Full Stack Python Application

1. Project Overview

Build a complete full stack **Task Management Application** using Python.

The project includes:

- Python backend
- FastAPI REST API
- SQLite database
- HTML frontend
- CSS styling
- JavaScript API integration
- Create, update and delete tasks
- Task completion tracking

FastAPI can serve HTML templates and static CSS files, while JavaScript can communicate with the backend API.

2. Features

- Add tasks
- View tasks
- Mark tasks as completed
- Delete tasks
- SQLite persistence
- REST API
- HTML/CSS/JavaScript frontend
- Responsive interface
- Automatic API documentation
- No frontend framework required

3. Technologies

- Python 3
- FastAPI
- Uvicorn
- SQLite
- Jinja2
- HTML
- CSS
- JavaScript

SQLite provides a lightweight database without requiring a separate database server. Parameterized queries are used for database operations.

4. Folder Structure

```
full_stack_python_app/
□□□ main.py
□□□ database.py
□□□ schemas.py
□□□ requirements.txt
□□□ README.md
□□□ templates/
□   □□□ index.html
□□□ static/
    □□□ style.css
    □□□ app.js
```

5. How the Application Works

```
Browser
  □
HTML + CSS + JavaScript
```

```
    def connect(self):
        return sqlite3.connect(self.database_name)

    def create_table(self):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS tasks (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                completed INTEGER NOT NULL DEFAULT 0
            )
        """)

        connection.commit()
        connection.close()

    def create_task(self, title):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute(
            "INSERT INTO tasks (title) VALUES (?)",
            (title,)
        )

        connection.commit()
        task_id = cursor.lastrowid
        connection.close()

        return task_id
```

```
    def get_tasks(self):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, title, completed
            FROM tasks
            ORDER BY id DESC
        """)

        tasks = [dict(row) for row in cursor.fetchall()]
        connection.close()

        return tasks

    def update_task(self, task_id, completed):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            UPDATE tasks
            SET completed = ?
            WHERE id = ?
        """, (completed, task_id))

        connection.commit()
        updated = cursor.rowcount > 0
        connection.close()

        return updated

    def delete_task(self, task_id):
        connection = self.connect()
```

```
    cursor = connection.cursor()

    cursor.execute(
        "DELETE FROM tasks WHERE id = ?",
        (task_id,)
    )

    connection.commit()
    deleted = cursor.rowcount > 0
    connection.close()

    return deleted
```

schemas.py

```
from pydantic import BaseModel, Field


class TaskCreate(BaseModel):
    title: str = Field(
        min_length=1,
        max_length=200
    )


class TaskUpdate(BaseModel):
    completed: bool


class Task(TaskCreate):
    id: int
    completed: bool
```

FastAPI uses Pydantic models to read and validate request bodies automatically.

main.py

```
from pathlib import Path

from fastapi import FastAPI, HTTPException, Request
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates

from database import TaskDatabase
from schemas import Task, TaskCreate, TaskUpdate


BASE_DIR = Path(__file__).resolve().parent


app = FastAPI(
    title="Full Stack Task Manager",
    description="A full stack Python task management application.",
    version="1.0.0"
)


app.mount(
    "/static",
    StaticFiles(directory=BASE_DIR / "static"),
    name="static"
)

templates = Jinja2Templates(
    directory=BASE_DIR / "templates"
)

database = TaskDatabase()
```

```
@app.get("/", response_class=HTMLResponse)
def home(request: Request):
    return templates.TemplateResponse(
        request=request,
        name="index.html"
    )


@app.post(
    "/api/tasks",
    response_model=Task,
    status_code=201
)
def create_task(task: TaskCreate):
    task_id = database.create_task(task.title)

    return {
        "id": task_id,
        "title": task.title,
        "completed": False
    }


@app.get(
    "/api/tasks",
    response_model=list[Task]
)
def get_tasks():
    tasks = database.get_tasks()

    return [
        {
```

```
            "id": task["id"],
            "title": task["title"],
            "completed": bool(task["completed"])
        }
        for task in tasks
    ]


@app.patch(
    "/api/tasks/{task_id}",
    response_model=Task
)
def update_task(
    task_id: int,
    task: TaskUpdate
):
    updated = database.update_task(
        task_id,
        int(task.completed)
    )

    if not updated:
        raise HTTPException(
            status_code=404,
            detail="Task not found."
        )

    tasks = database.get_tasks()

    for item in tasks:
        if item["id"] == task_id:
            return {
                "id": item["id"],
                "title": item["title"],
```

```
            "completed": bool(item["completed"])
        }


@app.delete("/api/tasks/{task_id}")
def delete_task(task_id: int):
    deleted = database.delete_task(task_id)

    if not deleted:
        raise HTTPException(
            status_code=404,
            detail="Task not found."
        )

    return {
        "message": "Task deleted successfully."
    }
```

7. Complete Frontend Code

templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta
        name="viewport"
        content="width=device-width, initial-scale=1.0"
    >

    <title>Task Manager</title>

    <link
        rel="stylesheet"
```

```
      href="{{ url_for('static', path='/style.css') }}"
    >
</head>

<body>

  <main class="container">

    <section class="app">

      <h1>Task Manager</h1>

      <p class="subtitle">
        Manage your tasks with Python.
      </p>

      <form id="task-form">

        <input
          id="task-input"
          type="text"
          placeholder="Enter a task..."
          maxlength="200"
          required
        >

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

      </form>

      <p id="message"></p>
```

```
        <ul id="task-list"></ul>

      </section>

    </main>

    <script
      src="{{ url_for('static', path='/app.js') }}"
    ></script>

</body>
</html>
```

static/style.css

```
* {
    box-sizing: border-box;
}

body {
    margin: 0;
    font-family: Arial, sans-serif;
    background: #0f172a;
    color: #f8fafc;
}

.container {
    width: 100%;
    max-width: 700px;
    margin: 0 auto;
    padding: 60px 20px;
}

.app {
    background: #1e293b;
```

```
    padding: 30px;
    border-radius: 16px;
}

h1 {
    margin-bottom: 8px;
}

.subtitle {
    color: #94a3b8;
    margin-bottom: 25px;
}

form {
    display: flex;
    gap: 10px;
}

input {
    flex: 1;
    padding: 12px;
    border: 1px solid #475569;
    border-radius: 8px;
    background: #0f172a;
    color: white;
}

button {
    border: none;
    border-radius: 8px;
    padding: 12px 18px;
    cursor: pointer;
    background: #38bdf8;
    color: #082f49;
```

```
    font-weight: bold;
}

ul {
    list-style: none;
    padding: 0;
    margin-top: 25px;
}

.task {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 15px;
    padding: 14px;
    margin-bottom: 10px;
    background: #0f172a;
    border-radius: 8px;
}

.task-left {
    display: flex;
    align-items: center;
    gap: 10px;
}

.completed {
    text-decoration: line-through;
    color: #64748b;
}

.delete-button {
    background: #ef4444;
    color: white;
```

```
}

#message {
    min-height: 20px;
    color: #94a3b8;
}

@media (max-width: 600px) {
    form {
        flex-direction: column;
    }

    .task {
        align-items: flex-start;
    }
}
```

static/app.js

```
const form = document.getElementById("task-form");
const input = document.getElementById("task-input");
const taskList = document.getElementById("task-list");
const message = document.getElementById("message");


async function loadTasks() {
    const response = await fetch("/api/tasks");

    if (!response.ok) {
        message.textContent = "Could not load tasks.";
        return;
    }

    const tasks = await response.json();
```

```
    renderTasks(tasks);
}


function renderTasks(tasks) {
    taskList.innerHTML = "";

    tasks.forEach(task => {

        const item = document.createElement("li");
        item.className = "task";

        const left = document.createElement("div");
        left.className = "task-left";

        const checkbox = document.createElement("input");
        checkbox.type = "checkbox";
        checkbox.checked = task.completed;

        checkbox.addEventListener(
            "change",
            () => updateTask(task.id, checkbox.checked)
        );

        const title = document.createElement("span");
        title.textContent = task.title;

        if (task.completed) {
            title.classList.add("completed");
        }

        left.appendChild(checkbox);
        left.appendChild(title);
```

```
        const deleteButton = document.createElement("button");
        deleteButton.textContent = "Delete";
        deleteButton.className = "delete-button";

        deleteButton.addEventListener(
            "click",
            () => deleteTask(task.id)
        );

        item.appendChild(left);
        item.appendChild(deleteButton);

        taskList.appendChild(item);
    });
}


form.addEventListener("submit", async event => {
    event.preventDefault();

    const title = input.value.trim();

    if (!title) {
        return;
    }

    const response = await fetch("/api/tasks", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            title: title
        })
```

```
    });

    if (response.ok) {
        input.value = "";
        message.textContent = "Task added.";
        loadTasks();
    } else {
        message.textContent = "Could not add task.";
    }
});


async function updateTask(id, completed) {
    const response = await fetch(`/api/tasks/${id}`, {
        method: "PATCH",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            completed: completed
        })
    });

    if (response.ok) {
        loadTasks();
    }
}


async function deleteTask(id) {
    const response = await fetch(`/api/tasks/${id}`, {
        method: "DELETE"
    });
```

```
    if (response.ok) {
        message.textContent = "Task deleted.";
        loadTasks();
    }
}


loadTasks();
```

8. How Frontend and Backend Connect

The frontend communicates with FastAPI using the browser's `fetch()` API.

Create

```
JavaScript
    
POST /api/tasks
    
FastAPI
    
SQLite
```

Read

```
JavaScript
    
GET /api/tasks
    
FastAPI
    
SQLite
    
JSON
    
```

JavaScript

HTML

Update

Checkbox

PATCH /api/tasks/{id}

FastAPI

SQLite

Delete

Delete Button

DELETE /api/tasks/{id}

FastAPI

SQLite

9. requirements.txt

```
fastapi
uvicorn
jinja2
```

10. How to Run

Create the project folders:

```
mkdir full_stack_python_app
```

```
cd full_stack_python_app
```

Create a virtual environment:

```
python -m venv venv
```

Activate it on Windows:

```
venv\Scripts\activate
```

Install dependencies:

```
pip install -r requirements.txt
```

Start the server:

```
uvicorn main:app --reload
```

Open the application:

```
http://127.0.0.1:8000
```

Open the API documentation:

```
http://127.0.0.1:8000/docs
```

FastAPI provides interactive API documentation through its generated OpenAPI documentation.

11. Basic Testing

Test the application by:

```
1. Open the website
```

2. Add a task
3. Add several tasks
4. Mark a task as completed
5. Refresh the browser
6. Confirm the data remains
7. Delete a task
8. Open /docs
9. Test the API endpoints directly

The SQLite database is stored locally as:

```
tasks.db
```

12. Small Improvements

Possible next improvements:

- User authentication
- User specific tasks
- Task categories
- Due dates
- Priority levels
- Search
- Filtering
- Dark/light mode
- Edit task titles
- PostgreSQL
- Docker
- Deployment
- React frontend
- JWT authentication

Visit haas.dev for more resources and guides.