Flask REST API with Versioning and SQLAlchemy

Project Overview

This project implements a simple **REST API** using **Flask** and **SQLAlchemy**, which manages user data with **versioning** support (`v1` and `v2`) for future extensibility. The API exposes four main HTTP methods to interact with the user data: **GET**, **POST**, **PUT**, and **DELETE**. SQLAlchemy is used to interact with a SQLite database to store user information.

Key Features

- **Versioned API**: Supports multiple versions of the API (e.g., `/api/v1` and `/api/v2`).
- **CRUD Operations**: Create, Read, Update, and Delete user data.
- **Database Integration**: Uses SQLAlchemy to manage user data in a SQLite database.
- **Flask Blueprints**: Modularize routes by splitting them into versioned blueprints.

Project Structure

```
/project
├── app.py                      # Main application file to initialize
Flask and register Blueprints
├── models.py                   # Database models defined using
SQLAlchemy
├── config.py                   # Configuration file for the Flask app
├── controllers                 # Contains versioned API controllers
│   ├── __init__.py
│   ├── v1
│   │   ├── __init__.py
│   │   └── user_controller.py  # v1 user API routes
│   └── v2
│       ├── __init__.py
│       └── user_controller.py  # v2 user API routes
├── migrations                  # Database migrations folder (for future
use with Flask-Migrate)
└── templates                   # Optional folder for templates (if you
use Flask for rendering HTML)
```

Setup and Installation

1. Clone the Repository

```
git clone https://github.com/your-username/flask-api-with-versioning.git
cd flask-api-with-versioning
```

2. Create and Activate a Virtual Environment (Optional but recommended)

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Install Dependencies

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

4. Configuration

- Open `config.py` and update the configuration for the database connection.
    - Default is `sqlite:///users.db`, which creates a SQLite database in the project directory.

5. Initialize Database

Before running the app, you can initialize the SQLite database:

This will create the `users.db` SQLite database automatically if it doesn't exist yet.

API Endpoints

1. User Routes (v1)

- **GET /api/v1/users**
  Retrieve all users from the database.
  - **Response**: List of users (ID, Name, Email, Age).
  - **Example**:

```
[
  {
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com",
    "age": 30
  }
]
```

- **POST /api/v1/users**
  Create a new user.
  - **Request Body**:

```
json
CopyEdit
{
  "name": "Jane Doe",
  "email": "jane@example.com",
  "age": 25
}
```

  - **Response**:

```
json
CopyEdit
{
  "message": "User added"
}
```

2. User Routes (v2)

- **GET /api/v2/users**
  Retrieve all users from the database (v2).
  - **Response**: Same as v1, but may be extended in future versions.

- **POST /api/v2/users**
  Create a new user (v2). The message might differ from v1.
  - **Request Body**:

    ```
    {
      "name": "Sam Smith",
      "email": "sam@example.com",
      "age": 28
    }
    ```

  - **Response**:

    ```
    {
      "message": "User added successfully with v2"
    }
    ```

Database Schema

The `User` model is defined in `models.py` and includes the following fields:

- `id`: Integer (Primary Key)
- `name`: String (User's name)
- `email`: String (User's email address)
- `age`: Integer (User's age)

Example SQLAlchemy Model:

```
# models.py
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    age = db.Column(db.Integer, nullable=False)

    def __repr__(self):
        return f'<User {self.name}>'
```

Running the Application

1. **Run the Flask Application**:
   To start the Flask application, run the following command:

   ```
   python app.py
   ```

   By default, the app runs on `http://127.0.0.1:5000`.

2. **Testing the API**:
   You can use **Postman** or **CURL** to test the API endpoints:
   - **GET /api/v1/users** to fetch all users from version 1.
   - **POST /api/v1/users** to add a new user in version 1.
   - **GET /api/v2/users** to fetch all users from version 2.
   - **POST /api/v2/users** to add a new user in version 2.

Directory Overview

`app.py`

The main application file that configures Flask, registers blueprints, and initializes the database.

`models.py`

Contains the database model for `User`, where SQLAlchemy ORM is used for database interaction.

`controllers`

Contains the route definitions for each API version:

- **`v1/user_controller.py`**: Routes for the version 1 of the user-related API.
- **`v2/user_controller.py`**: Routes for version 2 of the user-related API.

`config.py`

Contains the configuration settings for the Flask app, including the database URI.
