Submitt[redacted]Subject: SSDD
Group Member : Muhammad Atif Waheed , Muhammad Hassan

PROJECT TITLE : Flask App Enhancement and Dockerization

Objective

The objective of this project is to build a secure web application using the Flask framework that allows authenticated users to manage student records using Create, Read, and Delete (CRUD) operations. The app is styled using Bootstrap 5 for a responsive interface, includes both client-side and server-side input validation, and is containerized using Docker to ensure consistent behavior across different systems. This project also demonstrates secure software practices such as session management, authentication, and code isolation.

Project Structure and File Purposes

The project follows a modular structure to separate concerns clearly and logically:

app.py is core Python file that defines the Flask application. It handles user registration, login, logout, session management, and all CRUD operations related to student data. It also includes decorators and route-level protection to ensure only logged-in users can access protected routes.

requirements.txt lists all the Python packages required to run the project. In this case, it specifies Flask==2.2.5 which Docker uses to install dependencies during image build process.

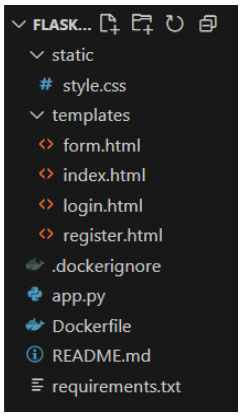
Dockerfile contains instructions to create a Docker image for the Flask application. It sets up a Python 3.8 environment, installs required packages, and runs app on port 5000 inside container.

.dockerignore excludes unnecessary files and folders from being copied into the Docker image, improving performance and avoiding security risks. It skips files like __pycache__, .env, .db, and compiled .pyc files.

templates/ is a folder containing HTML templates rendered by Flask using the Jinja2 engine. It includes:

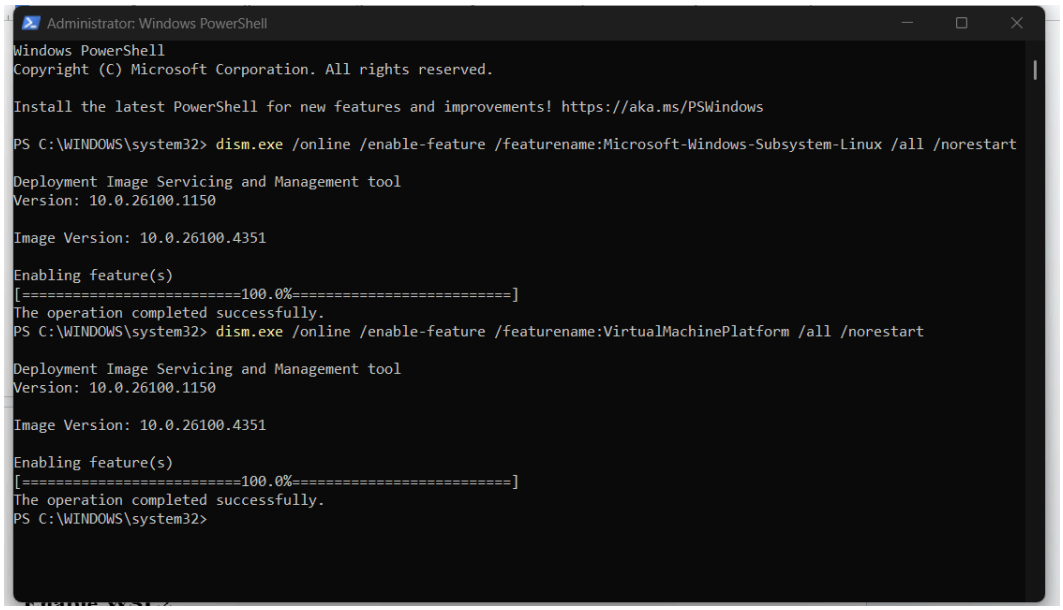
login.html for user login,|
register.html for new user registration,
index.html for viewing the student list (home page),
form.html for submitting new student entries.

static/style.css contains custom CSS styles that enhance the visual appeal of the application. It complements Bootstrap by customizing tables, buttons, background colors, and spacing.

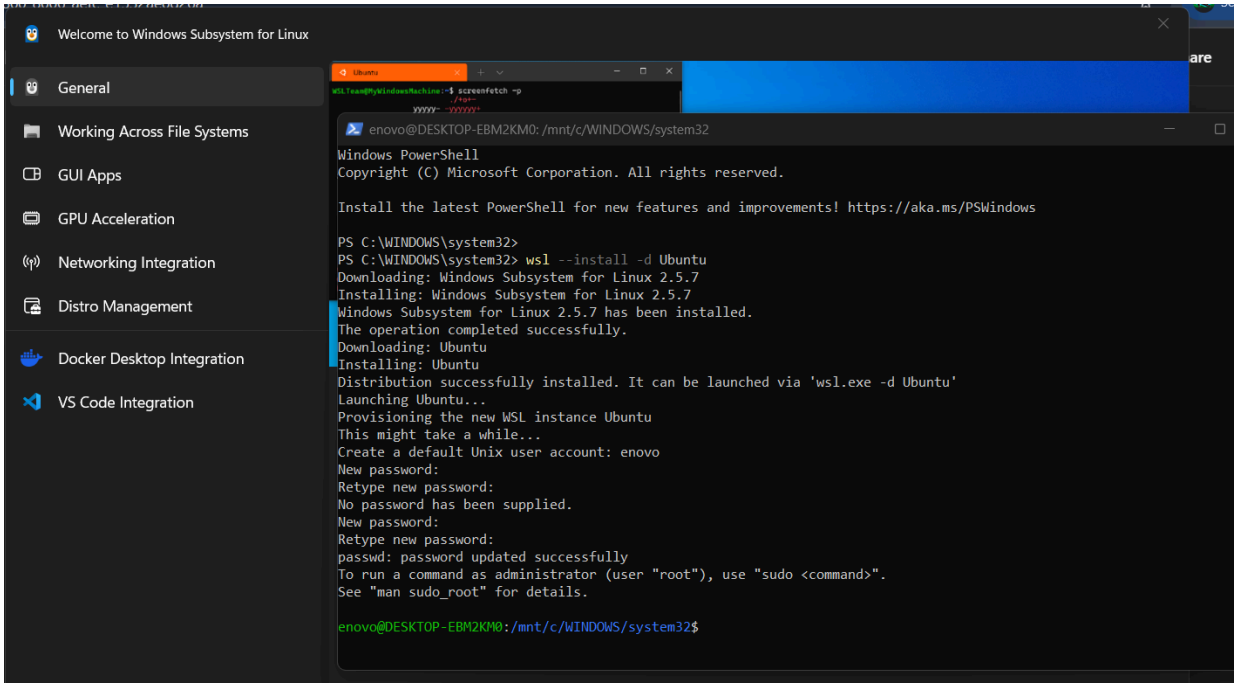


Setting up docker

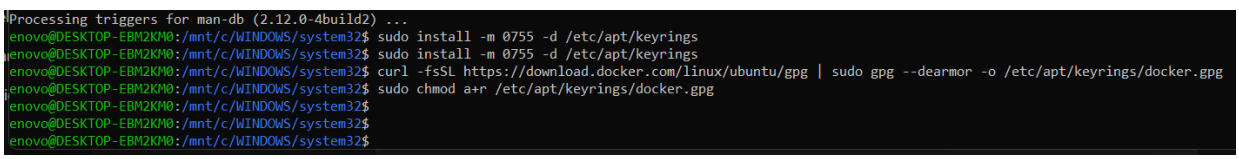
Enable WSL2

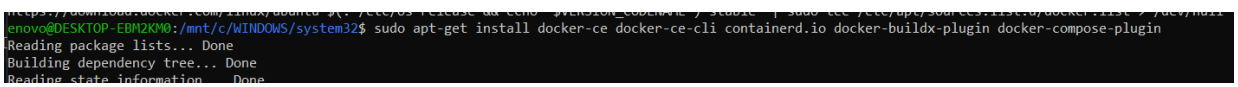


Install Ubuntu



Install Docker inside Ubuntu (WSL2)

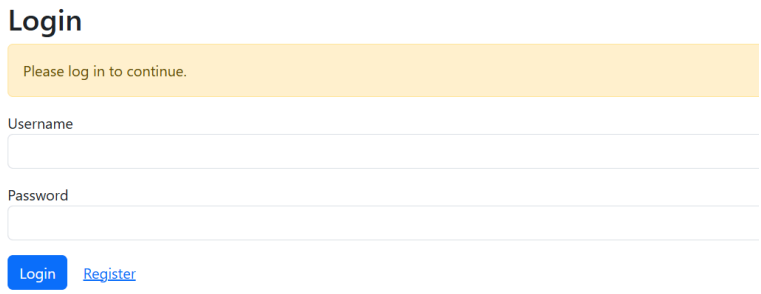


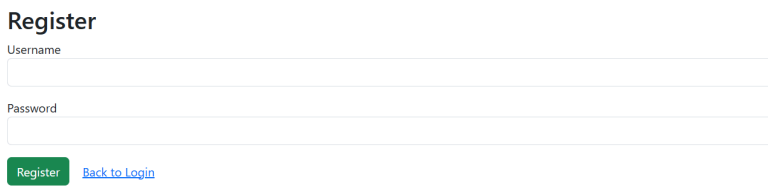


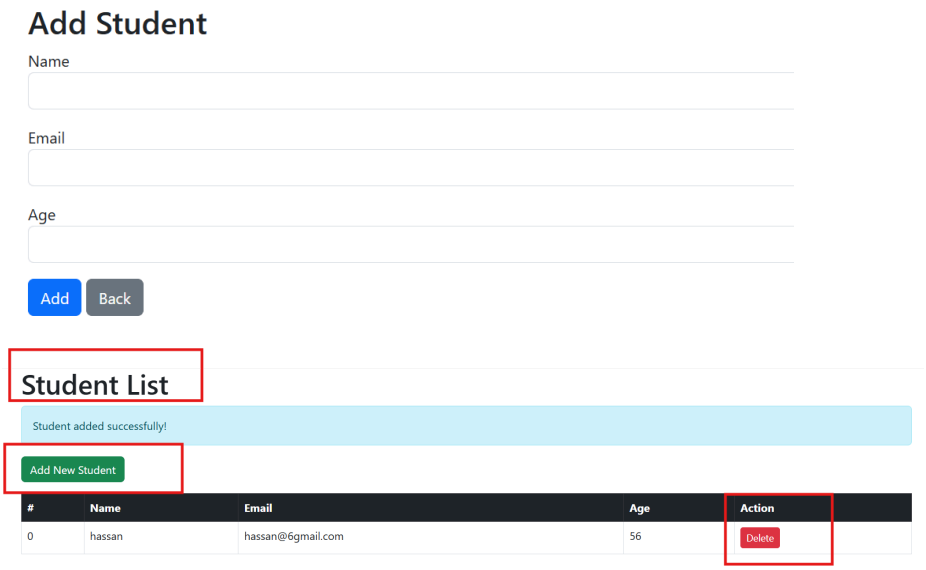
setup of the docker that how to install and complete setup is available in the [README.md]

Application Overview

The application starts at the login screen. Users must first register and then log in. Once authenticated, they are redirected to the home page, which displays the list of students. From there, users can add or delete student entries using clearly styled forms and action buttons.All user interactions display flash messages for feedback — such as successful login, validation errors, or logout notices. These are displayed in colored alert boxes using Bootstrap styles.







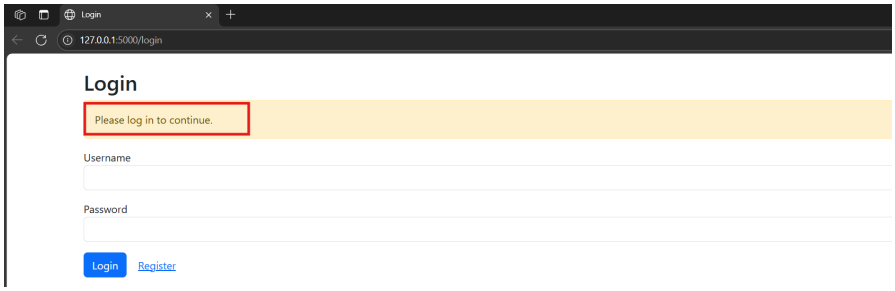
Security Features

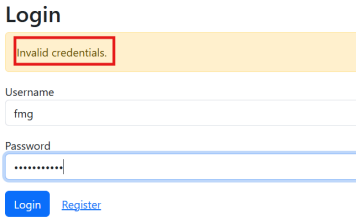
This project applies multiple layers of security to ensure secure user interaction, data validation, and session management. Each security feature plays a vital role in protecting the application from misuse, unauthorized access, and poor design practices.

1. Authentication and Access Control

A login and registration system is implemented to restrict access to sensitive functionalities. Users must register and log in before accessing student data. Once authenticated, Flask sessions store the user identity. **The @login_required decorator** is applied to the /, /add, and /delete routes, which ensures that only logged-in users can interact with student records. Unauthorized attempts are redirected to the login page.

This shows you're using the **@login_required** decorator for route protection



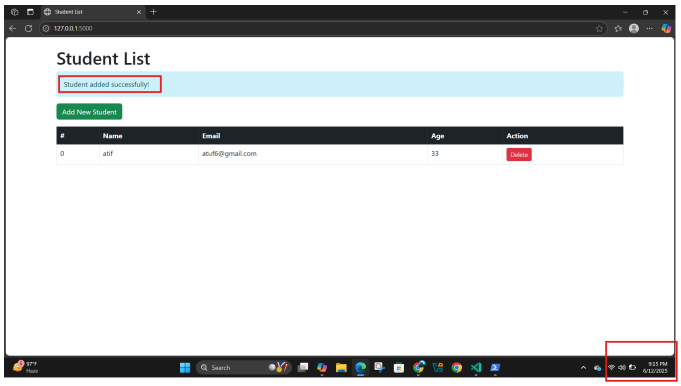


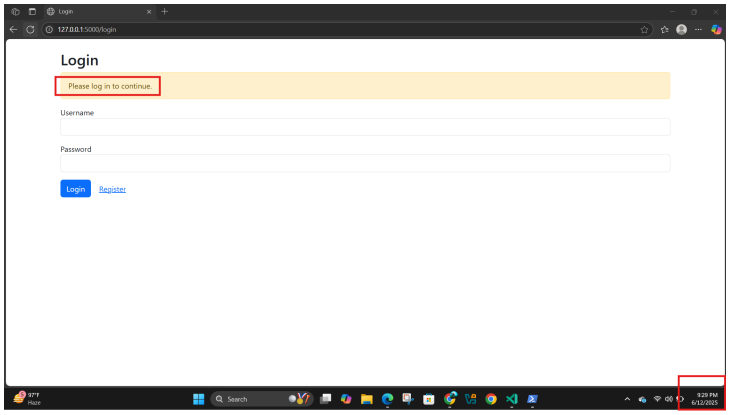

```python
@app.route('/')
@login_required
def index():
    return render_template('index.html', students=students)
```

```python
# ---------------- Auth Routes ----------------

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        if not username or not password:
            flash("Username and password are required.")
            return redirect('/register')
```

2. Session Management and Auto Logout

User sessions are stored using Flask's session mechanism. The session timeout is set to 10 minutes using **app.permanent_session_lifetime**. This ensures that inactive users are logged out automatically to prevent unauthorized use.

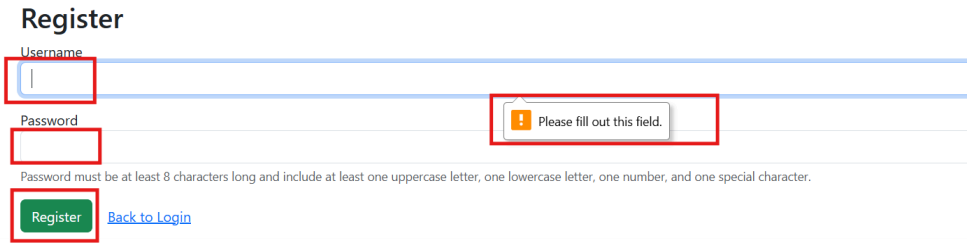




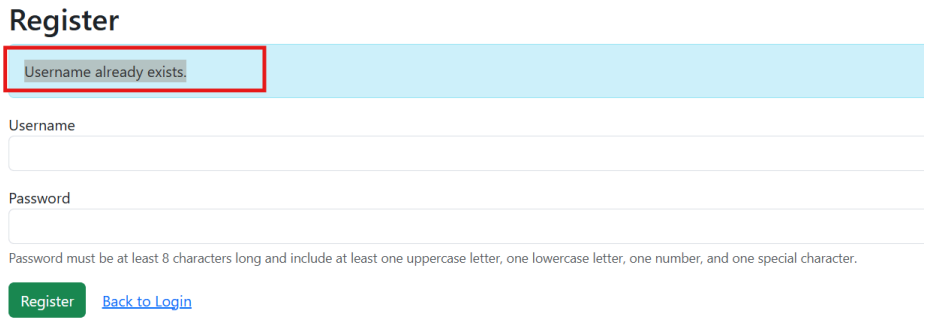

```python
#  Set session timeout to 10 minutes
app.permanent_session_lifetime = timedelta(minutes=10)
```

3. Input Validation (Client and Server Side)

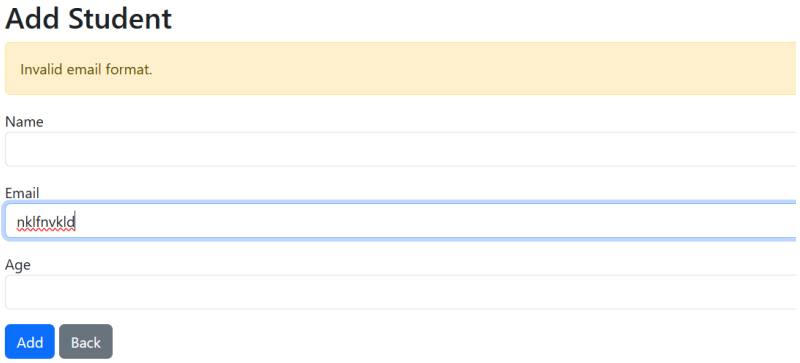
All form fields are validated both in the browser and on the server. HTML5 required, type, and pattern attributes restrict input on the **client side**, while **Python's re.match()** ensures correct formats **server-side**. This blocks invalid data and reduces attack surfaces.

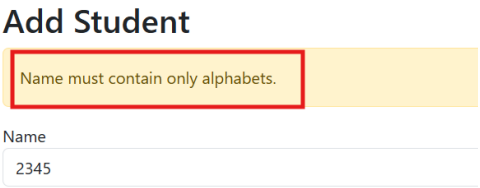



```python
if not username or not password:
    flash("Username and password are required.")
    return redirect('/register')
```




```python
if username in users:
    flash("Username already exists.")
    return redirect('/register')
```





```python
if not re.match(r'^[A-Za-z ]+$', name):
    flash('Name must contain only alphabets.')
    return redirect('/add')
if not re.match(r'^\S+@\S+\.\S+$', email):
    flash('Invalid email format.')
    return redirect('/add')
if not age.isdigit():
    flash('Age must be numeric.')
    return redirect('/add')
```

```html
<form method="POST" novalidate>
    <div class="mb-3">
        <label>Name</label>
        <input type="text" name="name" class="form-control" required pattern="[A-Za-z ]+">
    </div>
    <div class="mb-3">
        <label>Email</label>
        <input type="email" name="email" class="form-control" required>
    </div>
    <div class="mb-3">
        <label>Age</label>
        <input type="number" name="age" class="form-control" required>
    </div>
    <button type="submit" class="btn btn-primary">{{ action }}</button>
    <a href="/" class="btn btn-secondary">Back</a>
```

```html
        <input type="text" name="username" class="form-control" required>
    </div>
<div class="mb-3">
    <label>Password</label>
    <input type="password" name="password" class="form-control" required
        pattern="(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}">
    <div class="form-text">
        Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, one number, and one special cha
```

Great question, Muhammad! Let me show you the **client-side validation** happening in the HTML you shared:

→ Required

This makes sure the input **is not left empty** before the form can be submitted.
Example from your code:

<input type="text" name="name" required>

This line means: "You must type something in the Name field before submitting."

→ pattern="[A-Za-z]+"

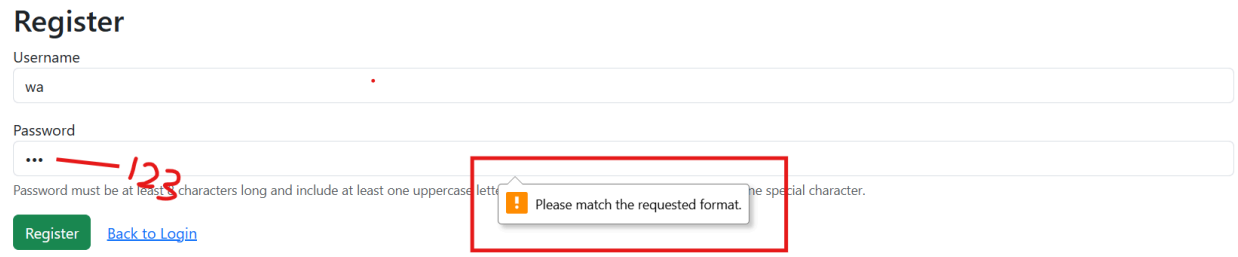
This is used in the **Name** input. It only allows **letters and spaces**—no numbers or special characters. So if someone types **1234 or @muhammad**, the browser will show an error.

→ type="email" and type="number"

These automatically check if the format is correct:
email will only accept something like example@gmail.com
number will only accept digits like 25, not twenty-five

4. Password Strength Enforcement

Strong password rules are enforced during registration. Passwords must have a minimum of 8 characters, include uppercase, lowercase, numbers, and special characters. Both client-side (pattern) and server-side (Python regex) checks are implemented.




```
pattern = r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$'
if not re.match(pattern, password):
    flash("Password must be at least 8 characters and include uppercase, lowercase, number, and special character.")
    return redirect('/register')
```

```
<label>Password</label>
<input type="password" name="password" class="form-control" required
       pattern="(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}">
```

5. Docker-Based Isolation

The app runs inside a Docker container built from a Dockerfile. This isolates the environment, limits access to the host, and ensures consistent behavior. Docker also simplifies sharing the app and its dependencies

First go to directory

```
enovo@DESKTOP-EBM2KM0:/mnt/c/WINDOWS/system32$ cd /mnt/c/Users/Lenovo/Desktop/flask-students-app
```

Build docker container

```
enovo@DESKTOP-EBM2KM0:/mnt/c/Users/Lenovo/Desktop/flask-students-app$ sudo docker build -t flask-students-app .
[+] Building 37.0s (4/8)                                                                    docker:default
[+] Building 37.7s (4/8)                                                                    docker:default
[+] Building 75.7s (9/9) FINISHED                                                           docker:default
 => [internal] load build definition from Dockerfile                                                0.1s
 => => transferring dockerfile: 164B                                                                0.1s
```

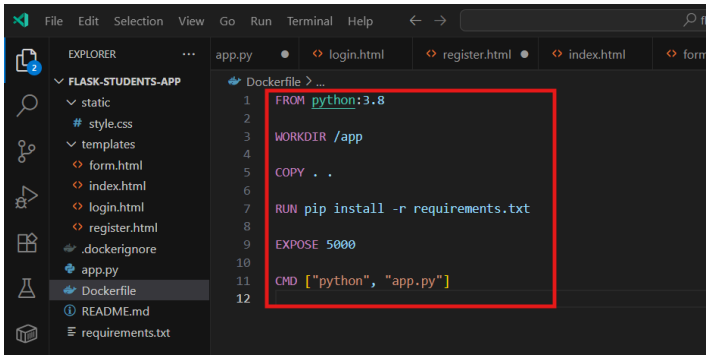
Run this

```
enovo@DESKTOP-EBM2KM0:/mnt/c/Users/Lenovo/Desktop/flask-students-app$ sudo docker run -p 5000:5000 flask-students-app
 * 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 all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.18.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 103-349-726
```

Check current containers

```
enovo@DESKTOP-EBM2KM0:/mnt/c/Users/Lenovo/Desktop/flask-students-app$ sudo docker ps -a
[sudo] password for enovo:
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS                     PORTS     NAMES
7f686719d5a5   flask-students-app "python app.py"          12 hours ago   Exited (0) 11 hours ago              elated_allen
d01e86d8988e   flask-students-app "python app.py"          12 hours ago   Exited (0) 12 hours ago              practical_williams
42f1ecbab3a3   2aa3ac892e11       "python app.py"          13 hours ago   Exited (0) 13 hours ago              mystifying_hertz
091f182dd8b7   2aa3ac892e11       "python app.py"          16 hours ago   Exited (0) 16 hours ago              agitated_clarke
d61351b89cca   busybox            "echo 'Hello from Bu…"   18 hours ago   Exited (0) 18 hours ago              romantic_swirles
e1bbde0a0b9d   hello-world        "/hello"                 18 hours ago   Exited (0) 18 hours ago              sweet_gould
enovo@DESKTOP-EBM2KM0:/mnt/c/Users/Lenovo/Desktop/flask-students-app$
```

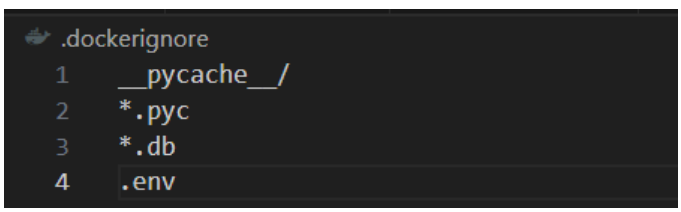
Docker file



The Dockerfile is a **script of instructions** that tells Docker **how to build a container image** for your application. It's like a **recipe** for packaging your app with everything it needs to run — including Python, Flask, and your source code.

.dockerignore

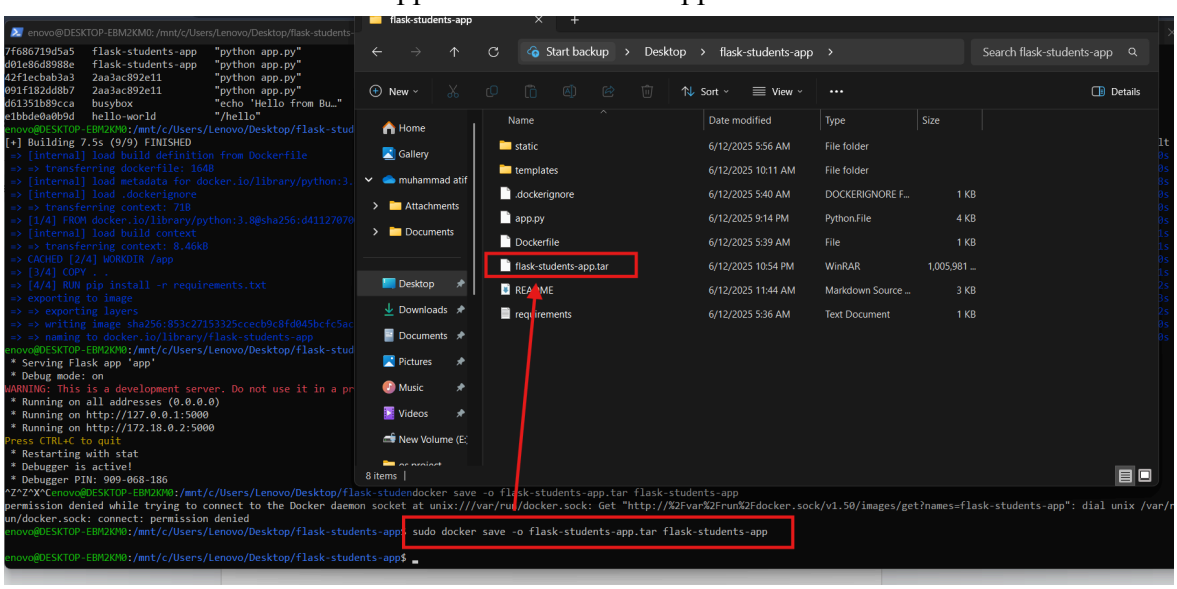
.dockerignore is a text file that tells Docker which files and folders to exclude when building the Docker image using docker build.



How to send the docker image to anyone and how will he runs !!

Save the Docker image to a .tar file

docker save -o flask-students-app.tar flask-students-app



Send the .tar file using anything like usb, drive etcc
Once they receive the flask-students-app.tar file
Open wsl ubuntu(docker setup)

```
C:\Windows\System32>wsl -d ubuntu
enovo@DESKTOP-EBM2KM0:/mnt/c/Windows/System32$
```

Open cmd as adminstrator and run this command or just search (ubuntu or wsl) then run
Load the Docker image
docker load -i flask-students-app.tar
Run the Docker container
docker run -p 5000:5000 flask-students-app

—----------end—----------