Todo App Project Documentation

Introduction

This documentation provides detailed instructions for setting up and running the Flask Todo App project locally on your machine. The project is a web application built using Flask, a popular Python web framework. It includes information on the project structure, dependencies, configuration, and how to run the app.

Prerequisites

Before you begin, ensure that you have the following installed on your local machine:

1. Python 3.x (https://www.python.org/downloads/)

2. pip package manager (usually comes with Python)

3. Virtual environment tool (optional but recommended, e.g., virtualenv)

Setup Instructions

Follow these steps to set up the Flask project on your local machine:

1. Clone the repository:

```
git clone <repository-url>
```

2. (Optional) Create a virtual environment (recommended):

```
# Install virtualenv if you haven't already
pip install virtualenv
# Create a virtual environment
virtualenv ven
# Activate the virtual environment
venv\Scripts\activate
```

3. Install project dependencies:

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

Running the Todo App

Now that you have set up the project and installed the dependencies, follow these steps to run the Todo app:

1. In the root directory of the project, execute the following command to set the Todo app environment variable:

```
# On Windows:
python main.py
```

2. Open your web browser and visit http://127.0.0.1:5000/ to access the app.

Project Structure

The project structure may vary depending on how it was organized. However, a typical Flask project structure may look like:

- `main.py`: The main Flask application file.
- `requirements.txt`: The file containing project dependencies.
- `__init__.py`: The configuration file for the Flask app.
- `api.py`: The file contains API methods.
- `auth.py`: Contains authentication routes.
- `models.py`: The file contains database models.
- `serializer.py`: File for converting objects into understandable data types by JavaScript and frontend framework.
- `views.py`: The views file contains navigation routes.
- `static/`: Directory for static files like CSS and JavaScript.
- `templates/`: Directory for HTML templates.

Database Schema

Table Name	Columns and Constraints
task	Columns "task_id" INTEGER NOT NULL, "task_name" VARCHAR(150), "task_desc" TEXT, "task_status" BOOLEAN NOT NULL, "user_fk" INTEGER, "todo_fk" INTEGER, Constraints FOREIGN KEY - ("user_fk") REFERENCES "user"("user_id") FOREIGN KEY - ("todo_fk") REFERENCES "todo"("todo_id"), PRIMARY KEY - ("task_id")
todo	Columns "todo_id" INTEGER NOT NULL, "todo_name" VARCHAR(150), "todo_date" DATETIME NOT NULL, "user_fk" INTEGER, Constraints UNIQUE - ("todo_name"), FOREIGN KEY - ("user_fk") REFERENCES "user"("user_id"), PRIMARY KEY - ("todo_id")
user	Columns "user_id" INTEGER NOT NULL, "user_email" VARCHAR(150), "user_password" VARCHAR(150), "user_name" VARCHAR(150), Constraints UNIQUE - ("user_email"), UNIQUE - ("user_password"), UNIQUE - ("user_name"), PRIMARY KEY - ("user_id")

API Endpoints

APIs are implemented for Tasks, it includes:

1. "/task"

    TaskList

        GET- displays all tasks

        POST- to create a new task

2. “/task/<int:id>”

    TaskId

        GET – to display task of particular id

        PUT - updates task with id given if not present display error message

        DELETE – deletes task with id given if not present display error message