

Technical Workflow

1 Home page :

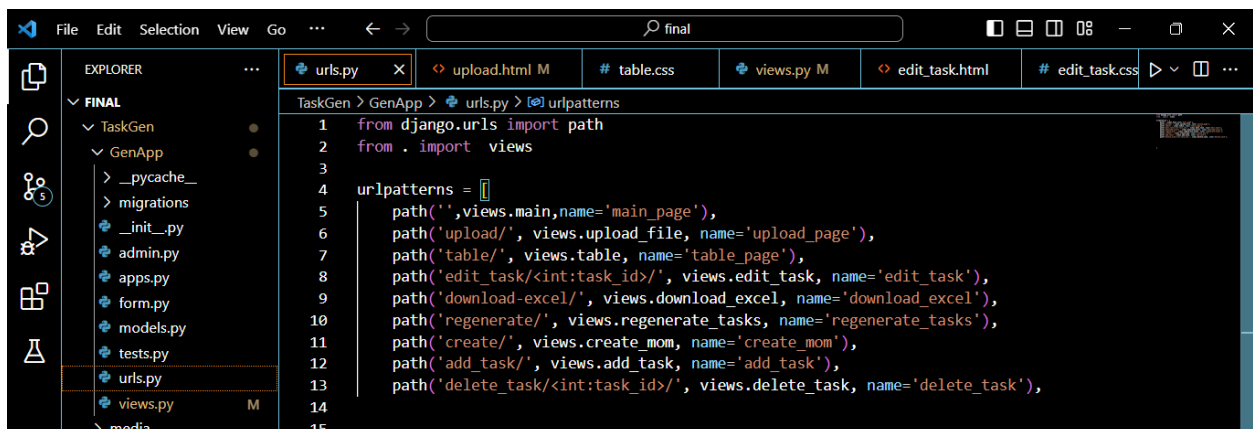
The home created using Django html template and CSS

Template:



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>GenTask</title>
6   {% load static %}
7   <link rel="stylesheet" type="text/css" href="{% static 'css/main.css' %}">
8
9 </head>
10 <body>
11   <div class="container">
12     <h1 class="heading">Tasks from Meeting Minutes</h1>
13
14     <p class="description">This Website is integrated with a Generative AI model capable of
15 processing and understanding text data.You can input the meeting minutes in a text format, either by
16 uploading a file or entering text directly.</p>
17
18     <button class="upload-btn"><a href="{% url 'upload_page' %}" class='link'>Upload File</a></button>
19 <br/>
20 <br/>
21     <button class="create-btn"><a href="{% url 'create_mom' %}" class="link">Create MOM</a></button>
22
23   </div>
24 </body>
25 </html>
```

Urls:

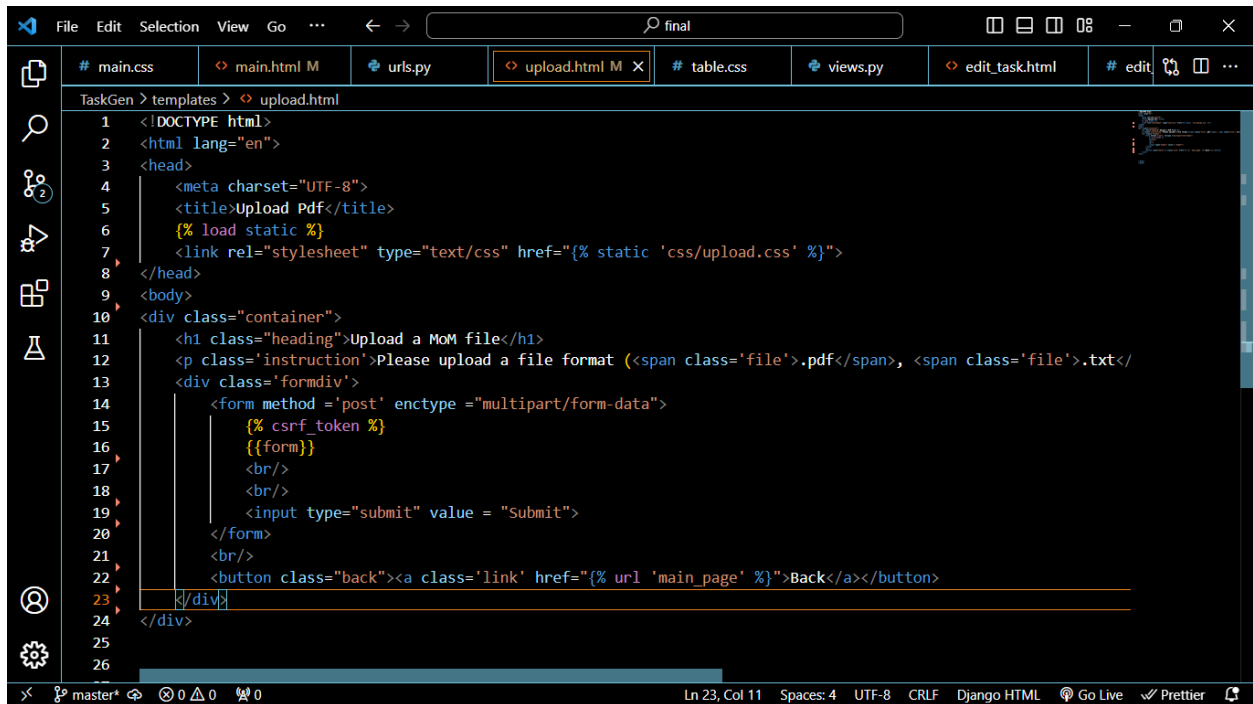


```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.main, name='main_page'),
6     path('upload/', views.upload_file, name='upload_page'),
7     path('table/', views.table, name='table_page'),
8     path('edit_task/<int:task_id>', views.edit_task, name='edit_task'),
9     path('download-excel/', views.download_excel, name='download_excel'),
10    path('regenerate/', views.regenerate_tasks, name='regenerate_tasks'),
11    path('create/', views.create_mom, name='create_mom'),
12    path('add_task/', views.add_task, name='add_task'),
13    path('delete_task/<int:task_id>', views.delete_task, name='delete_task'),
14
15 ]
```

2 Meeting Minutes Input:

2.1 File input:

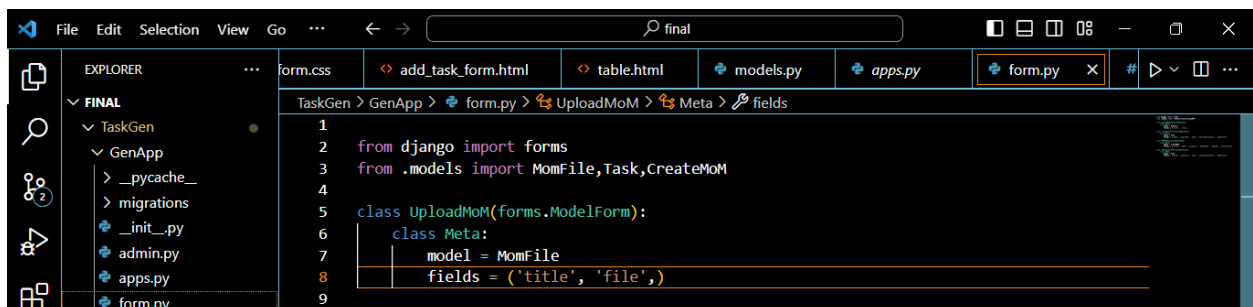
The file input page was created using Django form , html template, and model



The screenshot shows a code editor with the following HTML code in the file `upload.html`:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Upload Pdf</title>
6   {% load static %}
7   <link rel="stylesheet" type="text/css" href="{% static 'css/upload.css' %}">
8 </head>
9 <body>
10 <div class="container">
11   <h1 class="heading">Upload a MoM file</h1>
12   <p class="instruction">Please upload a file format (<span class='file'>.pdf</span>, <span class='file'>.txt</span>
13   <div class="formdiv">
14     <form method="post" enctype="multipart/form-data">
15       {% csrf_token %}
16       {{ form }}
17     <br/>
18     <br/>
19     <input type="submit" value="Submit">
20   </form>
21   <br/>
22   <button class="back"><a class="link" href="{% url 'main_page' %}">Back</a></button>
23 </div>
24 </div>
25
26
```


Form :



The screenshot shows a code editor with the following Python code in the file `form.py`:

```
1
2 from django import forms
3 from .models import MomFile, Task, CreateMoM
4
5 class UploadMoM(forms.ModelForm):
6     class Meta:
7         model = MomFile
8         fields = ('title', 'file',)
9
```

Model:



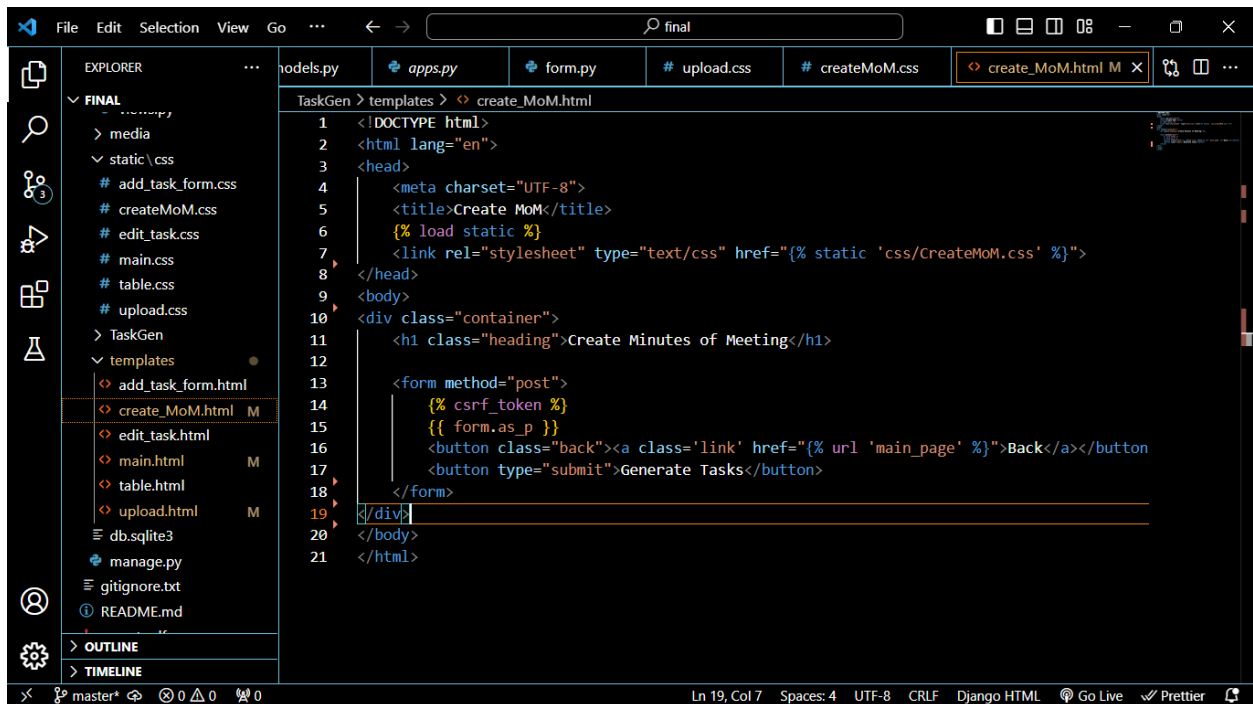
The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a project structure with folders 'FINAL', 'TaskGen', and 'GenApp'. Under 'GenApp', there are files: '_pycache_', 'migrations', '_init_.py', 'admin.py', 'apps.py', 'form.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'models.py' file is selected. The main editor area shows the code for 'models.py' with the following content:

```
1 from django.db import models
2
3 # Create your models here.
4 class MomFile(models.Model):
5
6     title = models.CharField(max_length = 80)
7     file = models.FileField(upload_to='files/')
8
9     class Meta:
10         ordering = ['title']
11
12     def __str__(self):
13         return f"{self.title}"
14
```

2.2 Text input:

By using django form,html template and CSS we can create a MoM input by entering text

Template:



The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a project structure with folders 'FINAL', 'TaskGen', and 'GenApp'. Under 'GenApp', there are files: '_pycache_', 'migrations', '_init_.py', 'admin.py', 'apps.py', 'form.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'models.py' file is selected. The main editor area shows the code for 'create_MoM.html' with the following content:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Create MoM</title>
6     {% load static %}
7     <link rel="stylesheet" type="text/css" href="{% static 'css/CreateMoM.css' %}">
8 </head>
9 <body>
10 <div class="container">
11     <h1 class="heading">Create Minutes of Meeting</h1>
12
13     <form method="post">
14         {% csrf_token %}
15         {{ form.as_p }}
16         <button class="back"><a class='link' href="{% url 'main_page' %}">Back</a></button>
17         <button type="submit">Generate Tasks</button>
18     </form>
19 </div>
20 </body>
21 </html>
```

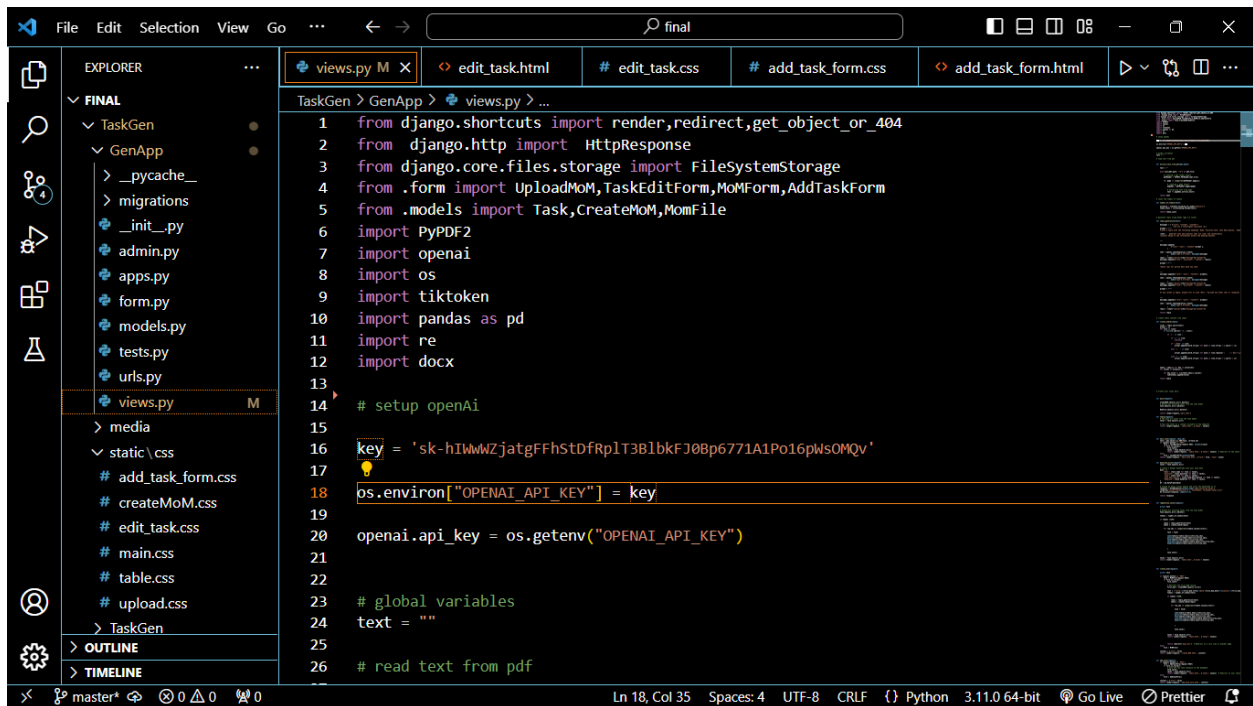
Form:

```
views.py 14
> media 15
static\css 16
# add_task_form.css 17
# createMoM.css 18
# edit_task.css 19

class MoMForm(forms.ModelForm):
    class Meta:
        model = CreateMoM
        fields = ['title', 'date', 'location', 'attendees', 'agenda', 'discussion']
```

3 Generative AI Integration:

We integrate ChatGPT 3.5 turbo using OpenAI library

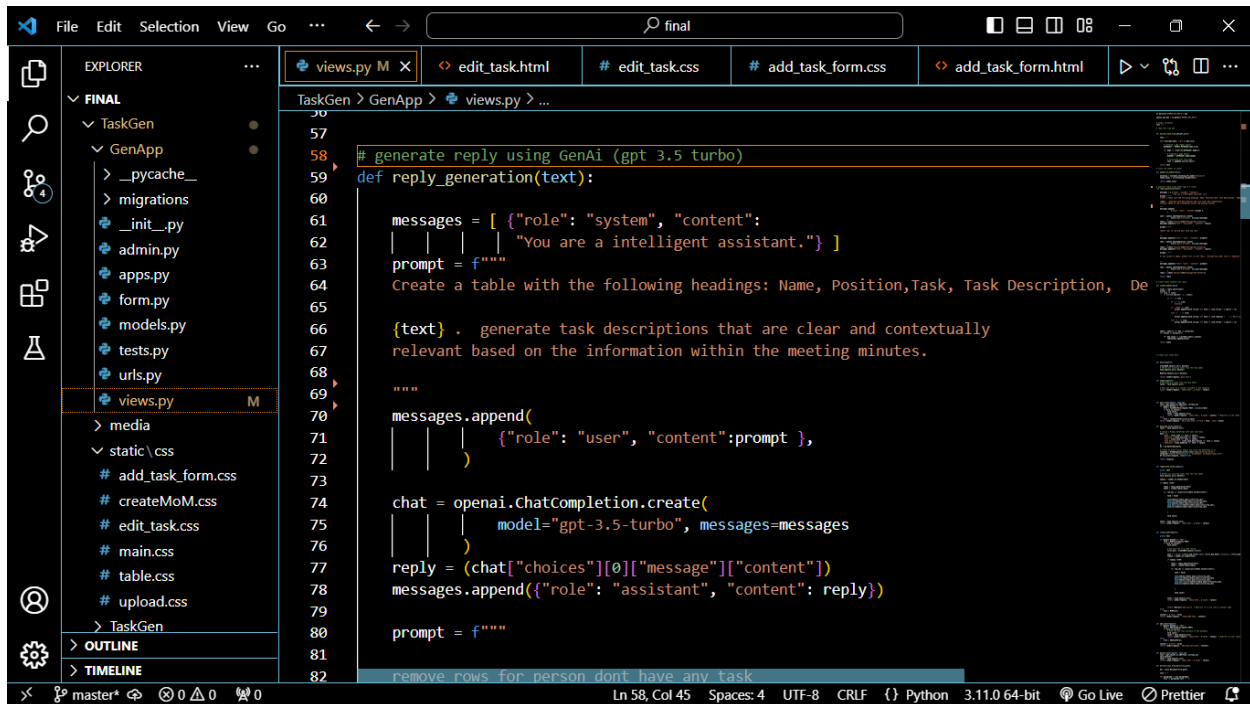


The screenshot shows a Visual Studio Code editor window with a Django project named 'TaskGen'. The Explorer sidebar on the left shows the project structure, including 'TaskGen', 'GenApp', and various files like 'views.py', 'models.py', and 'urls.py'. The main editor area displays the 'views.py' file, which contains the following code:

```
1 from django.shortcuts import render, redirect, get_object_or_404
2 from django.http import HttpResponse
3 from django.core.files.storage import FileSystemStorage
4 from .form import UploadMoM, TaskEditForm, MoMForm, AddTaskForm
5 from .models import Task, CreateMoM, MoMFile
6 import PyPDF2
7 import openai
8 import os
9 import tiktoken
10 import pandas as pd
11 import re
12 import docx
13
14 # setup openai
15 key = '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'
16
17 os.environ["OPENAI_API_KEY"] = key
18
19 openai.api_key = os.getenv("OPENAI_API_KEY")
20
21 # global variables
22 text = ""
23
24 # read text from pdf
```

The status bar at the bottom indicates the current file is 'views.py' at line 18, column 35, with a UTF-8 encoding and CRLF line endings. It also shows the Python version as 3.11.0 64-bit and the use of the Prettier formatter.

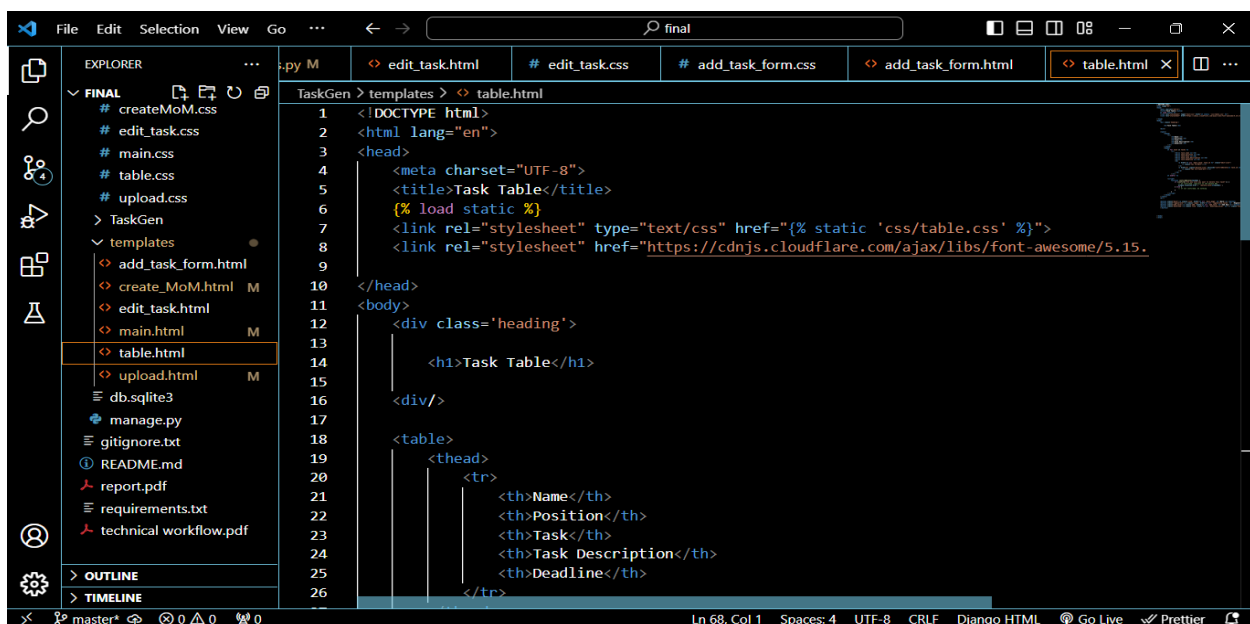
3.1 Text Understanding:



```
57
58 # generate reply using GenAI (gpt 3.5 turbo)
59 def reply_generation(text):
60
61     messages = [ {"role": "system", "content":
62     | | | | "You are a intelligent assistant."} ]
63     prompt = f"""
64     Create a table with the following headings: Name, Position,Task, Task Description, De
65
66     {text} . generate task descriptions that are clear and contextually
67     relevant based on the information within the meeting minutes.
68
69     """
70     messages.append(
71     | | | | {"role": "user", "content":prompt },
72     )
73
74     chat = openai.ChatCompletion.create(
75     | | | | model="gpt-3.5-turbo", messages=messages
76     )
77     reply = (chat["choices"][0]["message"]["content"])
78     messages.append({"role": "assistant", "content": reply})
79
80     prompt = f"""
81
82     remove rows for person dont have any task
```

3.2 Action Item Generation:

Using Django model, html and CSS we can create table for task



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Task Table</title>
6     {% load static %}
7     <link rel="stylesheet" type="text/css" href="{% static 'css/table.css' %}">
8     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.
9
10 </head>
11 <body>
12     <div class='heading'>
13
14         <h1>Task Table</h1>
15
16     </div>
17
18     <table>
19         <thead>
20             <tr>
21                 <th>Name</th>
22                 <th>Position</th>
23                 <th>Task</th>
24                 <th>Task Description</th>
25                 <th>Deadline</th>
26             </tr>
```

```
File Edit Selection View Go ... final
EXPLORER
FINAL
# createMoM.css
# edit_task.css
# main.css
# table.css
# upload.css
TaskGen
templates
add_task_form.html
create_MoM.html M
edit_task.html
main.html M
table.html
upload.html M
db.sqlite3
manage.py
gitignore.txt
README.md
report.pdf
requirements.txt
technical workflow.pdf
OUTLINE
TIMELINE
TaskGen > templates > table.html
28 <tbody>
29     {% for task in tasks %}
30     <tr>
31         <td>{{ task.name }}</td>
32         <td>{{ task.position }}</td>
33         <td>{{ task.task }}</td>
34         <td>{{ task.task_description }}</td>
35         <td>{{ task.deadline }}</td>
36     </tr>
37         <a href="{% url 'edit_task' task.id %}" class="edit-icon">
38             <i class="fas fa-edit"></i>
39         </a>
40         <a href="#" class="delete-icon" onclick="confirmDelete({{ task.id
41             <i class="fas fa-trash-alt"></i>
42         </a>
43     </td>
44 </tr>
45 {% endfor %}
46
47 <script>
48     function confirmDelete(taskId) {
49         if (confirm('Are you sure you want to delete this task?')) {
50             // If confirmed, redirect to the delete URL
51             window.location.href = `/delete_task/${taskId}/`;
52         } else {
53             // If not confirmed, do nothing
54         }
55     }
56 </script>
57
58 </tbody>
59 </table>
60 <br/>
61 <button class="back"><a class="link" href="{% url 'main_page' %}">Back</a></button>
62 <button class="regenerate"><a class="regen" href="{% url 'regenerate_tasks' %}">Regen
63 <button class="add_task"><a class="add" href="{% url 'add_task' %}">Add Row</a></butto
64 <button class="download"><a class="link" href="{% url 'download_excel' %}"><i class="f
65 </button>
66
67
68
69
70 </body>
71 </html>
72
Ln 68, Col 1 Spaces: 4 UTF-8 CRLF Django HTML Go Live Prettier
```

```
File Edit Selection View Go ... final
EXPLORER
FINAL
# createMoM.css
# edit_task.css
# main.css
# table.css
# upload.css
TaskGen
templates
add_task_form.html
create_MoM.html M
edit_task.html
main.html M
table.html
upload.html M
db.sqlite3
manage.py
gitignore.txt
README.md
report.pdf
requirements.txt
technical workflow.pdf
OUTLINE
TIMELINE
TaskGen > templates > table.html
46 <script>
47     function confirmDelete(taskId) {
48         if (confirm('Are you sure you want to delete this task?')) {
49             // If confirmed, redirect to the delete URL
50             window.location.href = `/delete_task/${taskId}/`;
51         } else {
52             // If not confirmed, do nothing
53         }
54     }
55 </script>
56 </tbody>
57 </table>
58 <br/>
59
60 <button class="back"><a class="link" href="{% url 'main_page' %}">Back</a></button>
61 <button class="regenerate"><a class="regen" href="{% url 'regenerate_tasks' %}">Regen
62 <button class="add_task"><a class="add" href="{% url 'add_task' %}">Add Row</a></butto
63 <button class="download"><a class="link" href="{% url 'download_excel' %}"><i class="f
64 </button>
65
66
67
68
69
70 </body>
71 </html>
72
Ln 68, Col 1 Spaces: 4 UTF-8 CRLF Django HTML Go Live Prettier
```

4 User Review and Confirmation:

4.1 Edit task:

The page for editing the task created using Django form, model, html, CSS

Template:



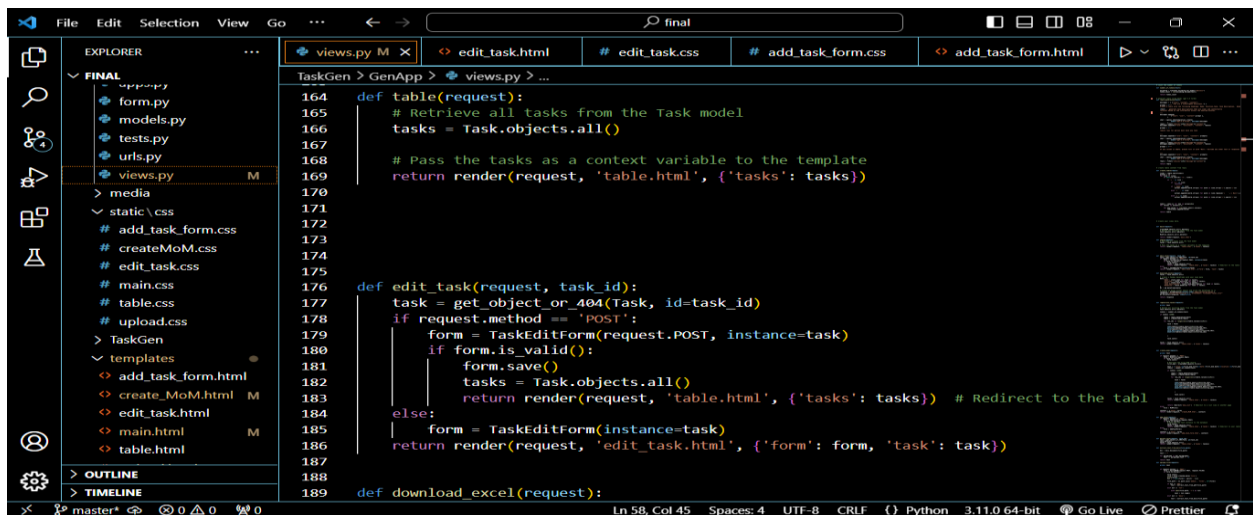
The screenshot shows a code editor with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure with the following files and folders:

- FINAL
 - # createMoM.css
 - # edit_task.css
 - # main.css
 - # table.css
 - # upload.css
 - TaskGen
 - templates
 - add_task_form.html
 - create_MoM.html M
 - edit_task.html
 - main.html M
 - table.html
 - upload.html M
 - db.sqlite3
 - manage.py
 - gitignore.txt
 - README.md
 - report.pdf
 - requirements.txt
 - technical workflow.pdf

The Editor panel shows the content of the `edit_task.html` template:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Edit Task</title>
6     {% load static %}
7     <link rel="stylesheet" type="text/css" href="{% static 'css/edit_task.css' %}">
8 </head>
9 <body>
10     <h1>Edit Task</h1>
11     <form method="post">
12         {{ csrf_token }}
13         {{ form.as_p }}
14         <button class="btn"><a class="link" href="{% url 'table_page' %}">Back</a></button>
15         <button type="submit">Save</button>
16     </form>
17 </body>
18 </html>
```

Views:



The screenshot shows a code editor with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure with the following files and folders:

- FINAL
 - app.py
 - form.py
 - models.py
 - tests.py
 - urls.py
 - views.py M
 - media
 - static/css
 - add_task_form.css
 - createMoM.css
 - edit_task.css
 - main.css
 - table.css
 - upload.css
 - TaskGen
 - templates
 - add_task_form.html
 - create_MoM.html M
 - edit_task.html
 - main.html M
 - table.html

The Editor panel shows the content of the `views.py` file:

```
164 def table(request):
165     # Retrieve all tasks from the Task model
166     tasks = Task.objects.all()
167
168     # Pass the tasks as a context variable to the template
169     return render(request, 'table.html', {'tasks': tasks})
170
171
172
173
174
175
176 def edit_task(request, task_id):
177     task = get_object_or_404(Task, id=task_id)
178     if request.method == 'POST':
179         form = TaskEditForm(request.POST, instance=task)
180         if form.is_valid():
181             form.save()
182             tasks = Task.objects.all()
183             return render(request, 'table.html', {'tasks': tasks}) # Redirect to the tabl
184         else:
185             form = TaskEditForm(instance=task)
186             return render(request, 'edit_task.html', {'form': form, 'task': task})
187
188
189 def download_excel(request):
```

4.2 Add task:

Views:

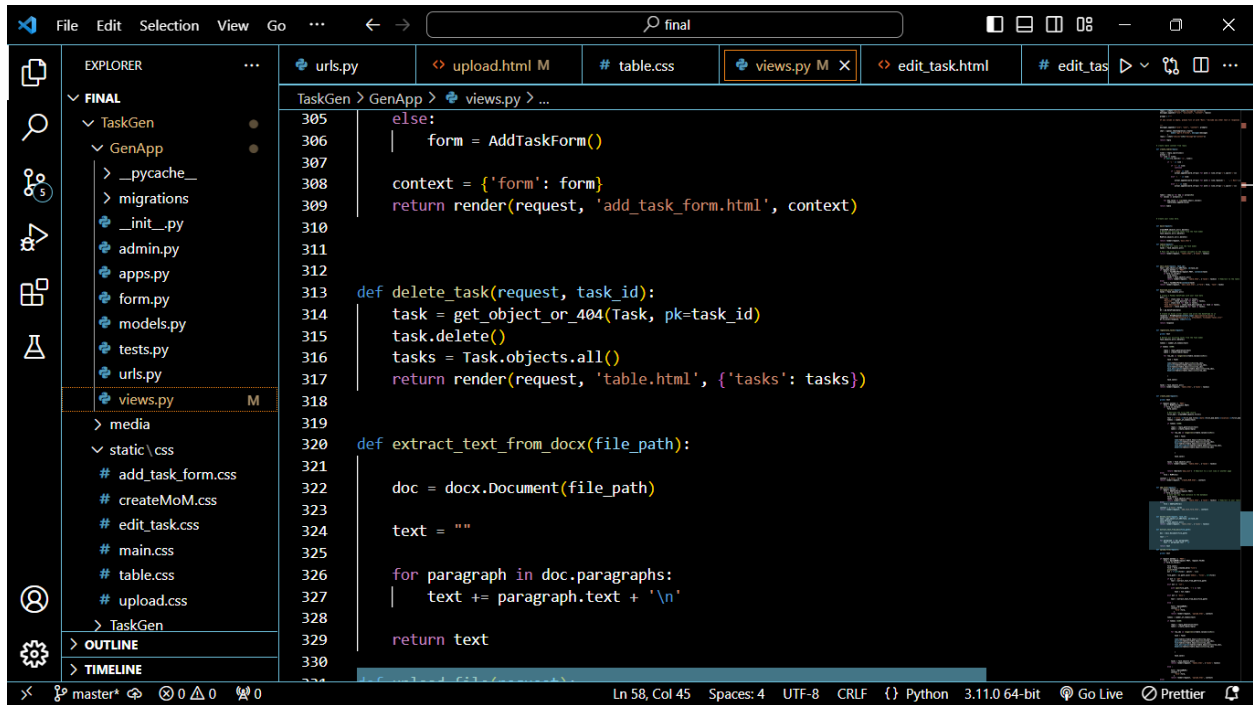
```
TaskGen > GenApp > views.py > ...
295
296
297 def add_task(request):
298     if request.method == 'POST':
299         form = AddTaskForm(request.POST)
300         if form.is_valid():
301             # Save the new Task instance to the database
302             form.save()
303             tasks = Task.objects.all()
304             return render(request, 'table.html', {'tasks': tasks}) # Redirect to your tab
305         else:
306             form = AddTaskForm()
307
308     context = {'form': form}
309     return render(request, 'add_task_form.html', context)
310
311
312
313 def delete_task(request, task_id):
314     task = get_object_or_404(Task, pk=task_id)
315     task.delete()
316     tasks = Task.objects.all()
317     return render(request, 'table.html', {'tasks': tasks})
318
319
320 def extract_text_from_docx(file_path):
```

Template:

```
TaskGen > templates > add_task_form.html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Add Task</title>
6     {% load static %}
7     <link rel="stylesheet" type="text/css" href="{% static 'css/add_task_form.css' %}">
8 </head>
9 <body>
10     <h1>Add Task</h1>
11     <form method="post">
12         {% csrf_token %}
13         {{ form.as_p }}
14         <button class="back"><a class='link' href="{% url 'table_page' %}">Back</a></button>
15         <input type="submit" value="Add Task">
16     </form>
17 </body>
18 </html>
19
```


4.3 Delete task:

We use python fuction for deleting a task from django model

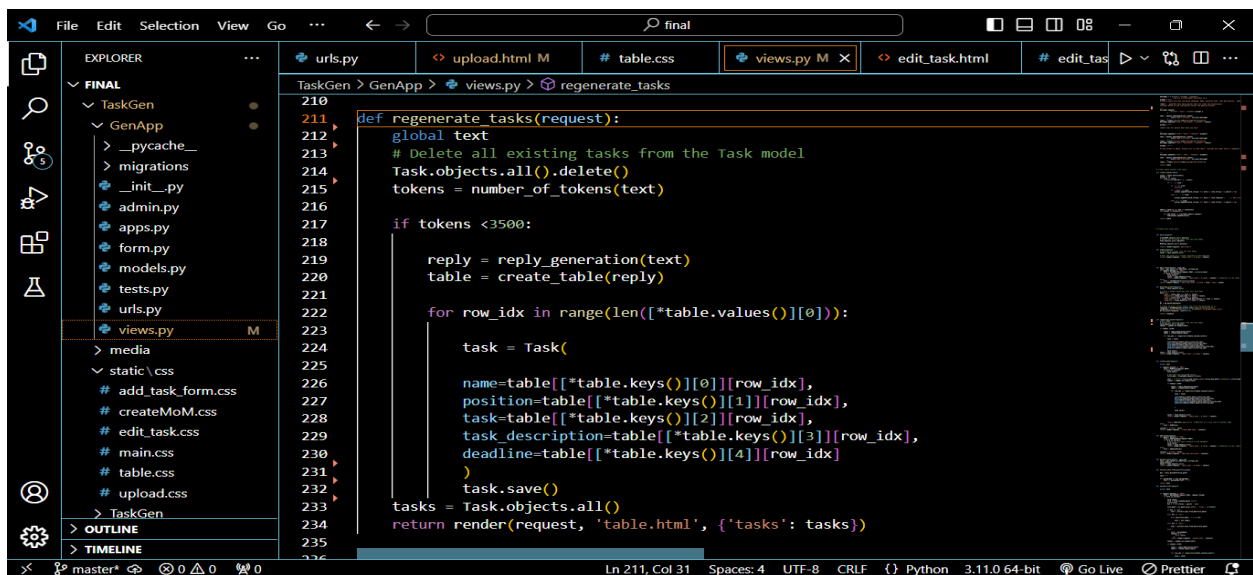


The screenshot shows a VS Code editor window with the Explorer sidebar on the left. The Explorer sidebar shows a project structure with folders like FINAL, TaskGen, and GenApp. The GenApp folder is expanded, showing files like __pycache__, migrations, _init_.py, admin.py, apps.py, form.py, models.py, tests.py, urls.py, and views.py. The views.py file is selected and its content is displayed in the main editor. The code shows a function delete_task that takes a request and task_id as arguments. It uses get_object_or_404 to retrieve the task, deletes it, and then renders the 'table.html' template with the updated list of tasks. The status bar at the bottom indicates the file is at line 58, column 45, with 4 spaces, UTF-8 encoding, CRLF line endings, and is using Python 3.11.0 64-bit.

```
305     else:
306         form = AddTaskForm()
307
308     context = {'form': form}
309     return render(request, 'add_task_form.html', context)
310
311
312
313 def delete_task(request, task_id):
314     task = get_object_or_404(Task, pk=task_id)
315     task.delete()
316     tasks = Task.objects.all()
317     return render(request, 'table.html', {'tasks': tasks})
318
319
320 def extract_text_from_docx(file_path):
321
322     doc = docx.Document(file_path)
323
324     text = ""
325
326     for paragraph in doc.paragraphs:
327         text += paragraph.text + '\n'
328
329     return text
330
331
```

4.4 Regenerate task :

Using URLs mapping we regenerate task

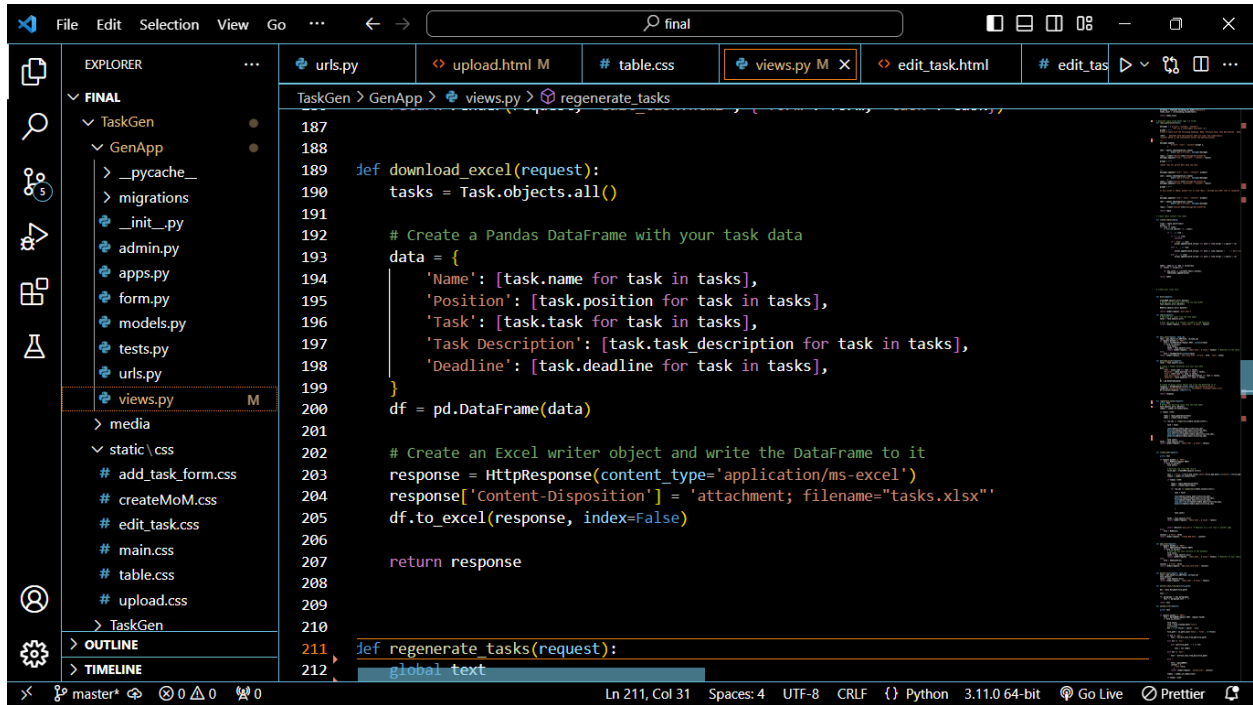


The screenshot shows a VS Code editor window with the Explorer sidebar on the left. The Explorer sidebar shows a project structure with folders like FINAL, TaskGen, and GenApp. The GenApp folder is expanded, showing files like __pycache__, migrations, _init_.py, admin.py, apps.py, form.py, models.py, tests.py, urls.py, and views.py. The views.py file is selected and its content is displayed in the main editor. The code shows a function regenerate_tasks that takes a request as an argument. It uses global text to delete all existing tasks from the Task model, calculates the number of tokens in the text, and then generates a new table based on the text. The status bar at the bottom indicates the file is at line 211, column 31, with 4 spaces, UTF-8 encoding, CRLF line endings, and is using Python 3.11.0 64-bit.

```
210
211 def regenerate_tasks(request):
212     global text
213     # Delete all existing tasks from the Task model
214     Task.objects.all().delete()
215     tokens = number_of_tokens(text)
216
217     if tokens < 3500:
218
219         reply = reply_generation(text)
220         table = create_table(reply)
221
222         for row_idx in range(len(table.values()[0])):
223
224             task = Task(
225
226                 name=table[table.keys()[0]][row_idx],
227                 position=table[table.keys()[1]][row_idx],
228                 task=table[table.keys()[2]][row_idx],
229                 task_description=table[table.keys()[3]][row_idx],
230                 deadline=table[table.keys()[4]][row_idx]
231             )
232             task.save()
233         tasks = Task.objects.all()
234         return render(request, 'table.html', {'tasks': tasks})
235
236
```

5 Task Creation:

By using Django, model and pandas created a download option for getting Excel file output



The screenshot shows a VS Code editor with a Django project structure. The Explorer panel on the left shows the project hierarchy: FINAL > TaskGen > GenApp. The file explorer lists various files including _pyscache_, migrations, _init_.py, admin.py, apps.py, form.py, models.py, tests.py, urls.py, views.py (selected), media, static/css, add_task_form.css, createMoM.css, edit_task.css, main.css, table.css, upload.css, TaskGen, OUTLINE, and TIMELINE. The main editor window displays the code for views.py, showing a function download_excel(request) that uses Pandas to create a DataFrame from task data and an HttpResponse to return an Excel file. The code is as follows:

```
187
188
189 def download_excel(request):
190     tasks = Task.objects.all()
191
192     # Create a Pandas DataFrame with your task data
193     data = {
194         'Name': [task.name for task in tasks],
195         'Position': [task.position for task in tasks],
196         'Task': [task.task for task in tasks],
197         'Task Description': [task.task_description for task in tasks],
198         'Deadline': [task.deadline for task in tasks],
199     }
200     df = pd.DataFrame(data)
201
202     # Create an Excel writer object and write the DataFrame to it
203     response = HttpResponse(content_type='application/ms-excel')
204     response['Content-Disposition'] = 'attachment; filename="tasks.xlsx"'
205     df.to_excel(response, index=False)
206
207     return response
208
209
210
211 def regenerate_tasks(request):
212     global text
```

The status bar at the bottom indicates the current file is views.py, line 211, column 31, with 4 spaces, UTF-8 encoding, CRLF line endings, Python 3.11.0 64-bit, and Prettier formatting.