



[Flask Templates](#) [Jinja2](#) [Flask-REST API](#) [Python SQLAlchemy](#) [Flask Bcrypt](#) [Flask Cookies](#) [Json](#) [Postman](#)

Last Updated : 17 May, 2024

Flask is a micro web framework written in python. Micro-framework is normally a framework with little to no dependencies on external libraries. Though being a micro framework almost everything can be implemented using python libraries and other dependencies when and as required.

Table of Content

- [Installing Flask](#)
- [Creating app.py](#)
- [Setting Up SQLAlchemy](#)
- [Creating Models](#)
- [Creating the database](#)
- [Making Migrations in database](#)
- [Creating the Index Page Of the Application](#)
- [Creating HTML page for form](#)
- [Function to add data using the form to the database](#)
- [Display data on Index Page](#)
- [Deleting data from our database](#)

In this article, we will be building a Flask application that **takes data** in a form from the user and then **displays** it on another page on the website. We can also **delete** the data. We won't focus on the front-end part rather we will be just coding the backend for the web application.

Installing Flask

In any directory where you feel comfortable create a folder and open the command line in the directory. Create a python virtual environment using the command below.

```
python -m venv <name>
```

Once the command is done running activate the virtual environment using the command below.

```
<name>\scripts\activate
```

Now, install Flask using pip(package installer for python). Simply run the command below.

```
pip install Flask
```

Creating app.py

Once the installation is done create a file name app.py and open it in your favorite editor. To check whether Flask has been properly installed you can run the following code.

Python



```
1  from flask import Flask
2  app = Flask(__name__)
3
4
5  '''If everything works fine you will get a
6  message that Flask is working on the first
7  page of the application
8  '''
9
10 @app.route('/')
11 def check():
12     return 'Flask is working'
13
14
15 if __name__ == '__main__':
16     app.run()
```

Output:

Flask is working

Setting Up SQLAlchemy

Now, let's move on to creating a **database for our application**. For the purpose of this article, we will be using SQLAlchemy a database toolkit, and an ORM(Object Relational Mapper). We will be using pip again to install SQLAlchemy. The command is as follows,

```
pip install flask-sqlalchemy
```

In your app.py file import SQLAlchemy as shown in the below code. We also need to add a configuration setting to our application so that we can use SQLite database in our application. We also need to create an SQLAlchemy database instance which is as simple as creating an object.

Python



```
1  from flask import Flask
2  from flask_sqlalchemy import SQLAlchemy
3
4  app = Flask(__name__)
5  app.debug = True
6
7  # adding configuration for using a sqlite database
8  app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
9
10 # Creating an SQLAlchemy instance
11 db = SQLAlchemy(app)
12
13 if __name__ == '__main__':
14     app.run()
```

Creating Models

In sqlalchemy we use classes to create our database structure. In our application, we will create a Profile table that will be responsible for holding the user's id, first name, last name, and age.

Python



```
1  from flask import Flask, request, redirect
2  from flask.templating import render_template
3  from flask_sqlalchemy import SQLAlchemy
4
5  app = Flask(__name__)
6  app.debug = True
7
8  # adding configuration for using a sqlite database
9  app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
10
11 # Creating an SQLAlchemy instance
12 db = SQLAlchemy(app)
13
14 # Models
15 class Profile(db.Model):
16     # Id : Field which stores unique id for every row in
17     # database table.
18     # first_name: Used to store the first name if the user
19     # last_name: Used to store last name of the user
20     # Age: Used to store the age of the user
21     id = db.Column(db.Integer, primary_key=True)
22     first_name = db.Column(db.String(20), unique=False,
23 nullable=False)
24     last_name = db.Column(db.String(20), unique=False,
25 nullable=False)
26     age = db.Column(db.Integer, nullable=False)
27
28     # repr method represents how one object of this
29     # datatable
30     # will look like
31     def __repr__(self):
32         return f"Name : {self.first_name}, Age: {self.age}"
33
34 if __name__ == '__main__':
```

```
app.run()
```

The table below explains some of the keywords used in the model class.

Column	used to create a new column in the database table
Integer	An integer data field
primary_key	If set to True for a field ensures that the field can be used to uniquely identify objects of the data table.
String	An string data field. String(<maximum length>)
unique	If set to True it ensures that every data in that field in unique.
nullable	If set to False it ensures that the data in the field cannot be null.
__repr__	Function used to represent objects of the data table.

Creating the database

In the command line which is navigated to the project directory and virtual environment running, we need to run the following commands.

```
python
```

The above command will initiate a python bash in your command line where you can use further lines of code to create your data table according to your model class in your database.

```
from app import db
db.create_all()
```

After the commands, the response would look like something in the picture and in your project directory you will notice a new file named 'site.db'.

```

Select C:\Windows\System32\cmd.exe - python
(vir) C:\Users\91747\Desktop\flask-app>python
Python 3.9.5 (tags/v3.9.5:0a7dcbb, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from app import db
C:\Users\91747\Desktop\flask-app\vir\lib\site-packages\flask_sqlalchemy\_init_.py:872: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True or False to suppress this warning.
  warnings.warn(FSADeprecationWarning(
>>> db.create_all()
>>>

```

Making Migrations in database

Install Flask-Migrate using pip

```
pip install Flask-Migrate
```

Now, in your app.py add two lines, the code being as follows,

Python



```

1 # Import for Migrations
2 from flask_migrate import Migrate, migrate
3
4 # Settings for migrations
5 migrate = Migrate(app, db)

```

Now to create migrations we run the following commands one after the other.

```
flask db init
```

```

/home/nikhil/PycharmProjects/gfg/venv/lib/python3.8/site-packages/flask_sqlalchemy/_init_.py:872: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future
  warnings.warn(FSADeprecationWarning(
Set it to True or False to suppress this warning.
warnings.warn(FSADeprecationWarning(
Creating directory /home/nikhil/PycharmProjects/gfg/migrations ... done
Creating directory /home/nikhil/PycharmProjects/gfg/migrations/versions ... done
Generating /home/nikhil/PycharmProjects/gfg/migrations/alembic.ini ... done
Generating /home/nikhil/PycharmProjects/gfg/migrations/env.py ... done
Generating /home/nikhil/PycharmProjects/gfg/migrations/script.py.mako ... done
Generating /home/nikhil/PycharmProjects/gfg/migrations/README ... done
Please edit configuration/connection/logging settings in '/home/nikhil/PycharmProjects/gfg/migrations/alembic.ini' before proceeding.

```

flask db init

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

```
(venv) nikhil@nikhil-Lenovo-Ideapad-330-15IKB:~/PycharmProjects/gfg$ flask db migrate -m "Initial migration"
/home/nikhil/PycharmProjects/gfg/venv/lib/python3.8/site-packages/flask_sqlalchemy/_init_.py:872: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future
  . Set it to True or False to suppress this warning.
  warnings.warn(FSADeprecationWarning(
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [alembic.autogenerate.compare] Detected added table 'profile'
Generating /home/nikhil/PycharmProjects/gfg/migrations/versions/e365ebd20914_initial_migration.py ... done
```

flask db migrate -m "Initial migration"

flask db upgrade

```
(venv) nikhil@nikhil-Lenovo-Ideapad-330-15IKB:~/PycharmProjects/gfg$ flask db upgrade
/home/nikhil/PycharmProjects/gfg/venv/lib/python3.8/site-packages/flask_sqlalchemy/_init_.py:872: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future
  . Set it to True or False to suppress this warning.
  warnings.warn(FSADeprecationWarning(
INFO [alembic.runtime.migration] Context impl SQLiteImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [alembic.runtime.migration] Running upgrade -> e365ebd20914, Initial migration
(venv) nikhil@nikhil-Lenovo-Ideapad-330-15IKB:~/PycharmProjects/gfg$
```

flask db upgrade

Now we have successfully created the data table in our database.

Creating the Index Page Of the Application

Before moving forward and building our form let's create an index page for our website. The HTML file is always stored inside a folder in the parent directory of the application named **'templates'**. Inside the templates folder create a file named index.html and paste the below code for now. We will go back to adding more code into our index file as we move on.

HTML



```
1 <html>
2   <head>
3     <title>Index Page</title>
4   </head>
5   <body>
6     <h3>Profiles</h3>
7   </body>
8 </html>
```

In the app.py add a small function that will render an HTML page at a specific route specified in app.route.

Python



```
1 from flask import Flask, request, redirect
2 from flask.templating import render_template
3 from flask_sqlalchemy import SQLAlchemy
4
5 app = Flask(__name__)
6 app.debug = True
7
8 # adding configuration for using a sqlite database
9 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
10
11 # Creating an SQLAlchemy instance
12 db = SQLAlchemy(app)
13
14 # Models
15 class Profile(db.Model):
16
17     id = db.Column(db.Integer, primary_key=True)
18     first_name = db.Column(db.String(20), unique=False,
19 nullable=False)
19     last_name = db.Column(db.String(20), unique=False,
20 nullable=False)
21     age = db.Column(db.Integer, nullable=False)
22
23     def __repr__(self):
24         return f"Name : {self.first_name}, Age: {self.age}"
25
26 # function to render index page
27 @app.route('/')
28 def index():
29     return render_template('index.html')
30
31 if __name__ == '__main__':
32     app.run()
```

To test whether everything is working fine you can run your application using the command

```
python app.py
```

The command will set up a local server at <http://localhost:5000>.

Output:

Profiles

[ADD](#)

Id First Name Last Name Age #

Creating HTML page for form

We will be creating an HTML page in which our form will be rendered. Create an HTML file named `add_profile` in your templates folder. The HTML code is as follows. The important points in the code will be **highlighted as you read on**.

HTML



```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Add Profile</title>
5   </head>
6   <body>
7     <h3>Profile form</h3>
8     <form action="/add" method="POST">
9       <label>First Name</label>
10      <input type="text" name="first_name"
placeholder="first name...">
11      <label>Last Name</label>
12      <input type="text" name="last_name"
placeholder="last name...">
13      <label>Age</label>
14      <input type="number" name="age"
placeholder="age..">
15      <button type="submit">Add</button>
16    </form>
17  </body>
18 </html>
```

Adding a function in our application to render the form page

In our app.py file, we will add the following function. At route or site path 'http://localhost:5000/add_data' the page will be rendered.

Python



```
1 @app.route('/add_data')
2 def add_data():
3     return render_template('add_profile.html')
```

To check whether the code is working fine or not, you can run the following command to start the local server.

```
python app.py
```

Now, visit http://localhost:5000/add_data and you will be able to see the form.

Output:

Profile form

First Name Last Name Age

Function to add data using the form to the database

To add data to the database we will be using the “**POST**” method. POST is used to send data to a server to create/update a resource. In flask where we specify our route that is app.route we can also specify the HTTP methods there. Then inside the function, we create variables to store data and use **request objects** to procure data from the form.

Note: The name used in the input tags in the HTML file has to be the same one that is being used in this function,

For example,

```
<input type="number" name="age" placeholder="age..">
```

“age” should also be used in the python function as,

```
age = request.form.get("age")
```

Then we move on to create an object of the Profile class and store it in our database using database sessions.

Python



```
1  # function to add profiles
2  @app.route('/add', methods=["POST"])
3  def profile():
4
5      # In this function we will input data from the
6      # form page and store it in our database.
7      # Remember that inside the get the name should
8      # exactly be the same as that in the html
9      # input fields
10     first_name = request.form.get("first_name")
11     last_name = request.form.get("last_name")
12     age = request.form.get("age")
13
14     # create an object of the Profile class of models
15     # and store data as a row in our datatable
16     if first_name != '' and last_name != '' and age is not
None:
17         p = Profile(first_name=first_name,
last_name=last_name, age=age)
18         db.session.add(p)
19         db.session.commit()
20         return redirect('/')
21     else:
22         return redirect('/')
```

Once the function is executed it redirects us back to the index page of the application.

Display data on Index Page

On our index page now, we will be displaying all the data that has been stored in our data table. We will be using ‘**Profile.query.all()**’ to query all the objects

of the Profile class and then use **Jinja templating language** to display it dynamically on our index HTML file.

Update your index file as follows. The delete function will be written later on in this article. For now, we will query all the data from the data table and display it on our home page.

HTML



```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Index Page</title>
5      </head>
6      <body>
7          <h3>Profiles</h3>
8          <a href="/add_data">ADD</a>
9          <br>
10         <table>
11             <thead>
12                 <th>Id</th>
13                 <th>First Name</th>
14                 <th>Last Name</th>
15                 <th>Age</th>
16                 <th>#</th>
17             </thead>
18             {% for data in profiles %}
19             <tbody>
20                 <td>{{data.id}}</td>
21                 <td>{{data.first_name}}</td>
22                 <td>{{data.last_name}}</td>
23                 <td>{{data.age}}</td>
24                 <td><a href="/delete/{{data.id}}"
type="button">Delete</a></td>
25             </tbody>
26             {% endfor %}
27         </table>
28     </body>
29 </html>
```

We loop through every object in profiles that we pass down to our template in our index function and print all its data in a tabular form. The index function in our app.py is updated as follows.

Python

```
1 @app.route('/')
2 def index():
3     # Query all data and then pass it to the template
4     profiles = Profile.query.all()
5     return render_template('index.html', profiles=profiles)
```

Deleting data from our database

To delete data we have already used an anchor tag in our table and now we will just be associating a function with it.

Python

```
1 @app.route('/delete/<int:id>')
2 def erase(id):
3     # Deletes the data on the basis of unique id and
4     # redirects to home page
5     data = Profile.query.get(id)
6     db.session.delete(data)
7     db.session.commit()
8     return redirect('/')
```

The function queries data on the basis of id and then deletes it from our database.

Complete Code

The entire code for app.py, index.html, and add-profile.html has been given.

app.py

Python



```
from flask import Flask, request, redirect
2 from flask.templating import render_template
3 from flask_sqlalchemy import SQLAlchemy
4 from flask_migrate import Migrate, migrate
5
6 app = Flask(__name__)
7 app.debug = True
8
9 # adding configuration for using a sqlite database
10 app.config['SQLALCHEMY_DATABASE_URI'] =
    'sqlite:///site.db'
11
12 # Creating an SQLAlchemy instance
13 db = SQLAlchemy(app)
14
15 # Settings for migrations
16 migrate = Migrate(app, db)
17
18 # Models
19 class Profile(db.Model):
20     # Id : Field which stores unique id for every row in
21     # database table.
22     # first_name: Used to store the first name if the user
23     # last_name: Used to store last name of the user
24     # Age: Used to store the age of the user
25     id = db.Column(db.Integer, primary_key=True)
26     first_name = db.Column(db.String(20), unique=False,
        nullable=False)
27     last_name = db.Column(db.String(20), unique=False,
        nullable=False)
28     age = db.Column(db.Integer, nullable=False)
29
30     # repr method represents how one object of this
    datatable
31     # will look like
32     def __repr__(self):
33         return f"Name : {self.first_name}, Age:
        {self.age}"
34
35 # function to render index page
36 @app.route('/')
37 def index():
38     profiles = Profile.query.all()
```

```
        return render_template('index.html',
                               profiles=profiles)
40
41 @app.route('/add_data')
42 def add_data():
43     return render_template('add_profile.html')
44
45 # function to add profiles
46 @app.route('/add', methods=["POST"])
47 def profile():
48     # In this function we will input data from the
49     # form page and store it in our database. Remember
50     # that inside the get the name should exactly be the
    same
51     # as that in the html input fields
52     first_name = request.form.get("first_name")
53     last_name = request.form.get("last_name")
54     age = request.form.get("age")
55
56     # create an object of the Profile class of models and
57     # store data as a row in our datatable
58     if first_name != '' and last_name != '' and age is not
    None:
59         p = Profile(first_name=first_name,
    last_name=last_name, age=age)
60         db.session.add(p)
61         db.session.commit()
62         return redirect('/')
63     else:
64         return redirect('/')
65
66 @app.route('/delete/<int:id>')
67 def erase(id):
68
69     # deletes the data on the basis of unique id and
70     # directs to home page
71     data = Profile.query.get(id)
72     db.session.delete(data)
73     db.session.commit()
74     return redirect('/')
75
76 if __name__ == '__main__':
77     app.run()
```

index.html

HTML



```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Index Page</title>
5   </head>
6   <body>
7     <h3>Profiles</h3>
8     <a href="/add_data">ADD</a>
9     <br>
10    <table>
11      <thead>
12        <th>Id</th>
13        <th>First Name</th>
14        <th>Last Name</th>
15        <th>Age</th>
16        <th>#</th>
17      </thead>
18      {% for data in profiles %}
19      <tbody>
20        <td>{{data.id}}</td>
21        <td>{{data.first_name}}</td>
22        <td>{{data.last_name}}</td>
23        <td>{{data.age}}</td>
24        <td><a href="/delete/{{data.id}}"
type="button">Delete</a></td>
25      </tbody>
26      {% endfor %}
27    </table>
28  </body>
29 </html>
```

add_profile.html

HTML

skills, this course is the perfect fit. Enroll now and master Python, the language of the future!

A aman...



15

Previous Article

How to return a JSON response from a Flask API ?

Next Article

How to Build a Web App using Flask and SQLite in Python

Similar Reads

How to execute raw SQL in Flask-SQLAlchemy app

In this article, we are going to see how to execute raw SQL in Flask-SQLAlchemy using Python. Installing requirements Install the Flask and Flask-SQLAlchemy...

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 know...

8 min read

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...

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...

4 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...

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 f...

11 min read

Read SQL database table into a Pandas DataFrame using SQLAlchemy

To read sql table into a DataFrame using only the table name, without executing any query we use read_sql_table() method in Pandas. This function does not...

2 min read

Connecting to SQL Database using SQLAlchemy in Python

In this article, we will see how to connect to an SQL database using SQLAlchemy in Python. To connect to a SQL database using SQLAlchemy we will require the...

3 min read

Connecting Pandas to a Database with SQLAlchemy

In this article, we will discuss how to connect pandas to a database and perform database operations using SQLAlchemy. The first step is to establish a connecti...

3 min read

Flask login without database - Python

In this article, we will talk about Python-based Flask login without a database in this article. In order to use Flask login without a database in this method basical...

4 min read

Article Tags : [Python](#) [Python Flask](#) [Python-SQLAlchemy](#)

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
All Cheat Sheets

Data Science & ML

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

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science

DevOps

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

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