

CPSC 449 WEB-BACKEND ASSIGNMENT

TO-DO FLASK APPLICATION SCREENSHOTS

This ToDo Application has two pages, Home page (index.html) which shows all the tasks and allows us to either delete a task or update it and Add Task Page(add_task.html) which allows us to add a task.

The following are the screenshots of the application pages.

- **Application Home page (index.html)**

The screenshot shows a web browser window with the URL '127.0.0.1' in the address bar. The title bar says 'Todo Application'. The main content area displays a heading 'WELCOME TO TODO-APPLICAITON!' followed by a 'Task List'. Three tasks are listed:

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 Delete Update
- Title ID: 14 Title: task2 Description: task2desc Date: 2024-03-21 - 0 Delete Update
- Title ID: 15 Title: task3 Description: task3desc Date: 2024-03-22 - 0 Delete Update

At the bottom left, there are links for 'Add Task' and 'View API Documentation'.

- **Add Task Page (add_task.html)**

The screenshot shows a web browser window with the URL 127.0.0.1 in the address bar. The title of the page is "Todo Application". The main content is titled "Add Task". It contains three input fields: "Title" with the value "task3", "Description" with the value "task3desc", and "Due Date" with the value "03/22/2024". Below these fields is a "Submit" button.

Field	Value
Title	task3
Description	task3desc
Due Date	03/22/2024

The following is the Database table schema and entries that have been used in the application (todo_db) and table (tasks)

```
[Tables_in_todo_db]
+-----+
| tasks |
+-----+
|1 row in set (0.01 sec)

mysql> SELECT * FROM tasks;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'SELCT * FROM tasks' at line 1
mysql> DESCRIBE todo_db.tasks;
+-----+
| Field | Type      | Null | Key | Default | Extra           |
+-----+
| id    | int       | NO   | PRI  | NULL    | auto_increment |
| title | varchar(255)| NO   |     | NULL    |                 |
| description | text    | YES  |     | NULL    |                 |
| due_date | date     | YES  |     | NULL    |                 |
| completed | tinyint(1) | YES  |     | 0       |                 |
+-----+
5 rows in set (0.01 sec)

mysql> SHOW COLUMNS FROM todo_db.tasks;
+-----+
| Field | Type      | Null | Key | Default | Extra           |
+-----+
| id    | int       | NO   | PRI  | NULL    | auto_increment |
| title | varchar(255)| NO   |     | NULL    |                 |
| description | text    | YES  |     | NULL    |                 |
| due_date | date     | YES  |     | NULL    |                 |
| completed | tinyint(1) | YES  |     | 0       |                 |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM todo_db.tasks;
+-----+
| id | title      | description | due_date | completed |
+-----+
| 2 | Task updated | updated description | 2024-03-17 | 1          |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM todo_db.tasks;
+-----+
| id | title      | description | due_date | completed |
+-----+
| 2 | Task updated | updated description | 2024-03-17 | 1          |
| 7 | task updated 2 | task updated 2 desc | 2024-03-12 | 0          |
+-----+
2 rows in set (0.00 sec)

mysql> USE todo_db
Database changed
mysql> SELECT * FROM todo_db.tasks;
+-----+
| id | title      | description | due_date | completed |
+-----+
| 13 | task1     | task1desc  | 2021-03-12 | 0          |
| 16 | task3     | task3desc  | 2024-03-17 | 0          |
| 17 | task 6    | task 6 desc | 2024-03-02 | 0          |
+-----+
3 rows in set (0.00 sec)

mysql> ]
```

The following are my app.py where I did all the API endpoint definitions, index.html and add_task.html which are the Frontend pages of the applications.

- **app.py code**

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a project structure for a "TODO_APP". The files listed are app.py, add_task.html, index.html, venv, 400.html, 404.html, and 500.html.
- Code Editor:** The main pane displays Python code for a Flask application. The code includes MySQL configuration, database connection setup, and definitions for TaskForm and Task classes. It also contains a create_task function and a get_tasks function.
- Status Bar:** At the bottom right, it shows "Ln 148, Col 10" and other system information like "Spaces:4", "UTF-8", "LF", and "Python 3.12.2 (venv: venv)".

```
from flask import Flask, render_template, request, redirect, jsonify, url_for
import pymysql
from wtforms import Form, StringField, TextAreaField, DateField, BooleanField, validators
from flask_swagger_ui import get_swaggerui_blueprint

app = Flask(__name__)

# MySQL Configuration
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'Dheeraj@123ms'
app.config['MYSQL_DB'] = 'todo_db'

db = pymysql.connect(host=app.config['MYSQL_HOST'],
                     user=app.config['MYSQL_USER'],
                     password=app.config['MYSQL_PASSWORD'],
                     db=app.config['MYSQL_DB'],
                     charset='utf8mb4',
                     cursorclass=pymysql.cursors.DictCursor)

cursor = db.cursor()

# TaskForm definition
class TaskForm(Form):
    title = StringField('Title', validators=[validators.DataRequired()])
    description = TextAreaField('Description')
    due_date = DateField('Due Date', format='%Y-%m-%d', validators=[validators.DataRequired()])

class Task:
    def __init__(self, id, title, description, due_date, completed):
        self.id = id
        self.title = title
        self.description = description
        self.due_date = due_date
        self.completed = completed

def create_task(title, description, due_date):
    sql = "INSERT INTO tasks (title, description, due_date) VALUES (%s, %s, %s)"
    cursor.execute(sql, (title, description, due_date))
    db.commit()

def get_tasks():
    sql = "SELECT * FROM tasks"
```

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a project structure under "TODO_APP".
- Editor:** Displays the content of "app.py".
- Bottom Status Bar:** Shows "Ln 148, Col 10" and other file-related information.

```
EXPLORER            app.py            add_task.html          index.html
Todo App
templates
  add_task.html
  index.html
venv
  400.html
  404.html
  500.html
app.py

app.py > ...
52
53
54
55     def get_tasks():
56         sql = "SELECT * FROM tasks"
57         cursor.execute(sql)
58         tasks = []
59         for row in cursor.fetchall():
60             task = Task(row['id'], row['title'], row['description'], row['due_date'], row['completed'])
61             tasks.append(task)
62
63     return tasks
64
64     def get_task_by_id_from_database(task_id):
65         sql = "SELECT * FROM tasks WHERE id = %s"
66         cursor.execute(sql, (task_id,))
67         row = cursor.fetchone()
68
69         if row:
70             task = Task(row['id'], row['title'], row['description'], row['due_date'], row['completed'])
71             return task
72         else:
73             return None
74
75
76     def update_task(task_id, title, description, due_date, completed):
77         sql = "UPDATE tasks SET title=%s, description=%s, due_date=%s, completed=%s WHERE id=%s"
78         cursor.execute(sql, (title, description, due_date, completed, task_id))
79         db.commit()
80
81
82
83     # Routes
84
85     #GET
86     @app.route('/', methods=['GET'])
87     def index():
88         tasks = get_tasks()
89         return render_template('index.html', tasks=tasks)
90
91
92     #POST
93     @app.route('/add_task', methods=['GET', 'POST'])
94     def add_task():
95         if request.method == 'POST':
96             print("Received a POST request to add a task")
97             form = TaskForm(request.form)
98             if form.validate():
99                 title = form.title.data
100                description = form.description.data
101                due_date = form.due_date.data
102                create_task(title, description, due_date)
103                return redirect(url_for('index'))
104            else:
105                return render_template('add_task.html', form=form)
106        else:
```

NAME: SRI HARI DHEERAJ KOMMINENI
CWD: 885177196

The screenshot shows a Python development environment with the following details:

- Explorer View:** Shows the project structure under `TODO_APP`, including files like `add_task.html`, `index.html`, `400.html`, `404.html`, and `500.html`.
- Code Editor:** The main editor window displays `app.py` containing the following code:

```
    else:
        return render_template('add_task.html', form=form)
    else:
        print("Rendering add task form for GET request")
        # If it's a GET request, render the add task Form
        form = TaskForm()
        return render_template('add_task.html', form=form)

114     #PUT
115     @app.route('/update_task<int:task_id>', methods=['PUT'])
116     def update_task_route(task_id):
117         form = TaskForm(request.form)
118         if form.validate():
119             title = form.title.data
120             description = form.description.data
121             due_date = form.due_date.data
122             completed = 'completed' in request.form
123             update_task(task_id, title, description, due_date, completed)
124             return jsonify({"message": "Task updated successfully"})
125         else:
126             return jsonify({"error": "Validation failed"}), 400
127
128     #DELETE
129     @app.route('/delete_task<int:task_id>', methods=['DELETE'])
130     def delete_task(task_id):
131         if request.method == 'DELETE':
132             print("Received a DELETE request to delete task with ID:", task_id)
133             try:
134                 # Perform deletion logic here
135                 sql = "DELETE FROM tasks WHERE id = %s"
136                 cursor.execute(sql, (task_id,))
137                 db.commit()
138                 return jsonify({"message": "Task deleted successfully"})
139             except Exception as e:
140                 return jsonify({"error": str(e)}), 500
141         else:
142             # If the request method is not DELETE, return a Method Not Allowed error
143             return jsonify({"error": "Method Not Allowed"}), 405
144
145
146     # Swagger
147     SWAGGER_URL = '/swagger'
148     API_URL = '/swagger.json'
149
150
151     swaggerui_blueprint = get_swaggerui_blueprint(
152         SWAGGER_URL,
153         API_URL,
154         config={
155             'app_name': "Todo Application"
156         }
157     )
158
```

The code implements a RESTful API for managing tasks. It includes endpoints for creating, updating, and deleting tasks, and a Swagger UI endpoint for generating documentation.

The screenshot shows a code editor interface with the following details:

- Explorer View:** Shows the project structure under `TODO_APP`, including `templates` (with `add_task.html` and `index.html`) and `venv`.
- Code Editor:** The main pane displays `app.py` containing the following Python code:

```
161 app.register_blueprint(swaggerui_blueprint, url_prefix=SWAGGER_URL)
162
163 @app.route('/swagger.json')
164 def swagger_spec():
165     spec = {
166         "swagger": "2.0",
167         "info": {
168             "title": "Todo Application API",
169             "description": "API for managing tasks in a todo application",
170             "version": "1.0"
171         },
172         "paths": {
173             "/add_task": {
174                 "post": {
175                     "summary": "Add a new task",
176                     "parameters": [
177                         {
178                             "name": "title",
179                             "in": "formData",
180                             "description": "Title of the task",
181                             "required": True,
182                             "type": "string"
183                         },
184                         {
185                             "name": "description",
186                             "in": "formData",
187                             "description": "Description of the task",
188                             "required": False,
189                             "type": "string"
190                         },
191                         {
192                             "name": "due_date",
193                             "in": "formData",
194                             "description": "Due date of the task (format: YYYY-MM-DD)",
195                             "required": True,
196                             "type": "string",
197                             "format": "date"
198                         }
199                     ],
200                     "responses": {
201                         "200": {
202                             "description": "Task added successfully"
203                         },
204                         "400": {
205                             "description": "Validation failed"
206                         }
207                     }
208                 }
209             },
210             "/update_task/{task_id}": {
211                 "put": {
212                     "summary": "Update an existing task",
213                     "parameters": [
214                         {
215                             "name": "title",
216                             "in": "formData",
217                             "description": "Title of the task",
218                             "required": True,
219                             "type": "string"
220                         },
221                         {
222                             "name": "description",
223                             "in": "formData",
224                             "description": "Description of the task",
225                             "required": False,
226                             "type": "string"
227                         },
228                         {
229                             "name": "due_date",
230                             "in": "formData",
231                             "description": "Due date of the task (format: YYYY-MM-DD)",
232                             "required": True,
233                             "type": "string",
234                             "format": "date"
235                         }
236                     ],
237                     "responses": {
238                         "200": {
239                             "description": "Task updated successfully"
240                         },
241                         "400": {
242                             "description": "Validation failed"
243                         }
244                     }
245                 }
246             }
247         }
248     }
249 
```

- Bottom Status Bar:** Shows the current file is `app.py`, with 147 lines of code, 4 spaces, and 8 tabs.

The screenshot shows the VS Code interface with the file `app.py` open. The code defines a REST API spec for a todo application. It includes endpoints for adding tasks, updating tasks, deleting tasks, and getting all tasks. The `PUT` method for updating a task requires parameters like `task_id`, `title`, `description`, `due_date`, and `completed`. The `DELETE` method for deleting a task requires a `task_id`. The `GET` method for getting all tasks returns a successful response with a list of tasks.

```
PUT: {
    "summary": "Update an existing task",
    "parameters": [
        {
            "name": "task_id",
            "in": "path",
            "description": "ID of the task to update",
            "required": True,
            "type": "integer"
        },
        {
            "name": "title",
            "in": "formData",
            "description": "New title of the task",
            "required": True,
            "type": "string"
        },
        {
            "name": "description",
            "in": "formData",
            "description": "New description of the task",
            "required": False,
            "type": "string"
        },
        {
            "name": "due_date",
            "in": "formData",
            "description": "New due date of the task (format: YYYY-MM-DD)",
            "required": True,
            "type": "string",
            "format": "date"
        },
        {
            "name": "completed",
            "in": "formData",
            "description": "Whether the task is completed",
            "required": False,
            "type": "boolean"
        }
    ],
    "responses": {
        "200": {
            "description": "Task updated successfully"
        },
        "400": {
            "description": "Validation failed"
        }
    }
},
"/delete_task/{task_id}": {
    "delete": {
        "summary": "Delete a task",
        "parameters": [
            {
                "name": "task_id",
                "in": "path",
                "description": "ID of the task to delete",
                "required": True,
                "type": "integer"
            }
        ],
        "responses": {
            "200": {
                "description": "Task deleted successfully"
            },
            "404": {
                "description": "Task not found"
            }
        }
    }
},
"/": {
    "get": {
        "summary": "Get all tasks",
        "responses": {
            "200": {
                "description": "Tasks retrieved successfully"
            }
        }
    }
}
}

return jsonify(spec)
```

Ln 147, Col 1 Spaces: 4 UTF-8 LF (Python 3.12.2 ('venv': venv))

This screenshot shows the same `app.py` file as the previous one, but with additional code at the bottom. The new code handles the `__main__` condition and runs the application with debug mode enabled.

```
if __name__ == '__main__':
    app.run(debug=True)
```

Ln 147, Col 1 Spaces: 4 UTF-8 LF (Python 3.12.2 ('venv': venv))

- **index.html (Home Page)**

The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows a project structure for "TODO_APP" containing "templates", "add_task.html", "index.html", "venv", "400.html", "404.html", "500.html", and "app.py".
- EDITOR:** Displays the content of "index.html".
- STATUS BAR:** Shows "Ln 23, Col 54" and "Spaces: 4 - UTF-8 LF HTML".

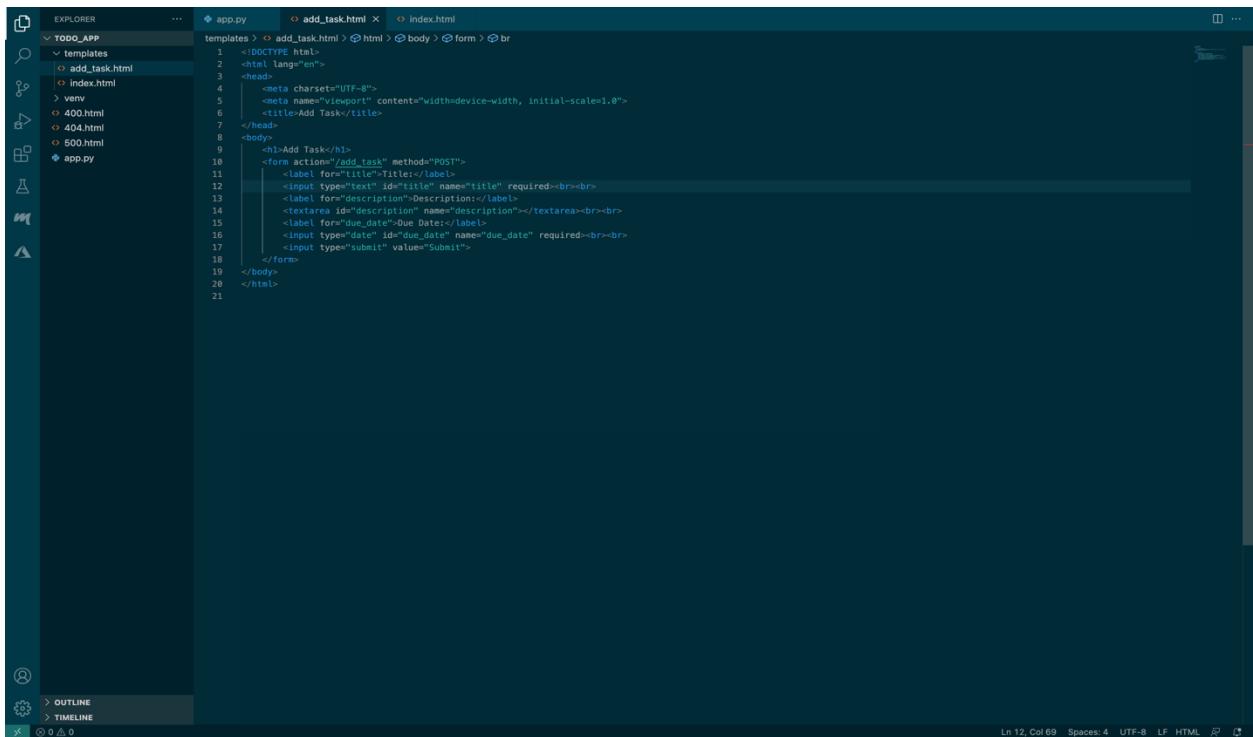
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Task List</title>
7  </head>
8  <body>
9      <h1>WELCOME TO TODO-APPLICATION!</h1>
10     <h2>Task List</h2>
11     <ul>
12         {% for task in tasks %}
13             <li><strong>Title ID: {{task.id}}</strong> <strong>Title: {{ task.title }}</strong> <strong>Description:</strong>{{ task.description }} <strong>Date:</strong>{{ task.due_date|date:'Y-m-d' }}<br>
14             <!-- Button to trigger the delete operation -->
15             <button onclick="deleteTask( {{task.id}} )">Delete</button>
16             <button onclick="updateTask( {{ task.id }} )">Update</button>
17         </li>
18         <br>
19     {% endfor %}
20     </ul>
21     <button><a href="/add_task">Add Task</a></button>
22     <button><a href="/swagger">View API Documentation</a></button>
23
24
25     <!-- Script to send DELETE request -->
26     <script>
27         function deleteTask(taskId) {
28             if (confirm("Are you sure you want to delete this task?")) {
29                 fetch('/delete_task/' + taskId), {
30                     method: 'DELETE'
31                 }
32                 .then(response => {
33                     if (response.ok) {
34                         console.log('Task deleted successfully');
35                         // Reload the page after successful deletion
36                         location.reload();
37                     } else {
38                         console.error('Error deleting task:', response.status);
39                         // Handle error if needed
40                     }
41                 })
42                 .catch(error => console.error('Error:', error));
43             }
44         }
45
46         function updateTask(taskId) {
47             // Prompt user for new task details
48             var newTitle = prompt("Enter new title:");
49             var newDescription = prompt("Enter new description:");
50             var newDueDate = prompt("Enter new due date (YYYY-MM-DD):");
51
52             // Create a new FormData object with updated task data
53             var formData = new FormData();
54             formData.append('title', newTitle);
55             formData.append('description', newDescription);
56             formData.append('due_date', newDueDate);
57
58             // Send PUT request using fetch API
59             fetch('/update_task/' + taskId, {
60                 method: 'PUT',
61                 body: formData
62             })
63             .then(response => {
64                 if (response.ok) {
65                     // If request is successful, reload the page to reflect changes
66                     window.location.reload();
67                 } else {
68                     // If request fails, display error message
69                     alert('Failed to update task. Please try again.');
70                 }
71             })
72             .catch(error => {
73                 console.error('Error:', error);
74                 alert('An unexpected error occurred. Please try again later.');
75             });
76         }
77     </script>
78 </body>
79 </html>
```

The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows a project structure for "TODO_APP" containing "templates", "add_task.html", "index.html", "venv", "400.html", "404.html", "500.html", and "app.py".
- EDITOR:** Displays the content of "index.html".
- STATUS BAR:** Shows "Ln 23, Col 54" and "Spaces: 4 - UTF-8 LF HTML".

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Task List</title>
7  </head>
8  <body>
9      <h1>WELCOME TO TODO-APPLICATION!</h1>
10     <h2>Task List</h2>
11     <ul>
12         {% for task in tasks %}
13             <li><strong>Title ID: {{task.id}}</strong> <strong>Title: {{ task.title }}</strong> <strong>Description:</strong>{{ task.description }} <strong>Date:</strong>{{ task.due_date|date:'Y-m-d' }}<br>
14             <!-- Button to trigger the delete operation -->
15             <button onclick="deleteTask( {{task.id}} )">Delete</button>
16             <button onclick="updateTask( {{ task.id }} )">Update</button>
17         </li>
18         <br>
19     {% endfor %}
20     </ul>
21     <button><a href="/add_task">Add Task</a></button>
22     <button><a href="/swagger">View API Documentation</a></button>
23
24
25     <!-- Script to send DELETE request -->
26     <script>
27         function deleteTask(taskId) {
28             if (confirm("Are you sure you want to delete this task?")) {
29                 fetch('/delete_task/' + taskId), {
30                     method: 'DELETE'
31                 }
32                 .then(response => {
33                     if (response.ok) {
34                         console.log('Task deleted successfully');
35                         // Reload the page after successful deletion
36                         location.reload();
37                     } else {
38                         console.error('Error deleting task:', response.status);
39                         // Handle error if needed
40                     }
41                 })
42                 .catch(error => console.error('Error:', error));
43             }
44         }
45
46         function updateTask(taskId) {
47             // Prompt user for new task details
48             var newTitle = prompt("Enter new title:");
49             var newDescription = prompt("Enter new description:");
50             var newDueDate = prompt("Enter new due date (YYYY-MM-DD):");
51
52             // Create a new FormData object with updated task data
53             var formData = new FormData();
54             formData.append('title', newTitle);
55             formData.append('description', newDescription);
56             formData.append('due_date', newDueDate);
57
58             // Send PUT request using fetch API
59             fetch('/update_task/' + taskId, {
60                 method: 'PUT',
61                 body: formData
62             })
63             .then(response => {
64                 if (response.ok) {
65                     // If request is successful, reload the page to reflect changes
66                     window.location.reload();
67                 } else {
68                     // If request fails, display error message
69                     alert('Failed to update task. Please try again.');
70                 }
71             })
72             .catch(error => {
73                 console.error('Error:', error);
74                 alert('An unexpected error occurred. Please try again later.');
75             });
76         }
77     </script>
78 </body>
79 </html>
```

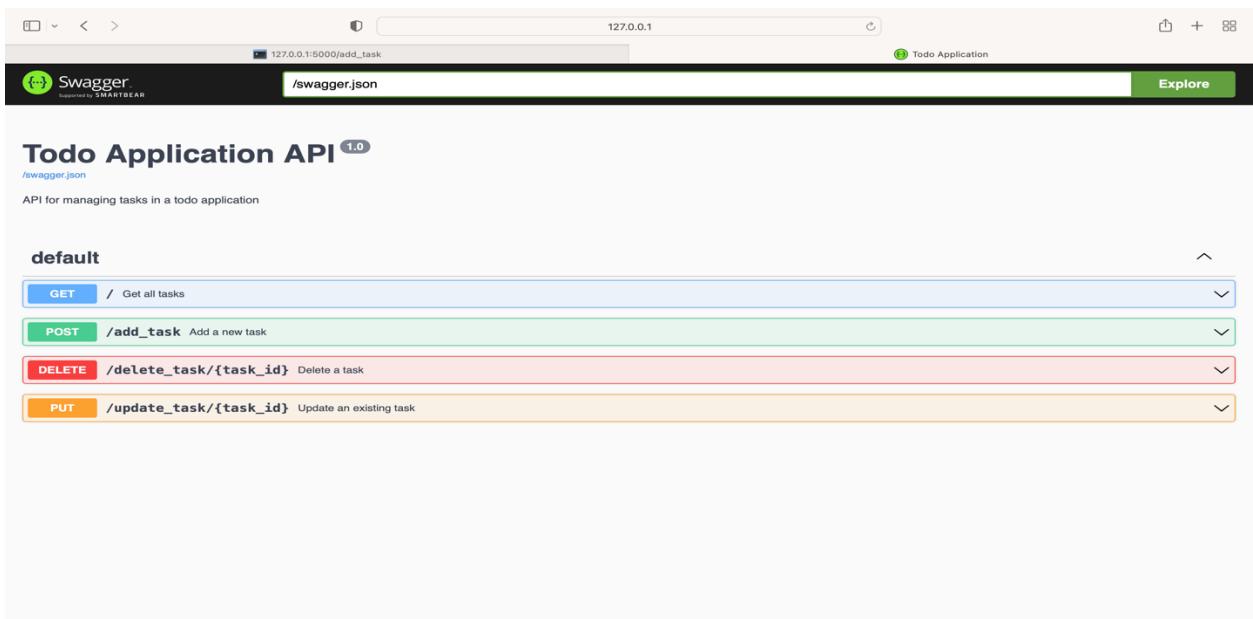
- **add_task.html (Add Task Page)**



```
EXPLORER app.py add_task.html index.html
templates> add_task.html > HTML > body > form > br
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Add Task</title>
7 </head>
8 <body>
9   <h1>Add Task</h1>
10  <form action="/add_task" method="POST">
11    <label for="title">Title:</label>
12    <input type="text" id="title" name="title" required=<br><br>
13    <label for="description">Description:</label>
14    <textarea id="description" name="description"></textarea><br><br>
15    <label for="due_date">Due Date:</label>
16    <input type="date" id="due_date" name="due_date" required=<br><br>
17    <input type="submit" value="Submit">
18  </form>
19 </body>
20 </html>
```

Following is the screenshot of Swagger UI of my application which has all the GET, POST, PUT, DELETE definitions.

- **SWAGGER UI**



The screenshot shows the Swagger UI for a Todo Application API. The URL is 127.0.0.1:5000/add_task. The interface includes:

- Todo Application API 1.0**
- /swagger.json**
- Explore** button
- default** section
 - GET /tasks** - Get all tasks
 - POST /add_task** - Add a new task
 - DELETE /delete_task/{task_id}** - Delete a task
 - PUT /update_task/{task_id}** - Update an existing task

Demonstrating each API in the Swagger

- GET in Swagger

The screenshot shows the Swagger UI interface for a 'Todo Application'. The URL in the browser is 127.0.0.1:5000/add_task. The 'default' operation is selected, which is a GET request to '/'. The 'Parameters' section indicates 'No parameters'. Below it are 'Responses' and 'Code Details' sections. The 'Code Details' section for status code 200 shows the response body as an HTML document. The HTML content includes a welcome message 'WELCOME TO TODO-APPLICATION!', a title 'Task List', and a list of tasks. The tasks listed are:

- task4 desc: task4 desc Date: 2024-03-14 - 0
- task AJ desc: task AJ desc Date: 2012-03-1
- task MA desc: task MA desc Date: 2022-06-1

This screenshot shows the same Swagger UI interface, but the response body is now displayed as a list of tasks. The tasks are:

- task4 desc: task4 desc Date: 2024-03-14 - 0
- task AJ desc: task AJ desc Date: 2012-03-1
- task MA desc: task MA desc Date: 2022-06-1

Below the response body, there is a 'Response headers' section showing the following details:

- connection: close
- content-length: 4189
- content-type: text/html; charset=utf-8
- date: Wed, 28 Mar 2024 12:13:33 GMT
- server: Werkzeug/3.0.1 Python/3.12.2

The 'Responses' section at the bottom indicates a successful response with code 200 and the message 'Tasks retrieved successfully'.

- **POST in Swagger**

The screenshot shows the Swagger UI interface for a 'Todo Application'. The URL is 127.0.0.1. The main title is 'Task List' and the sub-section is 'Todo Application'. The endpoint being viewed is 'POST /add_task' with the description 'Add a new task'. The 'Parameters' section contains three fields: 'title' (string, required, value: task1), 'description' (string, value: task1desc), and 'due_date' (string, required, value: 2021-03-12). Below the parameters is a large blue 'Execute' button. At the bottom, there's a 'Responses' section with a 'Code' dropdown set to 200 and a 'Description' field. The 'Response content type' is set to 'application/json'.

The screenshot shows the response for the POST /add_task endpoint. The status code is 200 and the response body is a JSON array of tasks:

```
[{"id": 9, "title": "task4", "description": "task4 desc", "date": "2024-03-14"}, {"id": 11, "title": "task AJ", "description": "task AJ desc", "date": "2012-03-11"}, {"id": 12, "title": "task MA", "description": "task MA desc", "date": "2022-06-12"}]
```

Below the response body, there are sections for 'Response headers' and 'Responses'. The 'Response headers' section shows the following details:

```
connection: close
content-length: 4109
content-type: text/html; charset=utf-8
date: Wed, 28 Mar 2024 12:13:33 GMT
server: Werkzeug/3.0.1 Python/3.12.2
```

The 'Responses' section has a 'Code' dropdown set to 200 and a 'Description' field.



WELCOME TO TODO-APPLICAITON!

Task List

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 [Delete](#) [Update](#)

[Add Task](#) [View API Documentation](#)

- PUT in Swagger

The screenshot shows the Swagger UI interface for a Todo Application. The URL in the address bar is 127.0.0.1. The main content area displays the `/update_task/{task_id}` endpoint, which is described as "Update an existing task".
Parameters:

- task_id** * required (path) - ID of the task to update. Value: 15
- title** * required (formData) - New title of the task. Value: task2
- description** (formData) - New description of the task. Value: task2desc
- due_date** * required (formData) - New due date of the task (format: YYYY-MM-DD). Value: 2022-11-02
- completed** (formData) - Whether the task is completed. Value: true

At the bottom right of the form, there is a blue "Execute" button.

The screenshot shows a Postman interface with a 'Task List' collection selected. A specific request titled 'true' is being viewed. The 'Responses' tab is active, showing a successful 200 response. The response body contains the message: '{ "message": "Task updated successfully" }'. The response headers include: connection: close, content-length: 45, content-type: application/json, date: Wed, 20 Mar 2024 12:20:28 GMT, server: Werkzeug/3.0.1 Python/3.12.2.

The screenshot shows a Postman interface with a 'My first collection' collection selected. A specific request titled 'http://127.0.0.1:5000' is being viewed. The 'Body' tab is active, showing form-data parameters: title (task5), description (task5desc), due_date (2024-03-19), and completed (true). The response status is 200 OK, and the response body is: 'WELCOME TO TODO-APPLICAITON!'. Below the response, a 'Task List' section displays three tasks:

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 [Delete] [Update]
- Title ID: 15 Title: task5 Description: task5desc Date: 2024-03-19 - 1 [Delete] [Update]
- Title ID: 16 Title: task3 Description: task3desc Date: 2024-03-17 - 0 [Delete] [Update]

- **DELETE in Swagger**

The screenshot shows the Swagger UI for the Todo Application API. The URL is 127.0.0.1. The page title is "Todo Application API". The main section is titled "default". It lists three operations: GET / (Get all tasks), POST /add_task (Add a new task), and DELETE /delete_task/{task_id} (Delete a task). The DELETE operation is highlighted with a red border. The "Parameters" section shows a required parameter "task_id" of type integer (path) with value "14". Below it is an "Execute" button. The "Responses" section shows a 200 status code with the description "Task deleted successfully".

The screenshot shows the results of executing the DELETE /delete_task/{task_id} operation. The "Curl" section contains the command: curl -X 'DELETE' \ 'http://127.0.0.1:5000/delete_task/14' \ -H 'accept: application/json'. The "Request URL" is http://127.0.0.1:5000/delete_task/14. The "Server response" section shows a 200 status code. The "Response body" is a JSON object: { "message": "Task deleted successfully" }. The "Response headers" include: connection: close, content-length: 45, content-type: application/json, date: Wed, 28 Mar 2024 12:19:08 GMT, server: Werkzeug/3.0.1 Python/3.12.2. The "Responses" section shows a 200 status code with the description "Task deleted successfully".

WELCOME TO TODO-APPLICAITON!

Task List

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 [Delete](#) [Update](#)
- Title ID: 15 Title: task3 Description: task3desc Date: 2024-03-22 - 0 [Delete](#) [Update](#)

[Add Task](#) [View API Documentation](#)

The following are the screenshots of each API request and response in POSTMAN.

- **GET in POSTMAN**

Learn from experts and join the Postman community at POST/CON 24. Register by March 26 to save 30%.

My Workspace

My first collection

GET http://127.0.0.1:5000

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers (8 hidden)

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 12 ms Size: 3.41 KB Save as example

WELCOME TO TODO-APPLICAITON!

Task List

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 [Delete](#) [Update](#)
- Title ID: 15 Title: task2 Description: task2desc Date: 2022-11-02 - 1 [Delete](#) [Update](#)

Add Task View API Documentation

- **POST in POSTMAN**

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar is visible with options like 'Collections', 'Environments', and 'History'. A 'Create a collection for your requests' section is present. In the main workspace, a POST request is being prepared to the URL `http://127.0.0.1:5000/add_task`. The 'Body' tab is selected, showing a form-data structure with four fields: 'title' (Value: task3), 'description' (Value: task3desc), 'due_date' (Value: 2024-03-17), and 'completed' (Value: true). Below the table, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Test Results' tab shows a successful response with status 200 OK, time 5 ms, and size 3.79 KB. The response body contains the message 'WELCOME TO TODO-APPLICAITON!' and a 'Task List' section with three items. At the bottom, there are links for 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Trash', and other navigation icons.

• PUT in POSTMAN

PUT http://127.0.0.1:5000/update_task/15

Key	Value	Description
<input checked="" type="checkbox"/> title	Text task5	
<input checked="" type="checkbox"/> description	Text task5desc	
<input checked="" type="checkbox"/> due_date	Text 2024-03-19	
<input checked="" type="checkbox"/> completed	Text true	
Key	Text Value	Description

Status: 200 OK Time: 8 ms Size: 210 B Save as example

```
{ "message": "Task updated successfully" }
```

GET http://127.0.0.1:5000

Key	Value	Description
<input checked="" type="checkbox"/> title	Text task5	
<input checked="" type="checkbox"/> description	Text task5desc	
<input checked="" type="checkbox"/> due_date	Text 2024-03-19	
<input checked="" type="checkbox"/> completed	Text true	
Key	Text Value	Description

Status: 200 OK Time: 6 ms Size: 3.79 KB Save as example

WELCOME TO TODO-APPLICAITON!

Task List

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 [Delete] [Update]
- Title ID: 15 Title: task5 Description: task5desc Date: 2024-03-19 - 1 [Delete] [Update]
- Title ID: 16 Title: task3 Description: task3desc Date: 2024-03-17 - 0 [Delete] [Update]

- **DELETE in POSTMAN**

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays a collection named 'My first collection' with two nested folders: 'First folder inside collection' and 'Second folder inside collection'. Each folder contains several requests. A modal window is open in the center, showing a 'DELETE' request to the URL `http://127.0.0.1:5000/delete_task/15`. The 'Headers' tab is selected, showing 8 hidden headers. The 'Body' tab shows the response: `{ "message": "Task deleted successfully" }`. The status bar at the bottom indicates a 200 OK status, 7 ms time, and 210 B size.

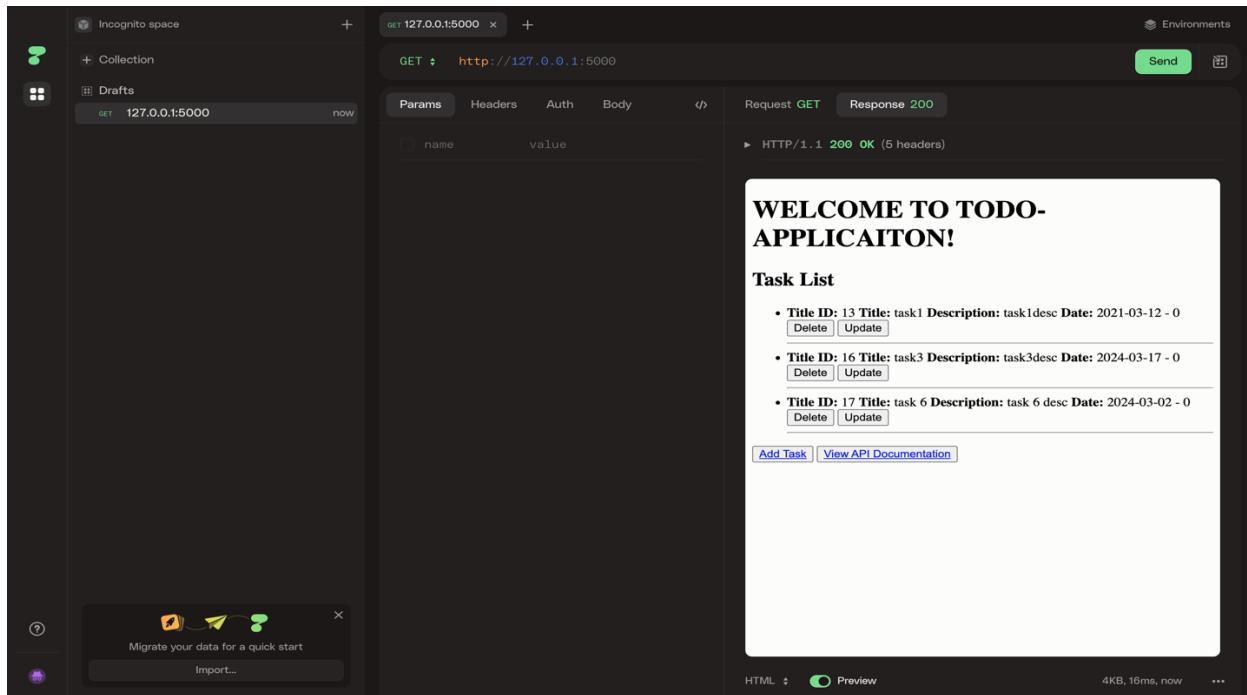
The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays a collection named 'My first collection' with two nested folders: 'First folder inside collection' and 'Second folder inside collection'. Each folder contains several requests. A modal window is open in the center, showing a 'GET' request to the URL `http://127.0.0.1:5000`. The 'Headers' tab is selected, showing 8 hidden headers. The 'Body' tab shows the response: `WELCOME TO TODO-APPLICATION!` followed by a 'Task List' section. The 'Task List' section contains two items:

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 [Delete] [Update]
- Title ID: 16 Title: task3 Description: task3desc Date: 2024-03-17 - 0 [Delete] [Update]

Buttons for 'Add Task' and 'View API Documentation' are also present. The status bar at the bottom indicates a 200 OK status, 6 ms time, and 3.41 KB size.

The following are the screenshots of each API request and response in HTTPPie.

- **GET in HTTPPie**



HTTPPie Application Screenshot (GET Request):

Request URL: `GET 127.0.0.1:5000`

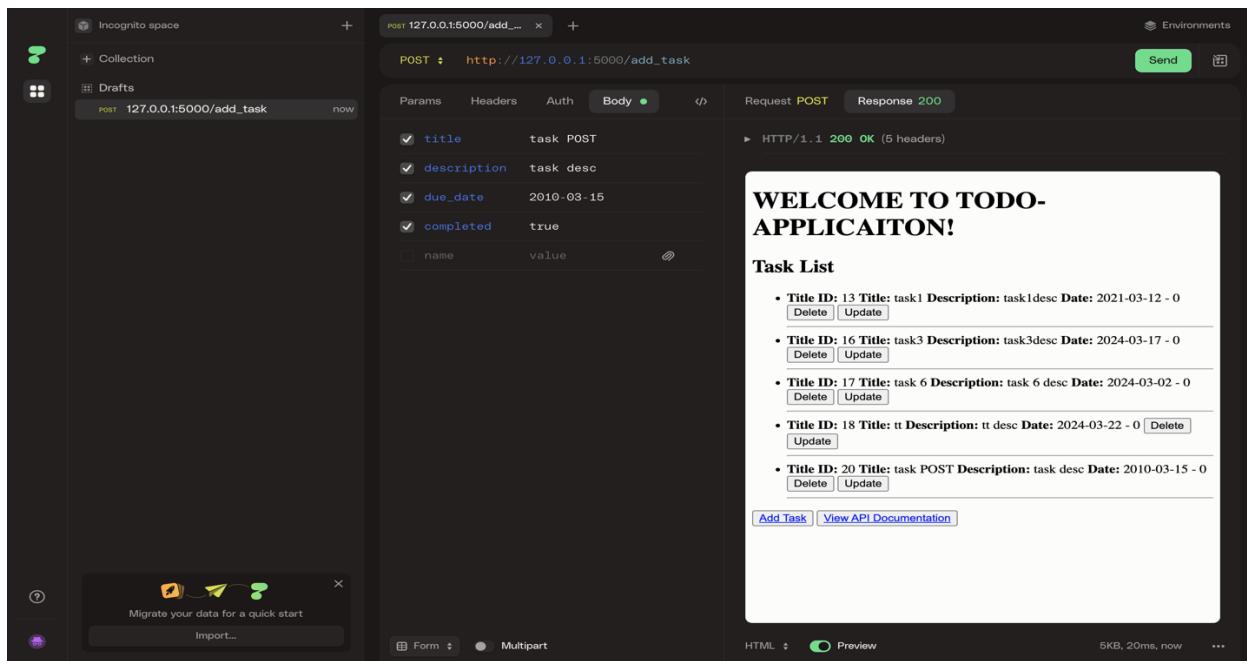
Response Status: `HTTP/1.1 200 OK (5 headers)`

Task List:

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0
- Title ID: 16 Title: task3 Description: task3desc Date: 2024-03-17 - 0
- Title ID: 17 Title: task 6 Description: task 6 desc Date: 2024-03-02 - 0

Buttons: Add Task, View API Documentation

- **POST in POSTMAN**



POSTMAN Application Screenshot (POST Request):

Request URL: `POST 127.0.0.1:5000/add_task`

Body Params:

- title: task POST
- description: task desc
- due_date: 2010-03-15
- completed: true

Response Status: `HTTP/1.1 200 OK (5 headers)`

Task List:

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0
- Title ID: 16 Title: task3 Description: task3desc Date: 2024-03-17 - 0
- Title ID: 17 Title: task 6 Description: task 6 desc Date: 2024-03-02 - 0
- Title ID: 18 Title: tt Description: tt desc Date: 2024-03-22 - 0
- Title ID: 20 Title: task POST Description: task desc Date: 2010-03-15 - 0

Buttons: Add Task, View API Documentation

- **PUT in HTTPPie**

The screenshot shows the HTTPPie application interface. On the left, there's a sidebar with 'Incognito space', '+ Collection', and 'Drafts'. A draft titled '127.0.0.1:5000/update_task/20' is selected. The main area has tabs for 'PUT', 'http://127.0.0.1:5000/update_task/20', 'Params', 'Headers', 'Auth', 'Body', and 'Response'. The 'Body' tab is active, showing a JSON payload with four fields: 'title' (task PUT), 'description' (task updated), 'due_date' (2010-03-15), and 'completed' (true). The 'Response' tab shows a 200 OK status with a message: "Task updated successfully". At the bottom, there's a progress bar indicating 210B, 32ms, now.

The screenshot shows the HTTPPie application interface. On the left, there's a sidebar with 'Incognito space', '+ Collection', and 'Drafts'. A draft titled '127.0.0.1:5000' is selected. The main area has tabs for 'GET', 'http://127.0.0.1:5000', 'Params', 'Headers', 'Auth', 'Body', and 'Response'. The 'Response' tab shows a 200 OK status with a message: "WELCOME TO TODO-APPPLICAITON!". Below this, there's a 'Task List' section with a list of tasks. Each task item includes a title, description, date, and delete/update buttons. The tasks listed are:

- Title ID: 13 Title: task1 Description: task1desc Date: 2021-03-12 - 0 [Delete] [Update]
- Title ID: 16 Title: task3 Description: task3desc Date: 2024-03-17 - 0 [Delete] [Update]
- Title ID: 17 Title: task 6 Description: task 6 desc Date: 2024-03-02 - 0 [Delete] [Update]
- Title ID: 18 Title: tt Description: tt desc Date: 2024-03-22 - 0 [Delete] [Update]
- Title ID: 20 Title: task PUT Description: task updated Date: 2010-03-15 - 1 [Delete] [Update]

At the bottom, there are buttons for 'Add Task' and 'View API Documentation'. The bottom right corner shows 'HTML', 'Preview', '5KB, 6ms, now', and three dots.

- **DELETE in HTTPPie**

Incognito space

DELETE : http://127.0.0.1:5000/delete_task/20

Params Headers Auth Body Request DELETE Response 200

✓ title task PUT
✓ description task updated
✓ due_date 2010-03-15
✓ completed true

► HTTP/1.1 200 OK (5 headers)

1 "message": "Task deleted successfully"
2
3

Migrate your data for a quick start Import... Form Multipart JSON 210B, 13ms, now ...

Incognito space

GET 127.0.0.1:5000

Params Headers Auth Body Request GET Response 200

name value

► HTTP/1.1 200 OK (5 headers)

WELCOME TO TODO-APPLICAITON!

Task List

- **Title ID: 13** **Title: task1** **Description: task1desc** **Date: 2021-03-12 - 0**
Delete Update
- **Title ID: 16** **Title: task3** **Description: task3desc** **Date: 2024-03-17 - 0**
Delete Update
- **Title ID: 17** **Title: task 6** **Description: task 6 desc** **Date: 2024-03-02 - 0**
Delete Update
- **Title ID: 18** **Title: tt** **Description: tt desc** **Date: 2024-03-22 - 0**
Delete Update

Add Task View API Documentation

HTML Preview 4KB, 5ms, 4m ago ...