National University of Computer & Emerging
Sciences (NUCES) Islamabad

Secure Software Design – FALL 2024

Cyber Security Department

LAB 07

Instructor: Nawfal Waqar

Learning Outcomes

In this lab you are expected to learn the following:

- Building a Flask App from Scratch
- Making CRUD operations, using templates etc.

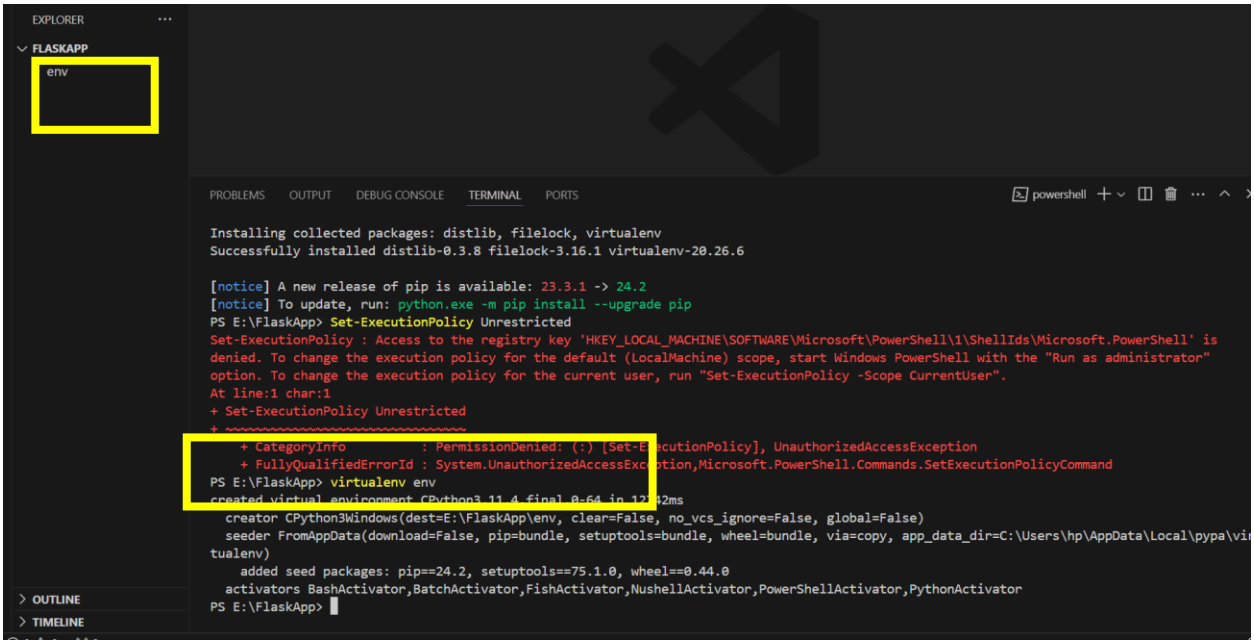
Setup for building a Flask Application

1. Install Visual Studio Code
2. Install latest version of Python on your Computer.
3. Install virtual environment by this command: pip install virtualenv
4. If it is giving some error, do this first,

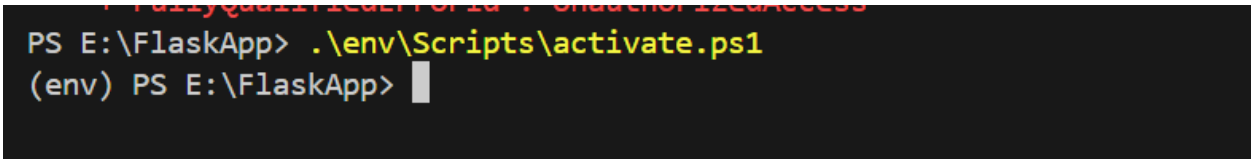
   Open Power Shell Admin and run this command:

   Set-ExecutionPolicy Unrestricted
5. Now create a virtual environment using this command:

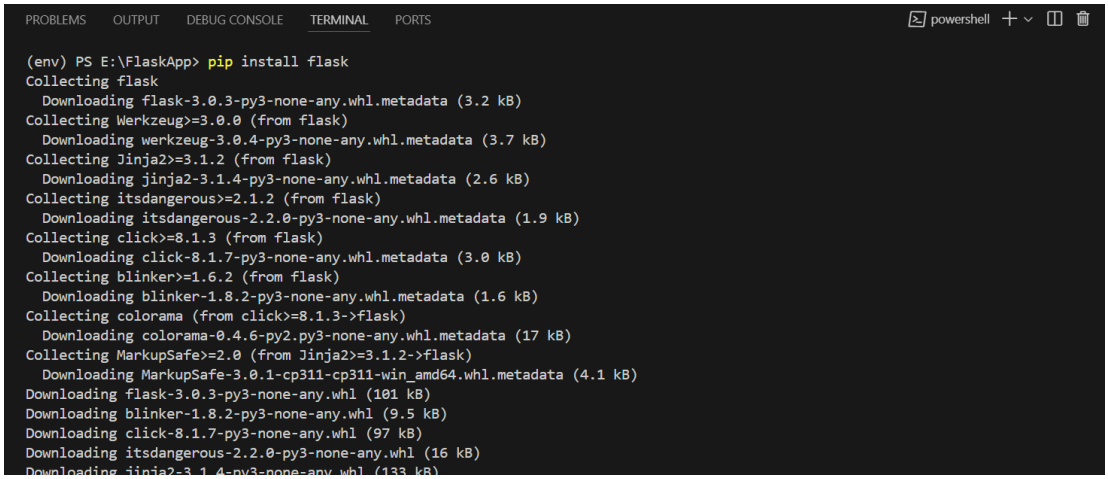
   virtualenv env



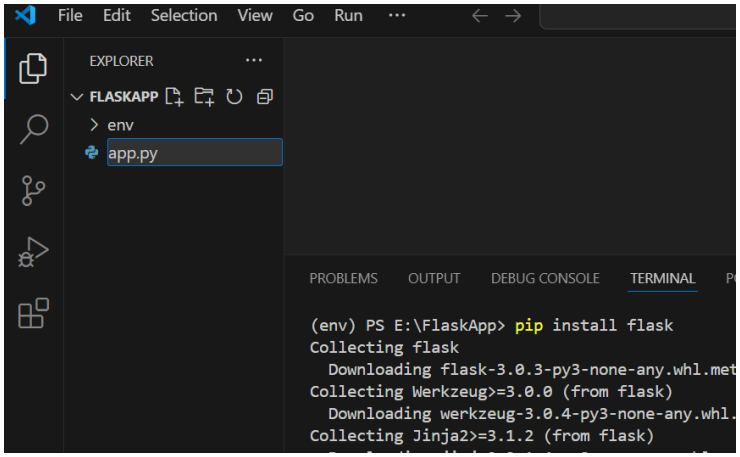
   It will create a subfolder in your main folder named “env”, now we have a virtual environment to work in
6. Activating Virtual Environment:



7. Installing Flask in this environment:



8. Make a new file name **app.py**



9. Creating A minimal Flask App
    i. Go to https://flask.palletsprojects.com/en/1.1.x/quickstart/
    ii. Copy the following code and paste it in your app.py file, it is making a simple hello world program in python.

- A Minimal Application

A minimal Flask application looks something like this:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

    iii. Also write **main** to call the function:

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'



# writing main function to call the above defined function
# otherwise the program won't do anything
if __name__ == "__main__":
    app.run(debug=True)
```

iv. Now run the file using the following command:

```
(env) PS E:\FlaskApp> python .\app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
Python 3.11.4
(env) PS E:\FlaskApp> python .\app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
```

v. Open http://localhost:5000/ on your browser to see the app running.

Congratulations! You have made your first Flask App

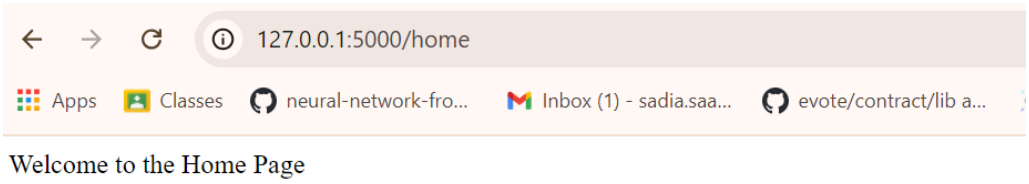
10. Creating another page in the application:

    Make another **app.route()** with your desired name

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route('/home')
def home():
    return 'Welcome to the Home Page'



# writing main function to call the above defined function
# otherwise the program won't do anything
if __name__ == "__main__":
    app.run(debug=True)
```

    Save the code and the changes will be reflected in the app simultaneously

    run localhost/home

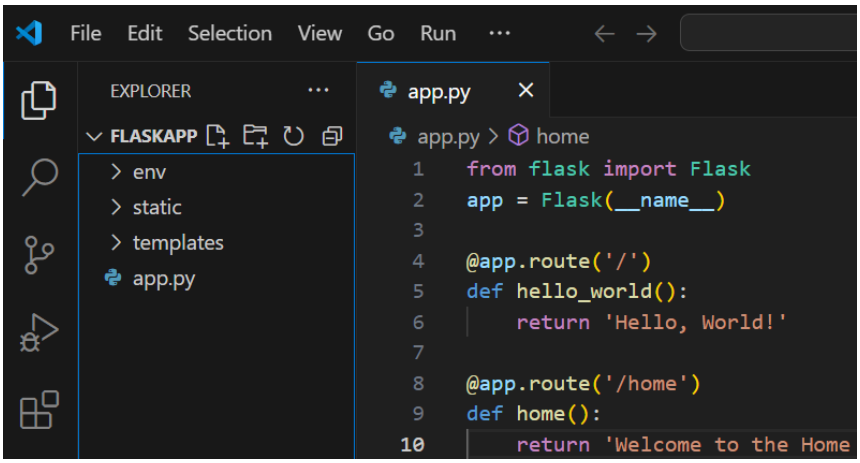


11. Now make 2 directories in the FLASKAPP folder:

    static (to keep all the static files e.g.: images, text files etc.)

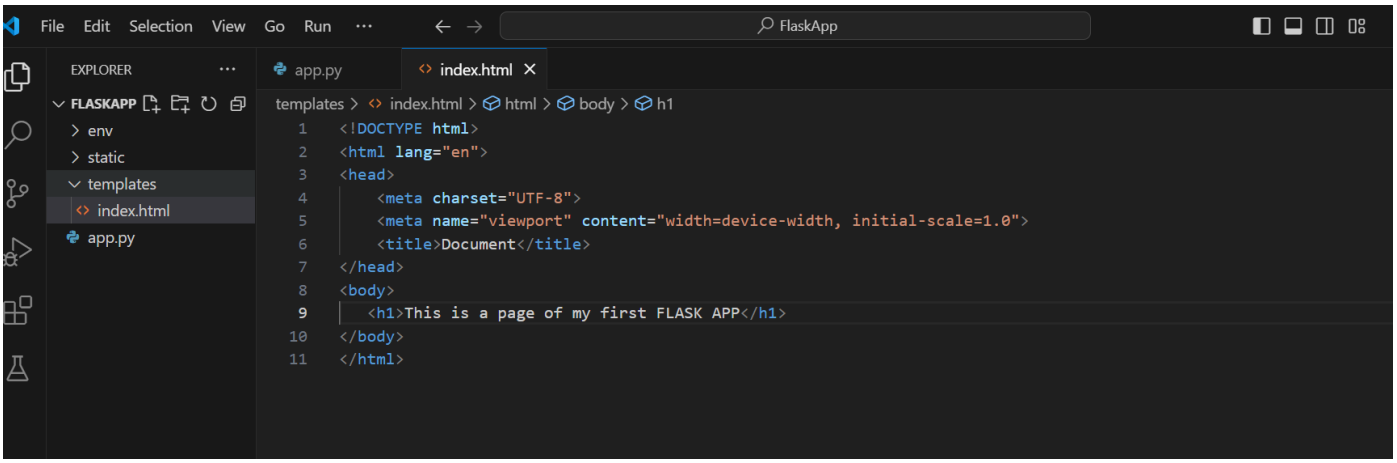
    templates (to keep prebuilt templates)

    Remember! Do not make them inside the env folder or nested with eachother



12. Make a file **index.html** in the templates folder

- Write **!** and Press **Enter,** it will write the basic starting code for you.
- Now write something in the **body tag** of the html file



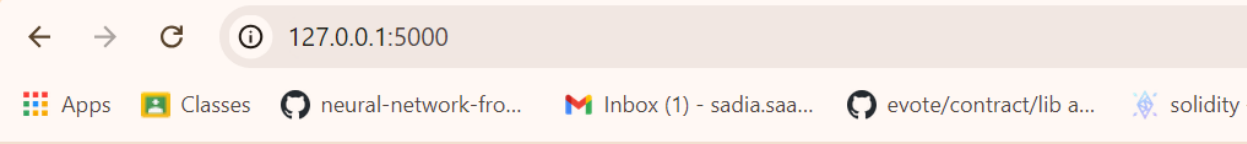
- Now we need to render this html so that it shows on the app
- Go to the **app.py** file and do the following
    - Import **render_template**
    - Write a return statement in **def hello_world ()**

app.py index.html

app.py > hello_world

```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def hello_world():
    return render_template('Index.html')
    #return 'Hello, World!'


@app.route('/home')
def home():
    return 'Welcome to the Home Page'

```

- Now if you go to the browser and refresh you will see the html page's content



This is a page of my first FLASK APP

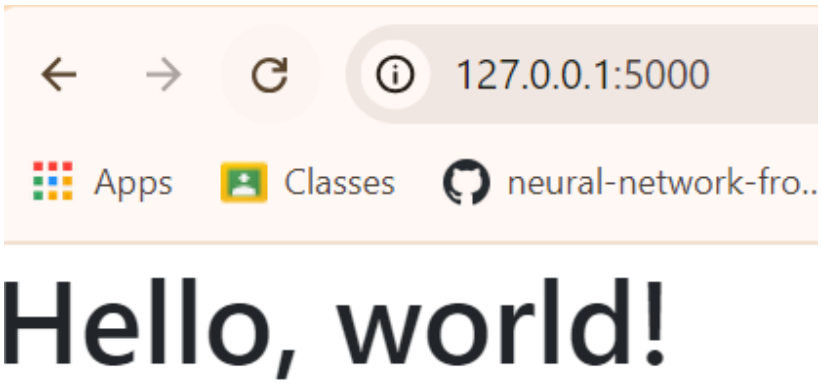
Beautification of the APP using HTML and CSS

- Go to https://getbootstrap.com/docs/5.3/getting-started/introduction/
- Copy the following code and paste it into your **index.html** file

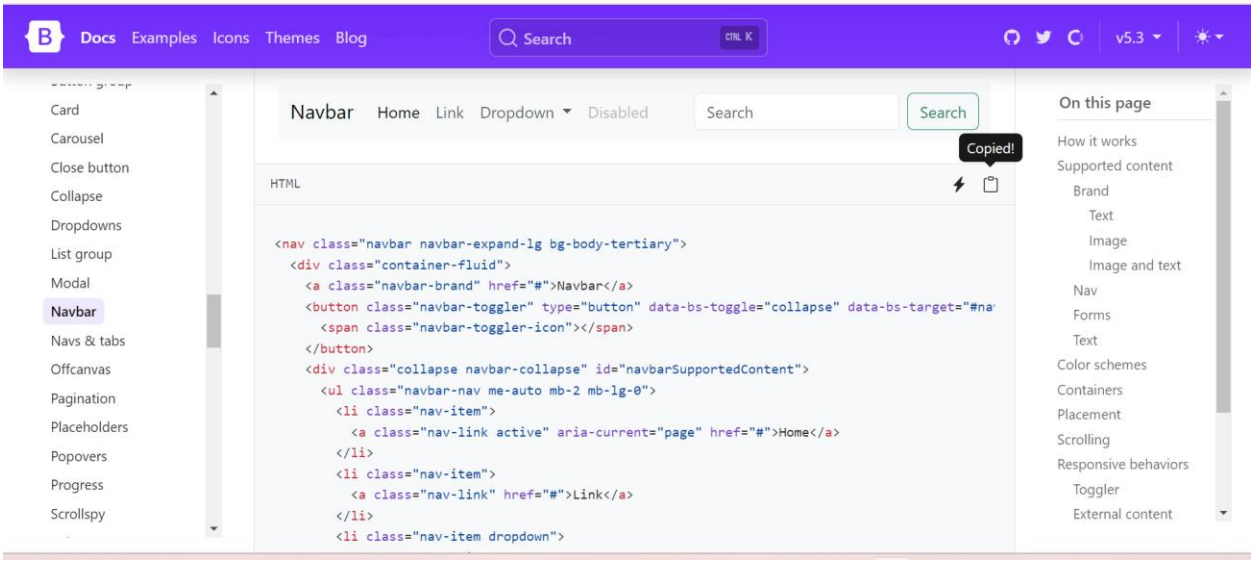
2. **Include Bootstrap's CSS and JS.** Place the `<link>` tag in the `<head>` for our CSS, and the `<script>` tag for our JavaScript bundle (including Popper for positioning dropdowns, poppers, and tooltips) before the closing `</body>`. Learn more about our CDN links.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.cs
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle
  </body>
</html>
```

- Save the code and refresh the browser:



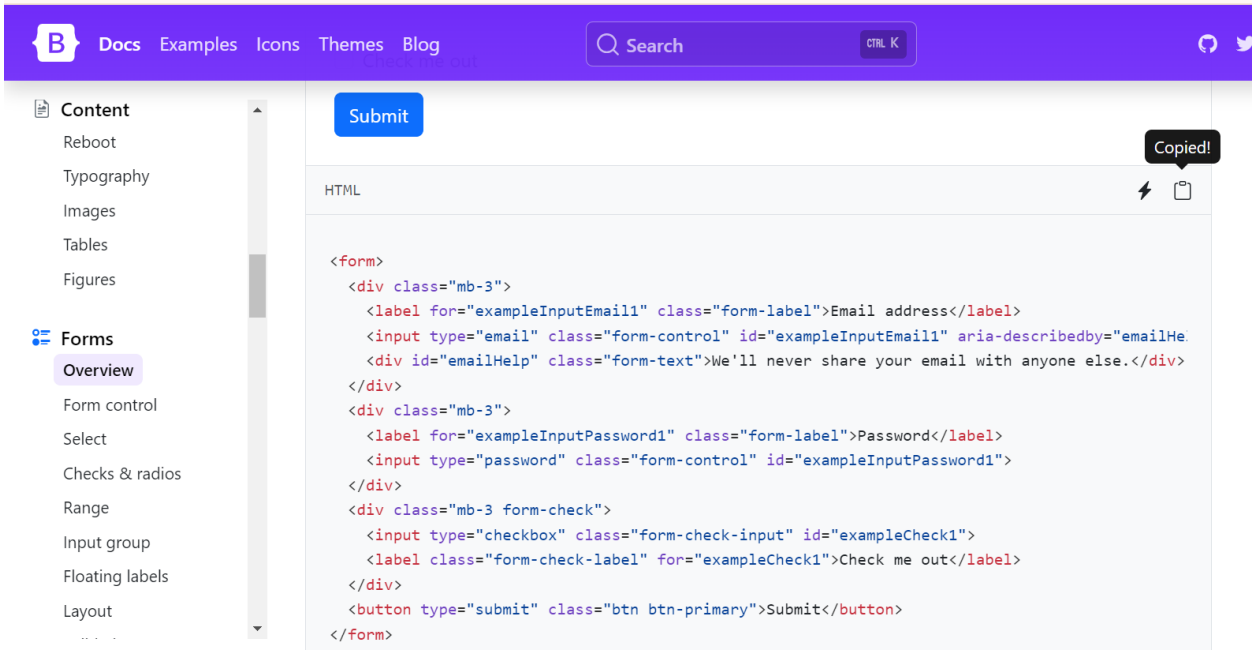
- Now, suppose we have to make a Navigation Bar/ Menu in our website. Go to the components section on getbootstrap.com and click on navbar, copy the given code.



- Paste the code in the **body** tag of your html file and save the file.

- Refresh the browser, now you have a navigation bar in your application.

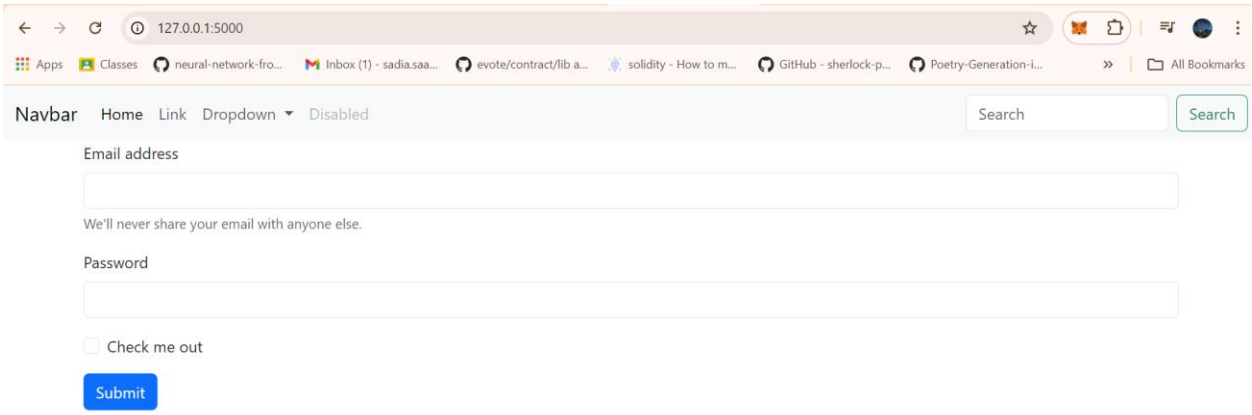
- Now if we want to add a form on our page, go to the forms section on the website and copy the following code:



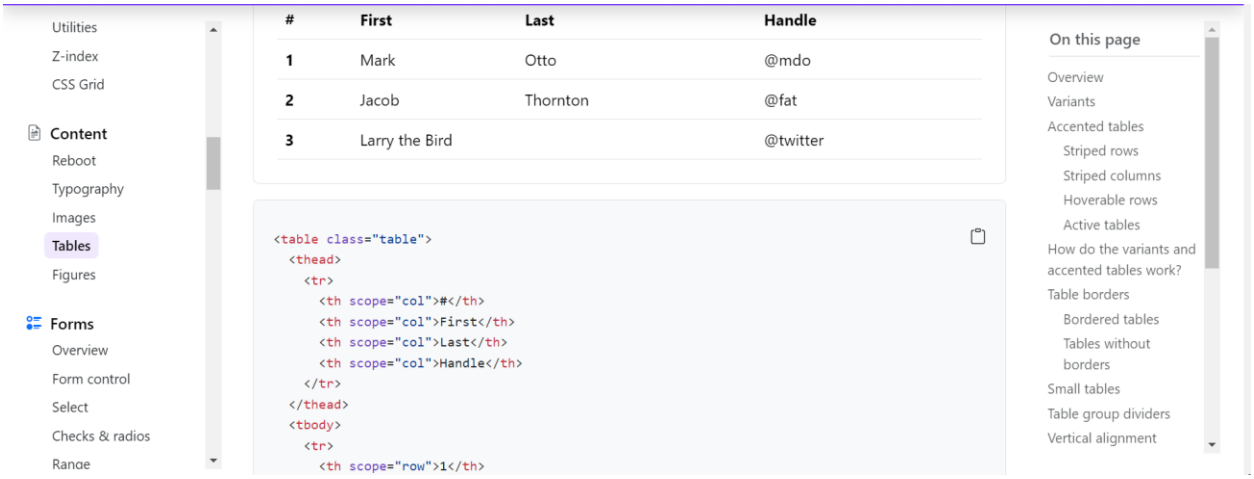
- In your Index.html file, make a container like this:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" re
  </head>
  <body>
    <nav class="navbar navbar-expand-lg bg-body-tertiary">···
      </nav>

      <div class = "container"> </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min
  </body>
```

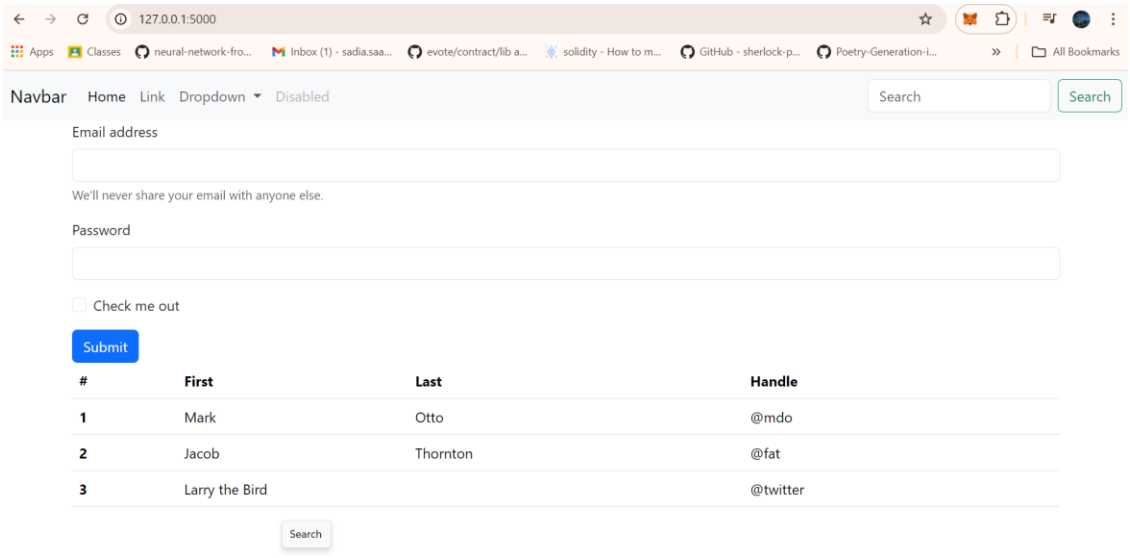
- Paste the copied code inside this div, save the file and refresh the browser. Now you have a form on your page:



- Now suppose we want to add a table to show some data, go the website and search for tables, copy the code for making a table:



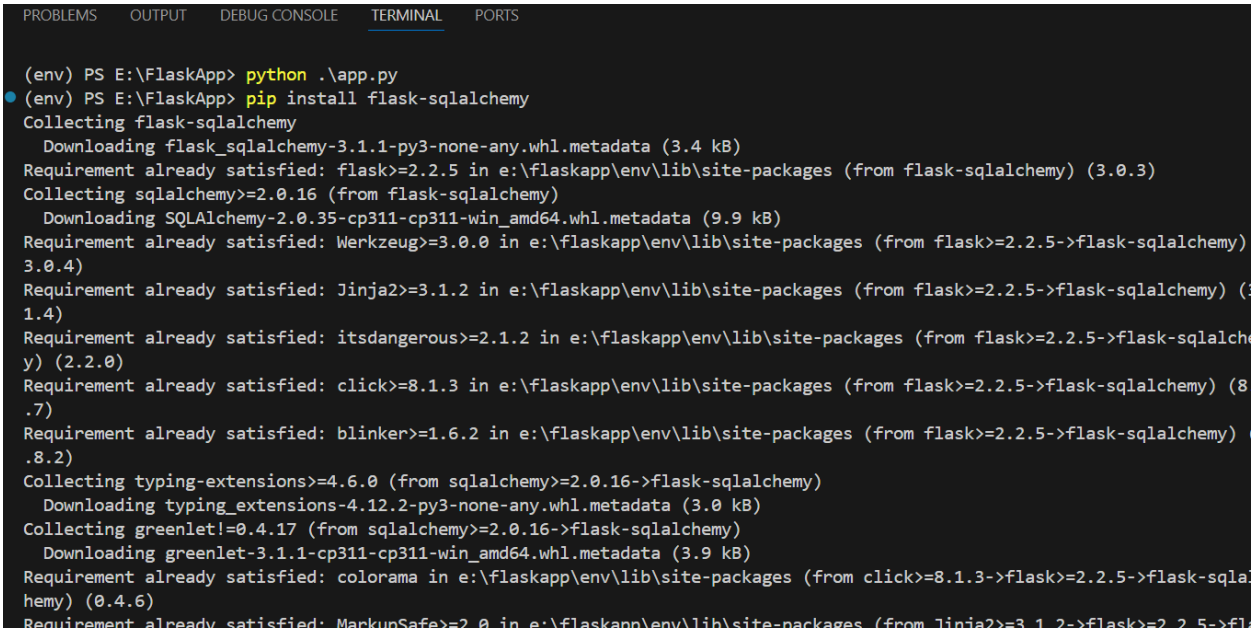
- Make another div container and paste this code there, save the code and refresh the browser, now we have a table



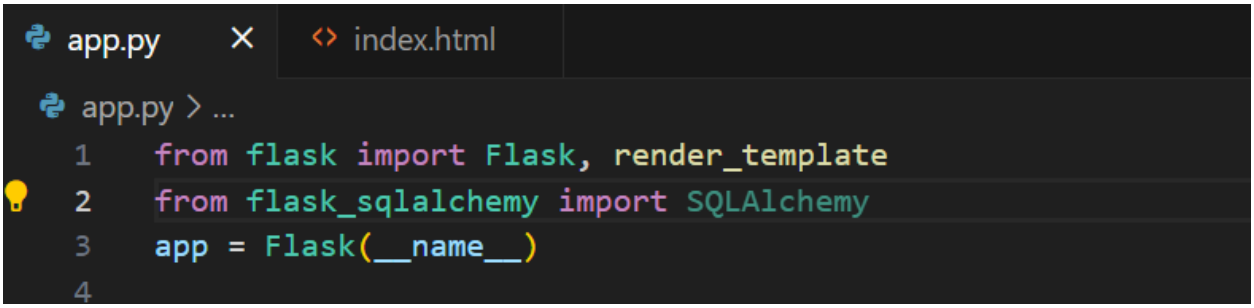
Creating a Database for our APP

- For FLASK, first we need to install SQL Alchemy, using the following command:

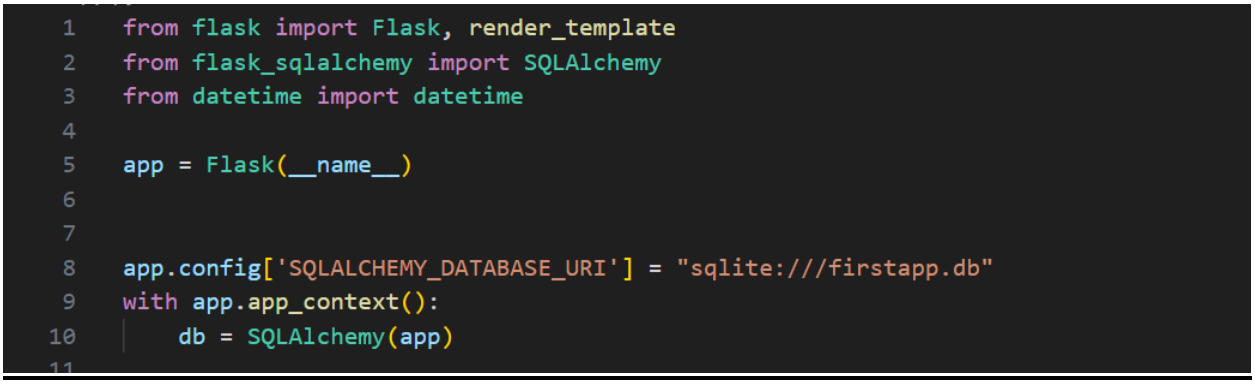
  pip install flask-sqlalchemy



- In app.py file, import sql alchemy:



- Now, initialize SQL Alchemy and connect it with SQL Lite:



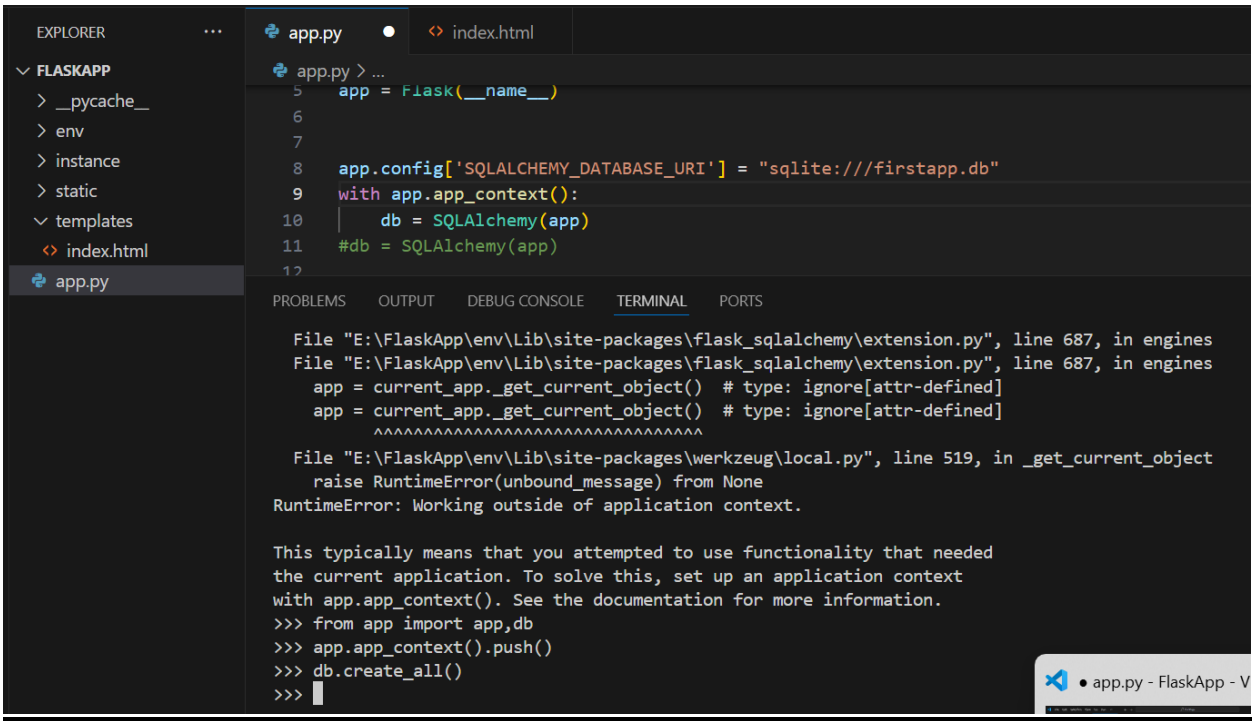
- In app.py file, make a class to define the structure of our data base like this:
  Here I have defined 5 columns, their names and data types like we do in SQL normally.
  __repr__ only returns the Serial Number and First Name when called.

```python
app.py > ...
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///firstapp.db"
with app.app_context():
    db = SQLAlchemy(app)



#db = SQLAlchemy(app)


# now making a class to define the structure of our db

class FirstApp(db.Model):
    sno = db.Column(db.Integer,primary_key=True, autoincrement=True)
    fname = db.Column(db.String(100), nullable=False)
    lname = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(200),nullable=False)


    def __repr__(self):
        return f"{self.sno} - {self.fname}"

@app.route('/')
def hello_world():
```

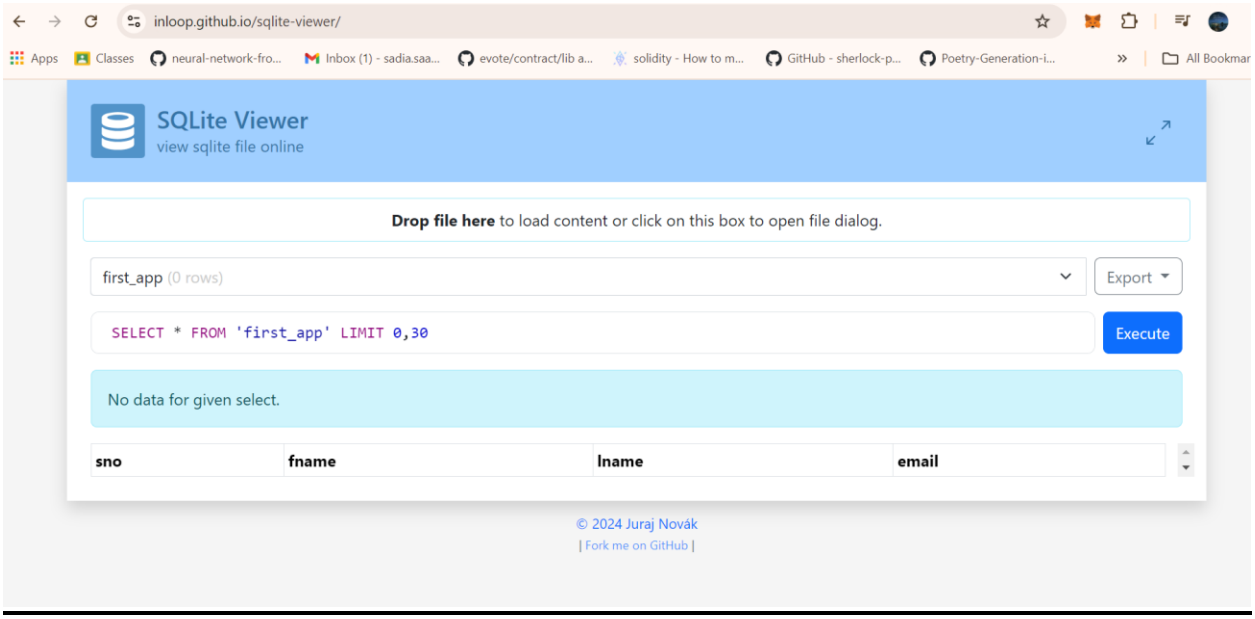
- Now to create db, open python interpreter:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

emy-3.1.1 greenlet-3.1.1 sqlalchemy-2.0.35 typing-extensions-4.12.2
(env) PS E:\FlaskApp> python
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- **Write the following commands:**
  from app import app,db
  app.app_context().push()
  db.create_all()
  It will create a folder names instance with your db file inside it.



- If we want to view our data base, open sql lite viewer on Google, drag and drop firstapp.db file here and see:

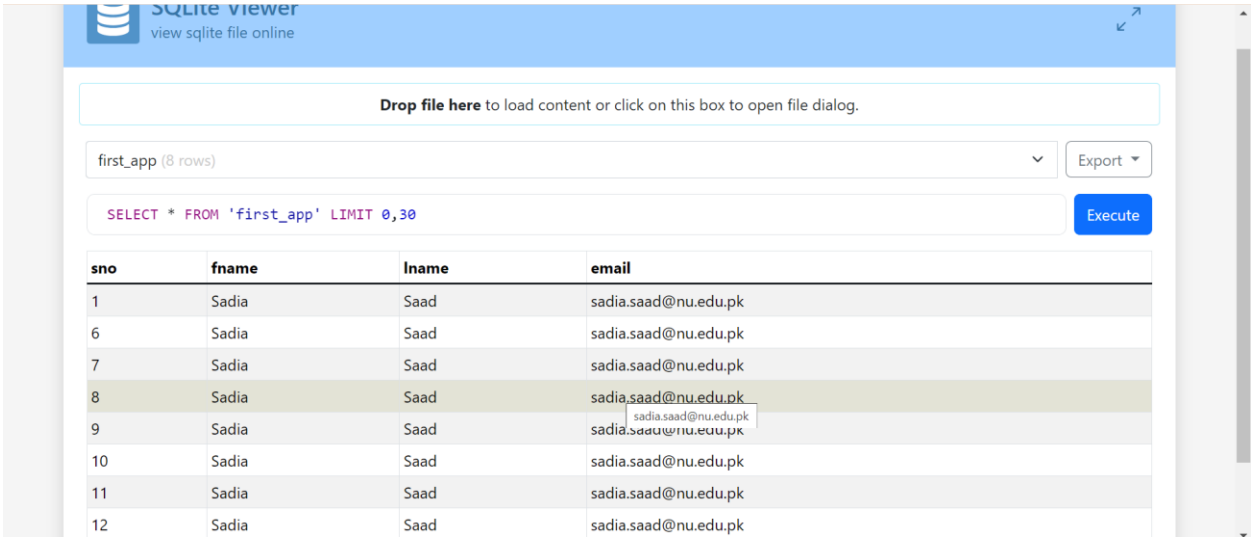


- Now exit from the python interpreter with the command **exit()**

- **Create an instance of db in app.route(), save and refresh the browser, with every refresh it will add an instance.**

```
28
29  @app.route('/')
30  def hello_world():
31
32      firstapp = FirstApp(fname="Sadia",lname="Saad")
33      db.session.add(firstapp)
34      db.session.commit()
35
36      return render_template('Index.html')
37      #return 'Hello, World!'
38
39
```

- Now see the firstapp.db in the sqllite viewer again:



- Here, install an extension to easily make and use templates for FLASK

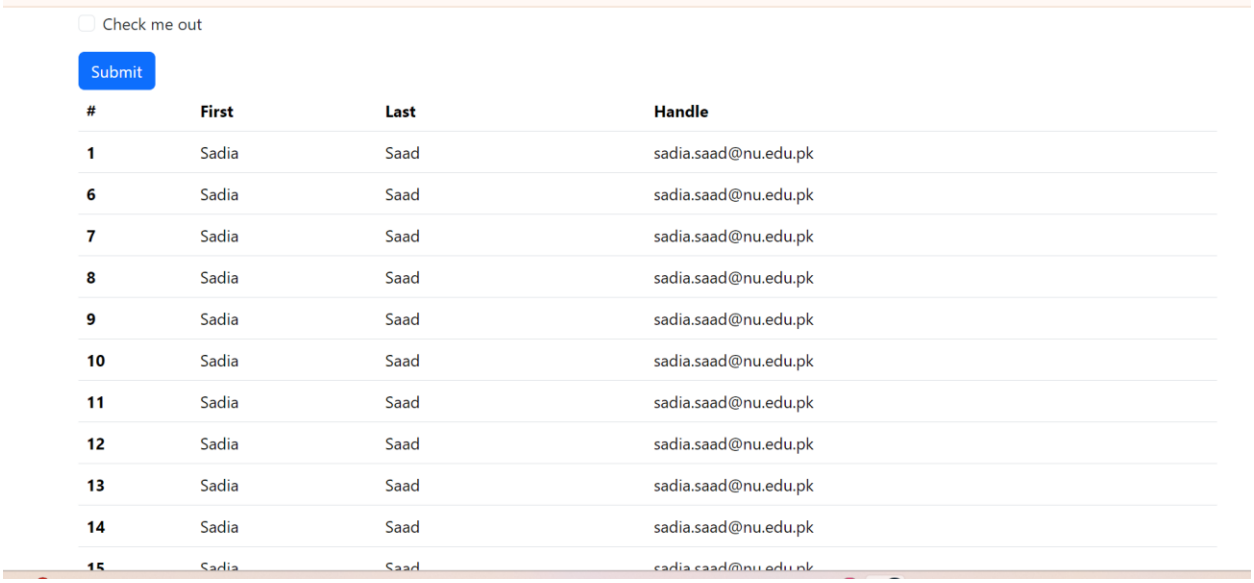
- To show all the records on the webpage do this, here we are getting all the records and sending them to the index.html page using return statement.

```
@app.route('/')
def hello_world():

    firstapp = FirstApp(fname="Sadia",lname="Saad",email="sa
    db.session.add(firstapp)
    db.session.commit()

    allpeople = FirstApp.query.all()
    #print(allpeople)

    return render_template('Index.html',allpeople=allpeople)
```

- In Index.html, Inside the tbody tag write the following loop counter,

```
      </tr>
    </thead>
    <tbody>

      {% for people in  allpeople %}
      <tr>
        <th scope="row">{{people.sno}}</th>
        <td>{{people.fname}}</td>
        <td>{{people.lname}}</td>
        <td>{{people.email}}</td>
      </tr>
      {% endfor %}

    <tr>
      <th scope="row">2</th>
```

- It will show all the entries in the table:



Make the forms Working

- In Index.html, previously used form tag, add the following:

- Now, in app.py, add the following lines in app.route(), Now save the file and refresh the browser, press the submit button, you will see POST written in the terminal showing that it is a POST request.

- Now save these variables and then add them to the data base at run time.

```
@app.route('/', methods = ['GET','POST'])
def hello_world():
    if request.method=='POST':
        # Check if form fields exist
        fname = request.form.get('fname')
        lname = request.form.get('lname')
        email = request.form.get('email')
        if fname and lname and email:
            # Create and commit new record to the database
            firstapp = FirstApp(fname=fname, lname=lname, email=email)
            db.session.add(firstapp)
            db.session.commit()
    allpeople = FirstApp.query.all()
    print(allpeople)

    return render_template('index.html',allpeople=allpeople)
```

- Save the code and refresh the page.

  Fill the form, click on submit.

  Changes will be reflected in the table:

First Name

xyz

We'll never share your email with anyone else.

Last Name

abc

Email address

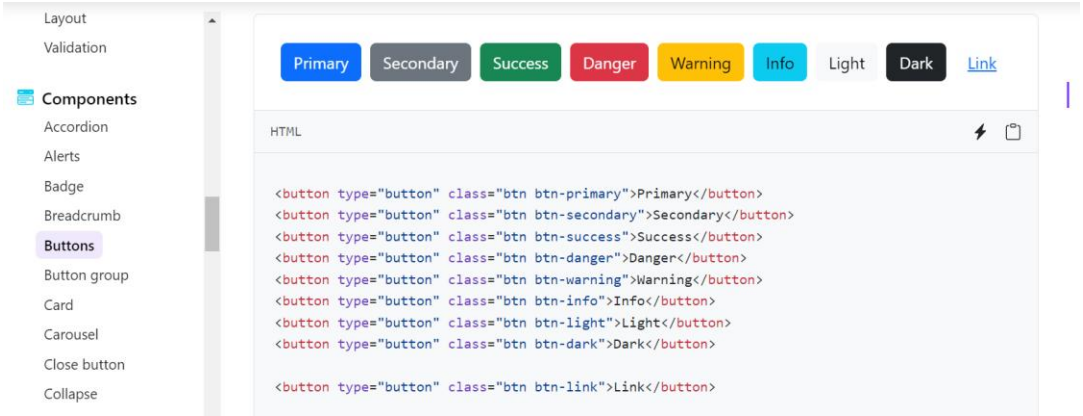
xyz@gmail.com

Submit

24	Sadia	Saad	sadia.saad@nu.edu.pk
25	Sadia	Saad	sadia.saad@nu.edu.pk
26	Sadia	Saad	sadia.saad@nu.edu.pk
27	Sadia	Saad	sadia.saad@nu.edu.pk
28	Sadia	Saad	sadia.saad@nu.edu.pk
29	laiba	imran	laiba.imran@gmail.com
30	xyz	abc	xyz@gmail.com
2	Jacob	Thornton	j@.com
3	Larry the Bird		

- To delete an entry from the form:

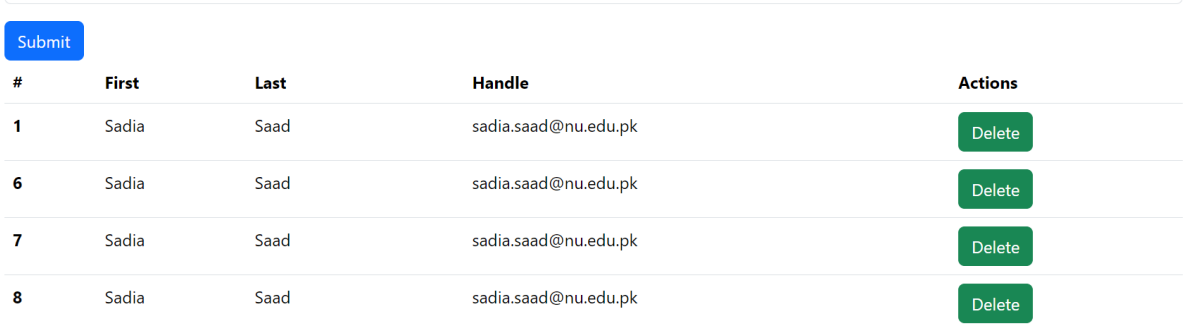
  Go to the bootstrap components, and select the code for any button:



- Paste it in index.html like the following code.
- Also add a new column

```
        <th scope="col"> Actions</th>
      </tr>
    <tbody>

      {% for people in  allpeople %}
      <tr>
          <th scope="row">{{people.sno}}</th>
          <td>{{people.fname}}</td>
          <td>{{people.email}}</td>
          <td> <a href="delete/{{people.sno}}" type="button" class="btn btn-success">Delete</button></td>
```

- Save it and refresh the browser.



- Add another button named update and make them smaller using the class tag.

  These are prebuilt tags in bootstrap:

```
{% for people in  allpeople %}
<tr>
    <th scope="row">{{people.sno}}</th>
    <td>{{people.fname}}</td>
    <td>{{people.lname}}</td>
    <td>{{people.email}}</td>
    <td> <a href="update/{{people.sno}}" type="button" class="btn btn-outline-dark btn-sm mx-1">Update</button>
        <a href="delete/{{people.sno}}" type="button" class="btn btn-outline-dark btn-sm mx-1">Delete</button></td>
```

- This is how the output looks like:

Submit

#	First	Last	Handle	Actions
1	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
6	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
7	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
8	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
9	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
10	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete

- Now make 2 methods for Update and Delete so that the buttons perform some functionality:
    - Import redirect at the start of the app.py file
    - For **Delete** we need an int argument that is the serial number.
    - And after deletion we redirect it to homepage

```
@app.route('/delete/<int:sno >')
def delete(sno):
    #first fetching the record with sno
    allpeople = FirstApp.query.filter_by(sno=sno).first()
    #now deleting the record
    db.session.delete(allpeople)
    db.session.commit()

    return redirect("/")
```

- Now if I press delete a record will be deleted:

Email

Submit

#	First	Last	Handle	Actions
6	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
7	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
8	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
9	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
10	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete
11	Sadia	Saad	sadia.saad@nu.edu.pk	Update Delete

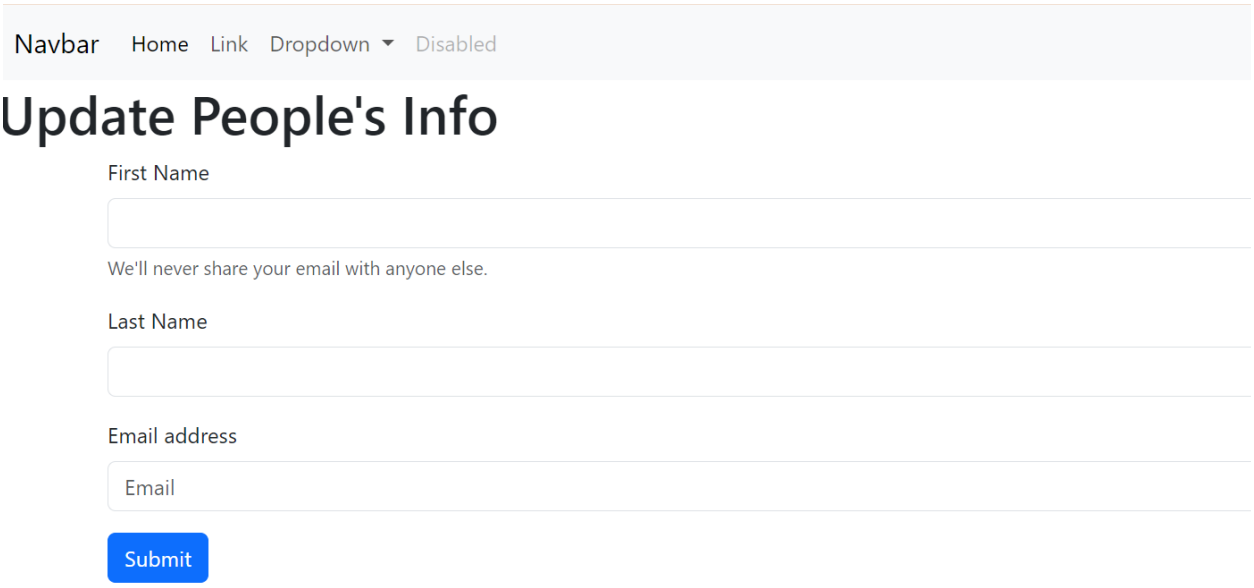
- For the Update function, make a function in app.py

```
@app.route('/update/<int:sno>')
def update(sno):
    #first fetching the record with sno
    allpeople = FirstApp.query.filter_by(sno=sno).first()

    return render_template('update.html',allpeople=allpeople)
```

- Make an empty update.html file inside templates folder

  Paste the code of index.html on to it, remove the table making part.

  Save the code, refresh the browser and click on update button, it will take you here:



- Add the following line with the form tag:

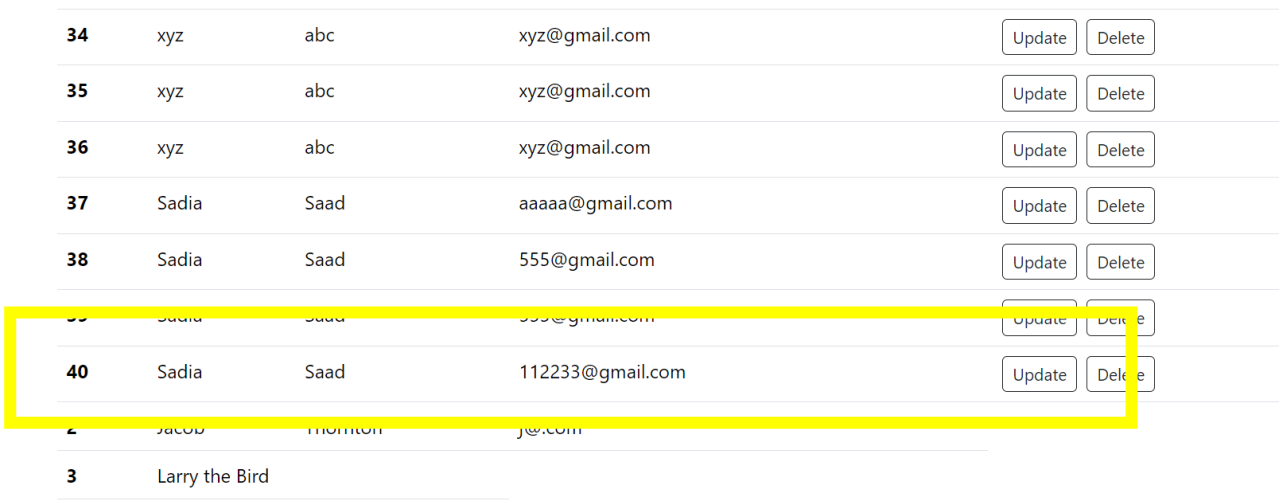
```
<h1>Update People's Info</h1>
    <form action = "/update/{{allpeople.sno}}" method="POST">
        <div class="mb-3">
            <input type="text" class="form-control" name="fname" id="exampleInputEmail1" aria-
            <div id="emailHelp" class="form-text">We'll never share your email with anyone else
```

- Make the following changes to the update function:

  Fetch the details and then change them in the database instance:

```python
@app.route('/update/<int:sno>', methods = ['GET','POST'])
def update(sno):

    if request.method=='POST':
        # Check if form fields exist
        fname = request.form.get('fname')
        lname = request.form.get('lname')
        email = request.form.get('email')
        if fname and lname and email:
                    # Create and commit new record to the database
                    allpeople = FirstApp.query.filter_by(sno=sno).first()
                    allpeople.fname=fname
                    allpeople.lname=lname
                    db.session.add(allpeople)
                    db.session.commit()

    #first fetching the record with sno
    allpeople = FirstApp.query.filter_by(sno=sno).first()

    return render_template('update.html',allpeople=allpeople)
```

- Now click on update button, make changes, the changes will be reflected in the table:



A BASIC FLASK Application with CRUD Operations in Ready!