: white;

padding: 10px 20px;

border: none;

cursor: pointer;

font-size: 16px;

transition: background-color 0.3s;

}

button:hover {

background-color: #45a049;

}

/\* Table Styling \*/

table {

width: 100%;

border-collapse: collapse;

margin-bottom: 20px;

}

table, th, td {

border: 1px solid #ddd;

}

th, td {

padding: 12px;

text-align: left;

}

th {

background-color: #4CAF50;

color: white;

}

td {

background-color: #f9f9f9;

}

/\* Form Styles \*/

form {

margin-bottom: 20px;

}

form input[type="text"],

form input[type="email"],

form textarea {

width: 100%;

padding: 10px;

margin-bottom: 10px;

border: 1px solid #ddd;

font-size: 16px;

}

form input[type="submit"] {

background-color: #4CAF50;

color: white;

padding: 10px 15px;

border: none;

cursor: pointer;

}

form input[type="submit"]:hover {

background-color: #45a049;

}

/\* Footer Styling \*/

footer {

background-color: #333;

color: white;

text-align: center;

padding: 20px;

margin-top: 20px;

}

footer p {

font-size: 16px;

}

/\* Media Queries for Responsive Design \*/

@media (max-width: 768px) {

nav ul li {

display: block;

margin-bottom: 10px;

}

main {

flex-direction: column;

}

.sidebar, .content {

width: 100%;

}

}

**Key Components:**

1. **Body and Container**:
   * Sets up a base font and body background.
   * Uses .container for consistent centering of content.
2. **Header and Navigation**:
   * A fixed header and nav bar with prominent typography.
   * Hover effects for interactive elements.
3. **Main Content and Sidebar**:
   * A flexible grid layout using flexbox to keep the content dynamic.
   * Sidebar styled separately for reports or filtering.
4. **Content Area**:
   * Clean styling for text, images, block quotes, and tables.
   * blockquote is used for quotes in reports or documentary sections.
5. **Buttons and Forms**:
   * Standardized button styling with hover effects.
   * Form fields that allow for structured report submissions.
6. **Table Styling**:
   * Consistent and readable table layouts for healthcare data.
7. **Responsive Design**:
   * A media query ensures the layout adapts to mobile screens by changing navigation and making the layout single-column.

**Next Steps:**

* Implement this styles.css into your HTML structure.
* Customize based on the CMS's specific requirements such as report formats, document uploads, or healthcare data integration.
* **8. Running the Application**
* To start the application, run:

A black rectangular object with a black stripe

Description automatically generatedpython app.py

**Explanation:**

1. **Models:**
   * **User**: Manages user authentication.
   * **Report**: Stores healthcare report details such as title, description, file name, and upload time.
2. **Forms:**
   * **LoginForm**: Handles user login.
   * **ReportForm**: Used for uploading and managing healthcare reports.
3. **Views:**
   * **Home**: Displays all reports.
   * **Login**: Allows users to log in.
   * **Dashboard**: Displays user-specific reports and upload options.
   * **Upload Report**: Enables users to upload new reports.
   * **View Report**: Allows users to view details of a report.
   * **Download File**: Allows users to download report files.

This is a basic Healthcare Report Documentary CMS. You can further extend it with additional features like user roles (admin, doctor, etc.), report categorization, or integrating a more complex document management system.

Healthcare Report Documentry Content Management System Application Development

Creating a healthcare report documentary Content Management System (CMS) for Windows 10 can involve building a more specialized desktop application with functionalities tailored for healthcare document management, such as uploading, viewing, editing, and managing healthcare reports.

Here’s a basic outline and source code for a simple healthcare CMS desktop application using **Python**, **Tkinter** for the GUI, and **SQLite** for the database, specifically built for Windows 10. This project will allow a user to:

 Log in to the system.

 Upload healthcare reports.

 View, edit, and delete reports.

 Export reports as PDF.

Project Structure

A screenshot of a computer

Description automatically generated

healthcare\_cms/

│

├── app.py

├── templates/

│ ├── home.html

│ ├── add\_report.html

│ ├── edit\_report.html

│ └── report\_list.html

├── reports.db

├── reports/

│ └── (uploaded reports are stored here)

└── static/

└── styles.css

**Step 1: Python Libraries**

Make sure you have the following libraries installed. You can install them using pip:

A screen shot of a computer

Description automatically generated

pip install flask

pip install flask\_sqlalchemy

pip install reportlab # For generating PDFs

Step 2: app.py - Main Application: A computer screen shot of a black screen

Description automatically generated

import os

from flask import Flask, render\_template, request, redirect, url\_for, flash, send\_from\_directory

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.utils import secure\_filename

from reportlab.lib.pagesizes import letter

from reportlab.pdfgen import canvas

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

app.config['SECRET\_KEY'] = 'your\_secret\_key'

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

app.config['UPLOAD\_FOLDER'] = 'reports'

db = SQLAlchemy(app)

# Define the Report model

class Report(db.Model):

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

patient\_name = db.Column(db.String(100), nullable=False)

doctor\_name = db.Column(db.String(100), nullable=False)

diagnosis = db.Column(db.String(200), nullable=False)

report\_file = db.Column(db.String(200), nullable=True)

# Home page to display all reports

@app.route('/')

def home():

reports = Report.query.all()

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

# Upload a new healthcare report

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

def add\_report():

if request.method == 'POST':

patient\_name = request.form['patient\_name']

doctor\_name = request.form['doctor\_name']

diagnosis = request.form['diagnosis']

report\_file = request.files['report\_file']

if report\_file:

filename = secure\_filename(report\_file.filename)

report\_file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

new\_report = Report(patient\_name=patient\_name, doctor\_name=doctor\_name, diagnosis=diagnosis, report\_file=filename)

else:

new\_report = Report(patient\_name=patient\_name, doctor\_name=doctor\_name, diagnosis=diagnosis)

db.session.add(new\_report)

db.session.commit()

flash('Report added successfully')

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

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

# Edit an existing report

@app.route('/edit\_report/<int:report\_id>', methods=['GET', 'POST'])

def edit\_report(report\_id):

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

if request.method == 'POST':

report.patient\_name = request.form['patient\_name']

report.doctor\_name = request.form['doctor\_name']

report.diagnosis = request.form['diagnosis']

if 'report\_file' in request.files:

report\_file = request.files['report\_file']

if report\_file:

filename = secure\_filename(report\_file.filename)

report\_file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

report.report\_file = filename

db.session.commit()

flash('Report updated successfully')

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

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

# Delete a report

@app.route('/delete\_report/<int:report\_id>')

def delete\_report(report\_id):

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

if report.report\_file:

os.remove(os.path.join(app.config['UPLOAD\_FOLDER'], report.report\_file))

db.session.delete(report)

db.session.commit()

flash('Report deleted successfully')

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

# View report as PDF

@app.route('/generate\_pdf/<int:report\_id>')

def generate\_pdf(report\_id):

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

pdf\_file\_path = f'reports/{report.patient\_name}\_report.pdf'

# Generate PDF using ReportLab

c = canvas.Canvas(pdf\_file\_path, pagesize=letter)

c.drawString(100, 750, f"Patient Name: {report.patient\_name}")

c.drawString(100, 730, f"Doctor Name: {report.doctor\_name}")

c.drawString(100, 710, f"Diagnosis: {report.diagnosis}")

c.save()

return send\_from\_directory(directory='reports', filename=f'{report.patient\_name}\_report.pdf')

# Run the application

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

app.run(debug=True)

**Step 3: HTML Templates**

1. **home.html**

A screenshot of a computer program

Description automatically generated

<!DOCTYPE html>

<html>

<head>

<title>Healthcare Reports</title>

</head>

<body>

<h1>Healthcare Reports</h1>

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

<table>

<tr>

<th>Patient Name</th>

<th>Doctor Name</th>

<th>Diagnosis</th>

<th>Actions</th>

</tr>

{% for report in reports %}

<tr>

<td>{{ report.patient\_name }}</td>

<td>{{ report.doctor\_name }}</td>

<td>{{ report.diagnosis }}</td>

<td>

<a href="{{ url\_for('edit\_report', report\_id=report.id) }}">Edit</a> |

<a href="{{ url\_for('delete\_report', report\_id=report.id) }}" onclick="return confirm('Are you sure?')">Delete</a> |

<a href="{{ url\_for('generate\_pdf', report\_id=report.id) }}">View as PDF</a>

</td>

</tr>

{% endfor %}

</table>

</body>

</html>

1. add\_report.html

A screenshot of a computer program

Description automatically generated

<!DOCTYPE html>

<html>

<head>

<title>Add Report</title>

</head>

<body>

<h1>Add Healthcare Report</h1>

<form method="POST" enctype="multipart/form-data">

<label for="patient\_name">Patient Name:</label>

<input type="text" name="patient\_name" required>

<br>

<label for="doctor\_name">Doctor Name:</label>

<input type="text" name="doctor\_name" required>

<br>

<label for="diagnosis">Diagnosis:</label>

<input type="text" name="diagnosis" required>

<br>

<label for="report\_file">Upload Report:</label>

<input type="file" name="report\_file">

<br>

<button type="submit">Add Report</button>

</form>

</body>

</html>

**Step 4: Running the Application**

1. Create the database and tables:

A black rectangular object with a black border

Description automatically generated

from app import db

db.create\_all()

1. Start the Flask application:

A black rectangular object with a black stripe

Description automatically generated

python app.py

Access the application by opening a browser and navigating to <http://127.0.0.1:5000/>.

**Features**

1. **Healthcare Reports CRUD:**
   * Add new healthcare reports.
   * Edit existing reports.
   * Delete reports.
   * Upload additional report documents as files.
   * Generate reports as PDFs.
2. **PDF Generation:**
   * Uses ReportLab to create a simple PDF version of the report that can be viewed or downloaded.
3. **Security Considerations:**
   * Ensure proper handling of uploaded files to avoid malicious uploads. You might want to restrict the file types or sanitize the filenames.
   * Implement more robust user authentication and authorization in a real-world system.

This is a foundational structure that can be expanded with more features such as report search, advanced user roles (doctors, nurses, administrators), or integration with external medical databases.