



Declaring Models in Flask

Last Updated : 13 Dec, 2023

Models are used in Flask to conveniently handle interactions with databases (like SQL, [SQLite](#), etc.) using ORM (Object Relational Mapping). This article describes what is ORM, how to declare models in [Flask](#), and finally a simple example Flask application. It assumes basic familiarity with Flask and [Python](#) programming languages.

What is ORM in Python Flask?

ORM (Object Relational Mapping) is a programming technique which lets the programmer to write code using the Object-Oriented features of a language to interact with a database.

This abstracts away the details of the database language and makes the job easier for the programmer. Models are used in Flask to achieve Object Relational Mapping and thus help the programmer to interact with the database using Python's object-oriented features such as classes and methods.

Declaring Models in Flask

The Flask community provides the "[Flask-SQLAlchemy](#)" library/extension which is the go-to library for declaring models in Flask. It is a wrapper around the "SQLAlchemy" library with added capabilities to handle the details related to responses and requests so that you don't have to worry about that. Before proceeding any further, we need to have the following installations -

Installations required

Flask doesn't support ORM, but with the help of flask-sqlalchemy, we can achieve the ORM functionalities. Once Python is installed, we can use a

package manager such as pip to install the rest with this command:

```
pip install flask-sqlalchemy
```

Declaring Models in Flask with Flask-SQLAlchemy

This section contains the step-by-step explanation of the code. After doing the necessary imports, we define the `create_app` function which returns the flask app (the app-factory method of creating flask apps). We create the app and define everything inside this function since this app is small. If it is large, then in-practice, one would generally create multi-file definitions and then add them to the app inside this function by using flask features such as blueprints. Here is the explanation of the code inside this function -

Step 1: Creating flask app

First we create the flask app with the line

```
app = Flask(__name__, ...)
```

as usual and do the configurations with `app.config...`

Python3



```
1 app = Flask(__name__, instance_relative_config=True)
2 app.config.from_pyfile('config.py', silent=True)
3 app.config.from_mapping(SECRET_KEY='dev')
```

Step 2: Creating the instance folder

Instance folder is where our SQL Lite database will be saved. The path to the instance folder is accessed through `app.instance_path` (Note: the instance path is defined in configurations and can be altered from there if one wants). Then we try to create the instance folder with `os.makedirs` method. If it already exists, `OSError` will be raised. We catch the error and do nothing in this case because the folder already exists if error occurs.

Python3

```
3 except OSError:
```

```
pass
```

Step 3: Configure the database location

Next we need to specify flask where to find the database in configurations. For SQL Lite, we need to tell the path of the database file relative to the path to instance folder preceded by "sqlite://" string. This is done with the line "app.config["SQLALCHEMY_DATABASE_URI"] = ..." LINE. Remember this file doesn't need to exist yet.

Python3



```
1 app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///event_database.db"
```

Step 4: Creating the database object (db)

Define the Base class inheriting "DeclarativeBase" (this can be altered if you want some different model other than declarative base by modifying this class"). Next we create the database object using SQLAlchemy and passing this Base class to it and store it in "db". Next, we initiate db with the app using "db.init_app(app)" line.

Python3



```
1 class Base(DeclarativeBase):
2     pass
3
4 db = SQLAlchemy(model_class=Base)
5 db.init_app(app)
```

Step 5: Declaring the model(s)

Now we begin to define the models. A model defines what data elements/rows of the database contain. A model must inherit the db.Model class where db is our database object as defined in last step. Each of the attribute is defined as a class variable in this class (like date and event here) and is assigned the type of data it contains. We define the date to be the primary key. This is SQL Alchemy syntax and one needs to refer to its documentation to get more details.

Our model here is Event which contains the attributes date and event (containing event description).

Python3



```
1 class Event(db.Model):
2     date = mapped_column(db.String, primary_key=True)
3     event = mapped_column(db.String)
```

Step 6: Creating the database

Now we need to define the database to store the data for our model(s). This is done by calling the "db.create_all()" method. This creates the necessary tables and the database files to represent all the models defined by inheriting "db.Model" class. Since we don't need to call this function always, we create a command named 'init-db' using the **click** library and register it with the flask app using "app.cli.add_command" function and passing it the function. Once registered, we can call this command hence call the "db.create_all()" function only when we want using the command just like the **flask run** -

```
flask --app eventLog init-db
```

Once we call this command, our database file and database is ready to handle data for our model(s).

Python3

```

1  @click.command('init-db')
2  def init_db_command():
3      with app.app_context():
4          db.create_all()
5          click.echo('Database created successfully')
6
7  app.cli.add_command(init_db_command)

```

Step 7: Use the SQL Alchemy syntax to query the database

With everything ready here, we can query the database using SQL Alchemy syntax from **db.session** object. For example, here we define a home page querying the database at two places depending upon the request type. When it receives a **GET** request, it returns an HTML page containing table of all the events. For this we query the database to receive all the events using -

```
db.session.execute(db.select(Event).order_by(Event.date)).scalars()
```

and pass it to the rendering template. This query returns an iterable of Event class (or Event model) objects. So if e is an element in this list of iterables, we can access date using e.date and event description using e.event which are the attributes of our model. This is done in the template.

Python3

```

1  @app.route('/', methods=['GET', 'POST'])
2  def home():
3      if(request.method == 'POST'):
4          db.session.add(Event(date=datetime.datetime.now(
5              ).__str__(), event=request.form['eventBox']))
6          db.session.commit()
7          return redirect(url_for('home'))
8          return render_template('home.html', eventsList=db.session.execute(db.select(Event
9
10     return app

```

Similarly, homepage can receive a **POST** request to add an event. In this case we receive the event details from the form and execute a query to add the event with the current date and time and the provided description to the database. For this, we use "db.session.add" and pass it an instance of our Event model/class containing the information. Finally, call "db.session.commit()" to commit the changes i.e., the addition here.

Flask app using Models

We create a simple flask app named "eventLog" where you can see and add events. The date and time are added automatically when the event is added. It just contains a single home page.

File structure

```

rootFolder
|_ eventLog
    |_ templates
        |_ home.html
    |_ __init__.py

```

__init__.py File

This is a Flask web application with a SQLite database for managing events. It defines a "create_app" function setting up the Flask app, SQLAlchemy for database handling, and a simple "Event" model. The code includes a route for rendering and handling form submissions on the home page to add events to the database. Additionally, it has a CLI command "init-db" to initialize the database.

Python3



```

1  from flask import Flask, redirect, url_for, render_template, request
2  import os
3  import datetime
4
5  import click
6
7  from flask_sqlalchemy import SQLAlchemy
8  from sqlalchemy.exc import IntegrityError
9  from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
10
11 def create_app(test_config=None):
12     # a simple page that says hello
13     app = Flask(__name__, instance_relative_config=True)
14
15     app.config.from_pyfile('config.py', silent=True)
16     app.config.from_mapping(SECRET_KEY='dev')
17
18     # ensure instance folder exists
19     try:
20         os.makedirs(app.instance_path)
21     except OSError:
22         pass
23
24     # configure the path to SQLite database, relative to the app instance folder
25     app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///event_database.db"
26
27     class Base(DeclarativeBase):
28         pass
29
30     # create the database object and initiate it
31     db = SQLAlchemy(model_class=Base)
32     db.init_app(app)
33
34     # defining model for event
35     class Event(db.Model):
36         date = mapped_column(db.String, primary_key=True)
37         event = mapped_column(db.String)
38
39     @click.command('init-db')
40     def init_db_command():
41         ''' command for initiating the database '''
42         with app.app_context():
43             db.create_all()
44             click.echo('Database created successfully')
45
46     app.cli.add_command(init_db_command)
47
48     @app.route('/', methods=['GET', 'POST'])

```

```

def home():
50     if(request.method == 'POST'):
51         db.session.add(Event(date=datetime.datetime.now(
52             ).__str__(), event=request.form['eventBox']))
53         db.session.commit()
54         return redirect(url_for('home'))
55     return render_template('home.html', eventsList=db.session.execute(db.select(Event
56
57     return app

```

home.html Jinja Template

This HTML code defines a simple webpage displaying an event log with a table showing date-time and event details. It uses Jinja2 templating to iterate through a list of events passed from the Flask app and dynamically populate the table rows. The page also includes a form to add new events, with a text input and a submit button. Additionally, it links to an external stylesheet named "style.css" for styling.

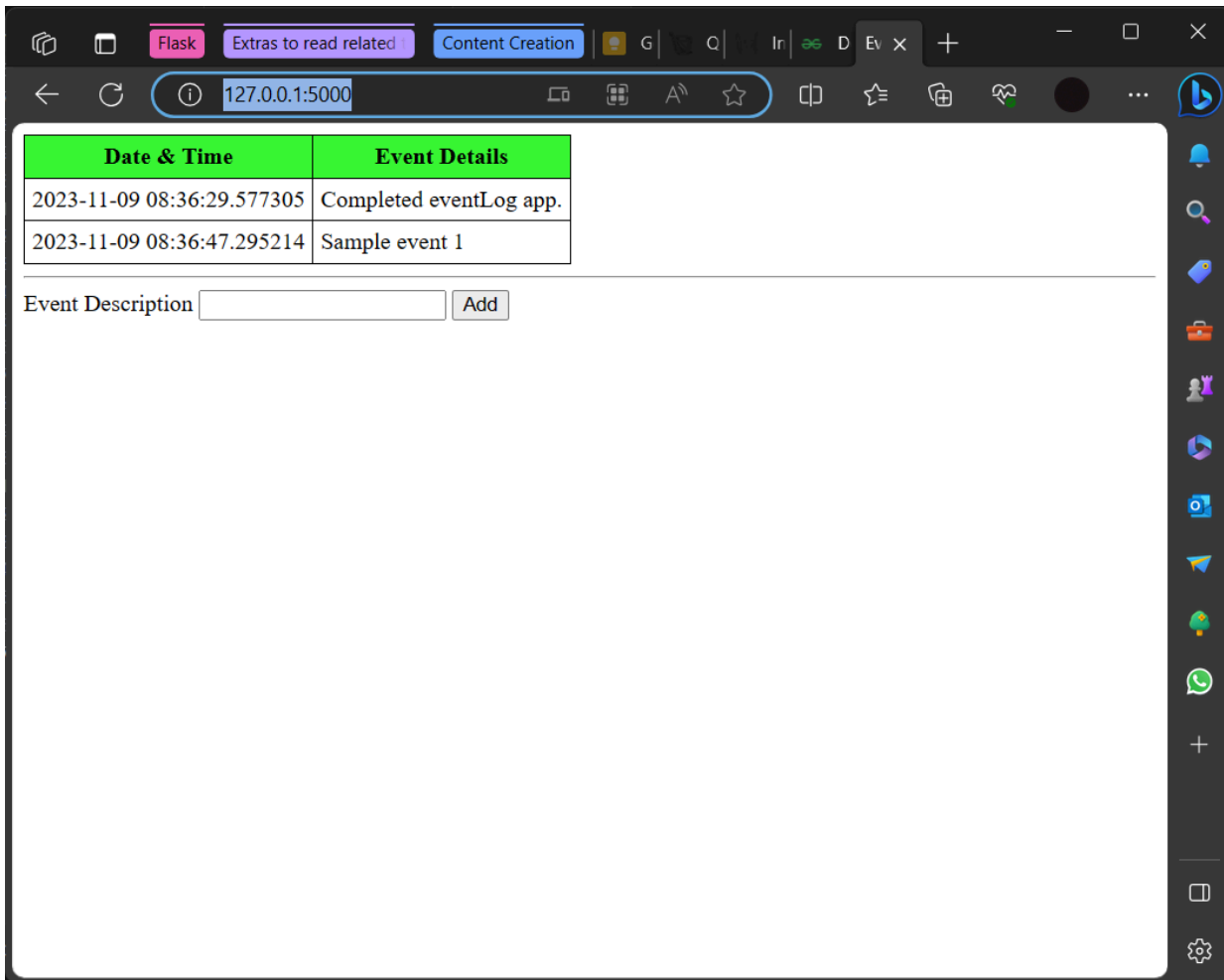
HTML

```

1  <html>
2      <head>
3          <title>EventLog</title>
4          <link rel = 'stylesheet' href = {{url_for('static', filename='style.css')}}/>
5      </head>
6      <body>
7          <table>
8              <tr>
9                  <th>Date & Time</th>
10                 <th>Event Details</th>
11             </tr>
12             {%-for row in eventsList-%}
13                 <tr>
14                     <td>{{row.date}}</td>
15                     <td>{{row.event}}</td>
16                 </tr>
17             {%-endfor-%}
18             </table>
19             <hr/>
20             <form method="post">
21                 <title>Add event</title>
22                 <label for="eventBox">Event Description</label>
23                 <input name="eventBox" id="eventBox" required/>
24                 <input type="submit" value = "Add">
25             </form>
26         </body>
27     </html>

```

Output



A simple flask app "eventLog" running on browser. Uses Flask Models for managing database.

Running the app

First run the following command from the rootFolder to initiate the database -

```
flask --app eventLog init-db
```

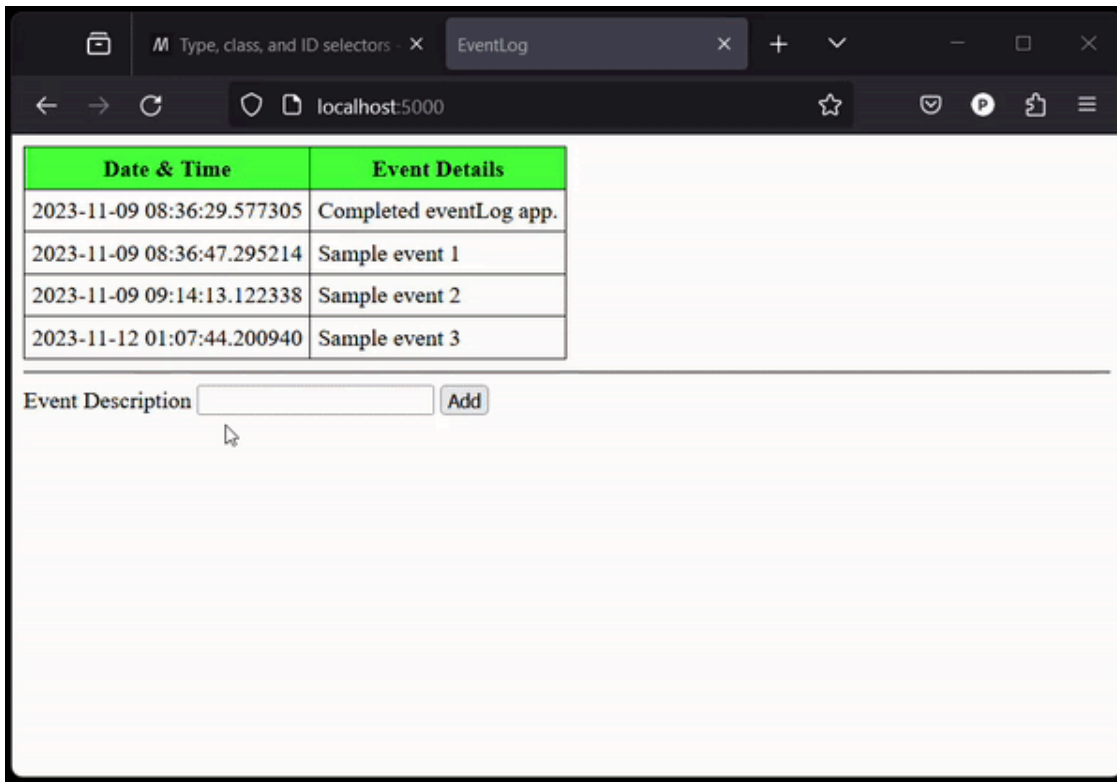
Once done, run the flask app using the command -

```
flask --app eventLog run --debug
```

This will start the app in debug mode at local host port 5000. Visit the following URL in browser -

```
http://127.0.0.1:5000/
```

Video Output



Date & Time	Event Details
2023-11-09 08:36:29.577305	Completed eventLog app.
2023-11-09 08:36:47.295214	Sample event 1
2023-11-09 09:14:13.122338	Sample event 2
2023-11-12 01:07:44.200940	Sample event 3

Event Description

Conclusion

In this article, we saw what object-relational mapping (ORM) is and then what are models in flask and how to declare them using Flask-SQLAlchemy. Declaring models is a way to implement ORM style of working with database which abstracts away the details of the underlying database system thus allowing the programmer to code easily using object-oriented features and use different databases without having to do any significant changes in the code. Finally, we presented an example flask app named "eventLog" which demonstrates declaring models with flask.



grass... + Follow



1

[Next Article](#)

[FloatField - Django Models](#)

Similar Reads

Documenting Flask Endpoint using Flask-Autodoc

Documentation of endpoints is an essential task in web development and being able to apply it in different frameworks is always a utility. This article discusses how endpoints ...

4 min read

How to use Flask-Session in Python Flask ?

Flask Session - Flask-Session is an extension for Flask that supports Server-side Session to your application. The Session is the time between the client logs in to the server and...

4 min read

How to Integrate Flask-Admin and Flask-Login

In order to merge the admin and login pages, we can utilize a short form or any other login method that only requires the username and password. This is known as...

8 min read

Minify HTML in Flask using Flask-Minify

Flask offers HTML rendering as output, it is usually desired that the output HTML should be concise and it serves the purpose as well. In this article, we would display minificatio...

12 min read

Flask URL Helper Function - Flask url_for()

In this article, we are going to learn about the flask url_for() function of the flask URL helper in Python. Flask is a straightforward, speedy, scalable library, used for building,...

11 min read

SQLAlchemy ORM - Declaring Mapping

In this article, we will see how to declare mapping using SQLAlchemy in Python. You will need a database (MySQL, PostgreSQL, SQLite, etc) to work with. Since we are going to...

4 min read

Declaring an Array in Python

An array is a container used to store the same type of elements such as integer, float, and character type. An Array is one of the most important parts of data structures. In...

4 min read

Python Flask - ImmutableMultiDict

MultiDict is a sub-class of Dictionary that can contain multiple values for the same key, unlike normal Dictionaries. It is used because some form elements have multiple values...

2 min read

Handling 404 Error in Flask

Prerequisite: Creating simple application in Flask A 404 Error is showed whenever a page is not found. Maybe the owner changed its URL and forgot to change the link or...

4 min read

Python | Using for loop in Flask

26/11/2024, 22:37

Declaring Models in Flask - GeeksforGeeks

Prerequisite: HTML Basics, Python Basics, Flask It is not possible to write front-end course every time user make changes in his/her profile. We use a template and it...

3 min read

Article Tags :

Python

Geeks Premier League

Geeks Premier League 2023

Python Flask

+1 More

Practice Tags :

python



Corporate & Communications Address:-
A-143, 9th Floor, Sovereign Corporate
Tower, Sector- 136, Noida, Uttar Pradesh
(201305) | Registered Address:- K 061,
Tower K, Gulshan Vivante Apartment,
Sector 137, Noida, Gautam Buddh
Nagar, Uttar Pradesh, 201305



Company

- About Us
- Legal
- In Media
- Contact Us
- Advertise with us
- GFG Corporate Solution
- Placement Training Program
- GeeksforGeeks Community

Languages

- Python
- Java
- C++
- PHP
- GoLang
- SQL
- R Language
- Android Tutorial
- Tutorials Archive

DSA

- Data Structures
- Algorithms
- DSA for Beginners
- Basic DSA Problems
- DSA Roadmap
- Top 100 DSA Interview Problems
- DSA Roadmap by Sandeep Jain

Data Science & ML

- Data Science With Python
- Data Science For Beginner
- Machine Learning
- ML Maths
- Data Visualisation
- Pandas
- NumPy

[All Cheat Sheets](#)[NLP](#)[Deep Learning](#)

Web Technologies

[HTML](#)[CSS](#)[JavaScript](#)[TypeScript](#)[ReactJS](#)[NextJS](#)[Bootstrap](#)[Web Design](#)

Computer Science

[Operating Systems](#)[Computer Network](#)[Database Management System](#)[Software Engineering](#)[Digital Logic Design](#)[Engineering Maths](#)[Software Development](#)[Software Testing](#)

System Design

[High Level Design](#)[Low Level Design](#)[UML Diagrams](#)[Interview Guide](#)[Design Patterns](#)[OOAD](#)[System Design Bootcamp](#)[Interview Questions](#)

School Subjects

[Mathematics](#)[Physics](#)[Chemistry](#)[Biology](#)[Social Science](#)[English Grammar](#)[Commerce](#)[World GK](#)

Python Tutorial

[Python Programming Examples](#)[Python Projects](#)[Python Tkinter](#)[Web Scraping](#)[OpenCV Tutorial](#)[Python Interview Question](#)[Django](#)

DevOps

[Git](#)[Linux](#)[AWS](#)[Docker](#)[Kubernetes](#)[Azure](#)[GCP](#)[DevOps Roadmap](#)

Interview Preparation

[Competitive Programming](#)[Top DS or Algo for CP](#)[Company-Wise Recruitment Process](#)[Company-Wise Preparation](#)[Aptitude Preparation](#)[Puzzles](#)

GeeksforGeeks Videos

[DSA](#)[Python](#)[Java](#)[C++](#)[Web Development](#)[Data Science](#)[CS Subjects](#)

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved