**Flask Setup**

**Install Virtual Environment**

**python3 -m pip install --upgrade pip**

pip install virtualenv

python3 – m venv fenv

source fenv/bin/activate

**Install Flask on Windows or Linux**

pip install flask

Files Directory:

flask\_project/

├── app.py # Main entry point

├── apps/

│ ├── \_\_init\_\_.py # Makes `apps` a package

│ ├── app1/

│ │ ├── \_\_init\_\_.py # Blueprint definition for app1

│ │ ├── routes.py # Routes for app1

│ │ ├── models.py # Models for app1 (if any)

│ │ ├── services.py # Business logic for app1

│ ├── app2/

│ ├── \_\_init\_\_.py # Blueprint definition for app2

│ ├── routes.py # Routes for app2

│ ├── models.py # Models for app2 (if any)

│ ├── services.py # Business logic for app2

├── config.py # Configuration file

├── requirements.txt # Dependencies

└── run.py # Script to run the application

Main Entry Point (app.py)

from flask import Flask

from apps.app1 import app1\_blueprint

def create\_app():

app = Flask(\_\_name\_\_)

# Register blueprints for modular apps

app.register\_blueprint(app1\_blueprint, url\_prefix='/app1')

# Application configurations can go here

app.config.from\_object('config.Config')

return app

To implement multiple apps in a Flask project and design APIs for them, Flask provides flexibility through **blueprints**. Blueprints allow you to organize your application into smaller, modular components. Here's a guide to creating a Flask project with multiple apps and implementing APIs:

**Project Structure**

plaintext

CopyEdit

flask\_project/

├── app.py # Main entry point

├── apps/

│ ├── \_\_init\_\_.py # Makes `apps` a package

│ ├── app1/

│ │ ├── \_\_init\_\_.py # Blueprint definition for app1

│ │ ├── routes.py # Routes for app1

│ │ ├── models.py # Models for app1 (if any)

│ │ ├── services.py # Business logic for app1

│ ├── app2/

│ ├── \_\_init\_\_.py # Blueprint definition for app2

│ ├── routes.py # Routes for app2

│ ├── models.py # Models for app2 (if any)

│ ├── services.py # Business logic for app2

├── config.py # Configuration file

├── requirements.txt # Dependencies

└── run.py # Script to run the application

**Step 1: Main Entry Point (app.py)**

from flask import Flask

from apps.app1 import app1\_blueprint

from apps.app2 import app2\_blueprint

def create\_app():

app = Flask(\_\_name\_\_)

# Register blueprints for modular apps

app.register\_blueprint(app1\_blueprint, url\_prefix='/app1')

app.register\_blueprint(app2\_blueprint, url\_prefix='/app2')

# Application configurations can go here

app.config.from\_object('config.Config')

return app

**Step 2: App Blueprints**

**App 1 (apps/app1/\_\_init\_\_.py):**

from flask import Blueprint # Create a blueprint for app1

app1\_blueprint = Blueprint('app1', \_\_name\_\_) # Import routes to register them

from . import routes

**Step 3: Define API Routes**

**App 1 Routes (apps/app1/routes.py):**

**from flask import jsonify**

**from . import app1\_blueprint**

**@app1\_blueprint.route('/hello', methods=['GET'])**

**def hello\_app1():**

**return jsonify({"message": "Hello from App 1!"})**

**@app1\_blueprint.route('/data', methods=['GET'])**

**def app1\_data():**

**data = {"key": "value from App 1"}**

**return jsonify(data)**

**Step 4: Configuration (config.py)**

class Config:

DEBUG = True

SECRET\_KEY = 'your\_secret\_key'

Step 5: Run the Application (run.py)

from app import create\_app

app = create\_app()

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 6: Install Dependencies**

Create a requirements.txt file:

Flask==2.3.2

pip install -r requirements.txt

Running the Project

python run.py

http://127.0.0.1:5000/app1/hello

Flask Data-Base

**To connect a Flask application to a database, you can use Flask-SQLAlchemy, which provides a convenient ORM (Object Relational Mapping) for database interaction. Below is a step-by-step guide:**

**Install Dependencies**

1. **Install Flask and Flask-SQLAlchemy:**

pip install Flask Flask-SQLAlchemy

**If you are using a specific database (e.g., PostgreSQL, MySQL), install the respective driver:**

* PostgreSQL: pip install psycopg2-binary
* MySQL: pip install pymysql
* SQLite: (No additional installation is required; it's built-in with Python)

**Configuration (config.py)**

**class Config:**

**SQLALCHEMY\_DATABASE\_URI = 'sqlite:///test.db' # SQLite**

**# Example for other databases:**

**# SQLALCHEMY\_DATABASE\_URI ='postgresql+psycopg2://user:password@localhost/dbname'**

**# SQLALCHEMY\_DATABASE\_URI = 'mysql+pymysql://user:password@localhost/dbname'**

**SQLALCHEMY\_TRACK\_MODIFICATIONS = False**

**# Disable tracking for performance**

**Define Models (models.py)**

**from flask\_sqlalchemy import SQLAlchemy**

**# Initialize SQLAlchemy**

**db = SQLAlchemy()**

**# Define a model (table) for the database**

**class User(db.Model):**

**\_\_tablename\_\_ = 'users'**

**id = db.Column(db.Integer, primary\_key=True)**

**name = db.Column(db.String(80), nullable=False)**

**email = db.Column(db.String(120), unique=True, nullable=False)**

**def \_\_repr\_\_(self):**

**return f"<User {self.name}>"**

**Initialize the Migrations Directory**

**flask db init**

**Create a Migration Script**

**flask db migrate -m "Initial migration"**

**Apply the Migration**

**flask db upgrade**

**Rolls back the last migration.**

**flask db downgrade**

**Displays the current migration state.**

**flask db show**

**Shows a list of all migrations.**

**flask db history**

sqlite3 test.db

.tables -- Lists all tables SELECT \* FROM products; -- Check the `products` table