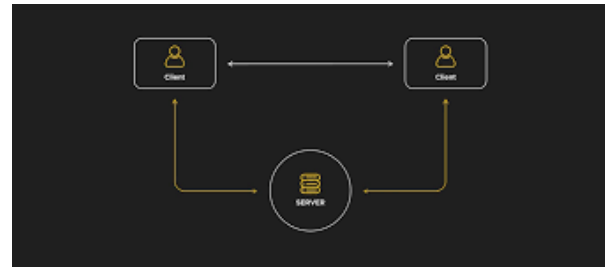


IDENTIFYING VULNERABILITIES



What is our GOAL for this CLASS?

In this class, students went through and understood the code and also tried finding out vulnerabilities in the code.

What did we ACHIEVE in the class TODAY?

- Understanding the e commerce website code
- Understanding how MVC architecture and servers work

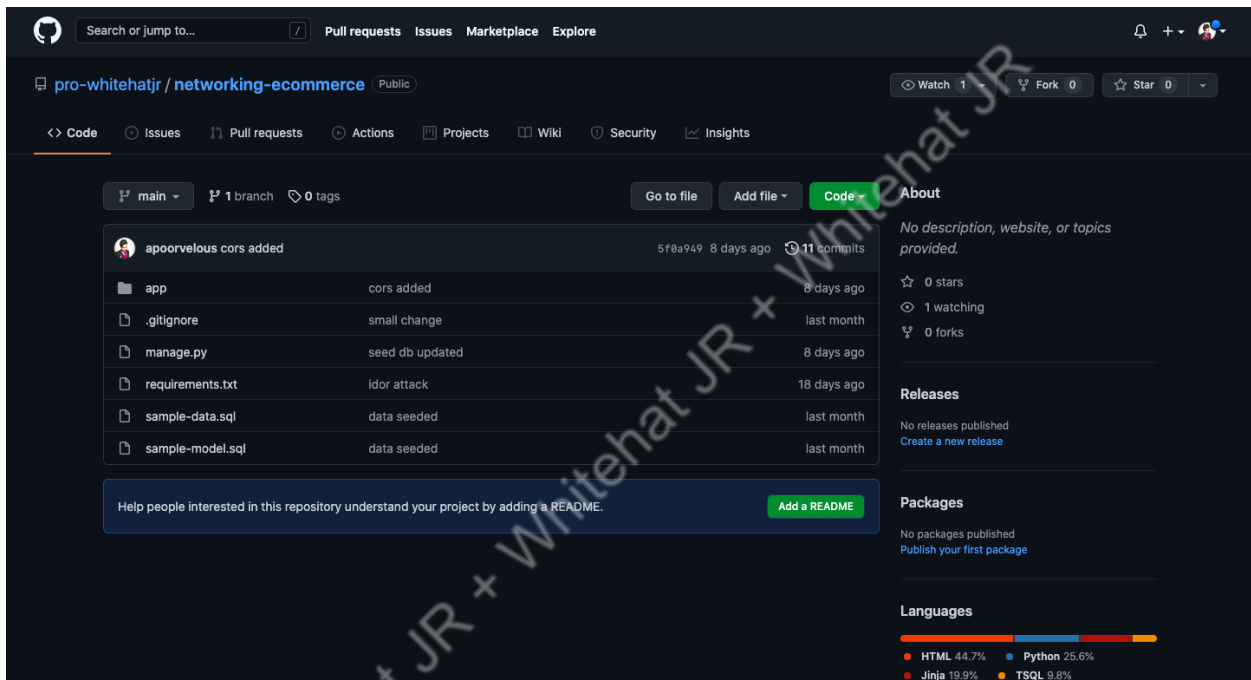
Which CONCEPTS/ CODING BLOCKS did we cover today?

- Flask
- Python
- SQLAlchemy
- Software Development

How did we DO the activities?

Activity:

1. Refer to the [following](#) repository to see the code of the ecommerce website.



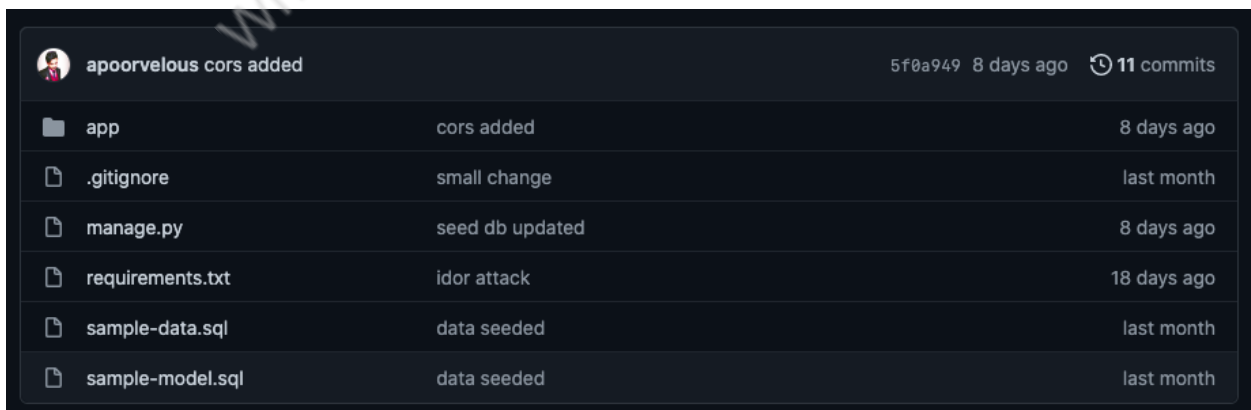
The screenshot shows the GitHub interface for the repository 'pro-whitehatjr/networking-commerce'. The repository is public and has 11 commits, 0 stars, 1 watcher, and 0 forks. The commit history is displayed in a table:

Commit	Message	Time
5f0a949	cors added	8 days ago
	small change	last month
	seed db updated	8 days ago
	idor attack	18 days ago
	data seeded	last month
	data seeded	last month

The repository also includes a README section with a button to 'Add a README'. The 'About' section indicates no description, website, or topics are provided. The 'Releases' section shows no releases published. The 'Packages' section shows no packages published. The 'Languages' section shows a bar chart with the following data:

Language	Percentage
HTML	44.7%
Python	25.6%
Jinja	19.9%
TSQL	9.8%

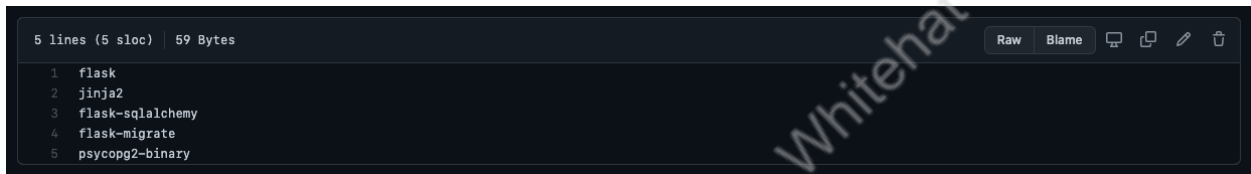
2. Before deep diving into the code, we look at the files and folders in the project -



The screenshot shows a detailed view of the commit history table from the repository. The table lists the following files and folders:

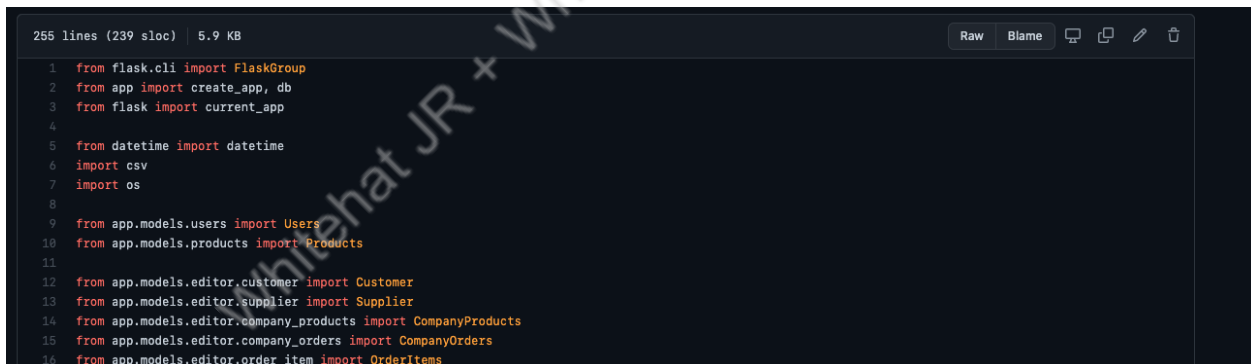
File/Folder	Message	Time
app	cors added	8 days ago
.gitignore	small change	last month
manage.py	seed db updated	8 days ago
requirements.txt	idor attack	18 days ago
sample-data.sql	data seeded	last month
sample-model.sql	data seeded	last month

3. Notice the following files and folders in the project -
 - a. **App** folder
 - b. **.gitignore** file
 - c. **Manage.py** file
 - d. **Requirements.txt** file
 - e. **Sql** files
4. In our files, the SQL files are irrelevant as they are just used to insert dummy data into the database.
5. .gitignore file is used to ignore certain files and folders to be uploaded to github.
6. Requirements.txt file is used to have the name of the modules that the project uses, so they can all be installed at once using the following command -
 - a. **pip install -r requirements.txt**
 - b. The contents of the file looks like this -



```
5 lines (5 sloc) | 59 Bytes
1 flask
2 Jinja2
3 flask-sqlalchemy
4 flask-migrate
5 psycopg2-binary
```

7. **Manage.py** file is the most important file of the project. It manages how the project will run.
8. At the top, we import some of the libraries and modules for the project where you will notice **app.models.*** syntax too. They are used to import the models of the database



```
255 lines (239 sloc) | 5.9 KB
1 from flask.cli import FlaskGroup
2 from app import create_app, db
3 from flask import current_app
4
5 from datetime import datetime
6 import csv
7 import os
8
9 from app.models.users import Users
10 from app.models.products import Products
11
12 from app.models.editor.customer import Customer
13 from app.models.editor.supplier import Supplier
14 from app.models.editor.company_products import CompanyProducts
15 from app.models.editor.company_orders import CompanyOrders
16 from app.models.editor.order_item import OrderItems
```

9. Next, we have some dictionaries that contain the seeded data of the database -

```
user_json = [  
    {  
        "name": "John Doe",  
        "email": "john.doe@gmail.com",  
        "password": "hello_john",  
        "contact": "+1 (1234) 123 123"  
    },  
    {  
        "name": "James Bond",  
        "email": "james.bond@gmail.com",  
        "password": "bond007",  
        "contact": "+37 (1234) 567 890"  
    },  
    {  
        "name": "Amy Brown",  
        "email": "amy.brown@gmail.com",  
        "password": "brownamy3",  
        "contact": "+83 (4456) 285 847"  
    },  
    {  
        "name": "John Wick",  
        "email": "john.wick@gmail.com",  
        "password": "johndog_1",  
        "contact": "+49 (8712) 419 063"  
    },  
    {  
        "name": "Helene Sebi",  
        "email": "helene.sebi@gmail.com",  
        "password": "helenelove1",  
        "contact": "+33 (9331) 740 903"  
    },  
]
```

10. We have the following functions -

- a. **FlaskGroup()** that can be used to create commands for managing this application -

```
cli = FlaskGroup(create_app=create_app)
```

- i.
- b. We have functions **recreate_db()** and **seeder()** that resets the DB where the **recreate_db()** functions deleted and then recreated the db and **seeder()** function re-enters the dummy data into the newly created database -

```
def recreate_db():
    db.drop_all()
    db.create_all()
    db.session.commit()

def seeder():
    for user in user_json:
        Users.create(user.get("name"), user.get("email"), user.get("password"), user.get("contact"))

    for product in product_json:
        Products.create(product.get("name"), product.get("image"), product.get("rating"), product.get("marked_price"), product.get("selling_price"))

    #Seeding the editor
    with open("app/editor_data/customer.csv", "r") as f:
        csvreader = csv.reader(f)
        for row in csvreader:
            try:
                Customer.create(int(row[0]), row[1], row[2], row[3], row[4], row[5])
            except:
                pass
```

11. The **recreate_db()** and **seeder()** are then used in a function called **rsd()** which also has a decorator **@cli.command** which means that this function can be used from the command line / terminal directly with the following command -

a. **python manage.py rsd**

```
@cli.command()
def rsd():
    # if current_app.config.get('ENV') not in ('development', 'test', 'testing'):
    #     print("ERROR: seed-db only allowed in development and testing env.")
    #     return
    recreate_db()
    seeder()

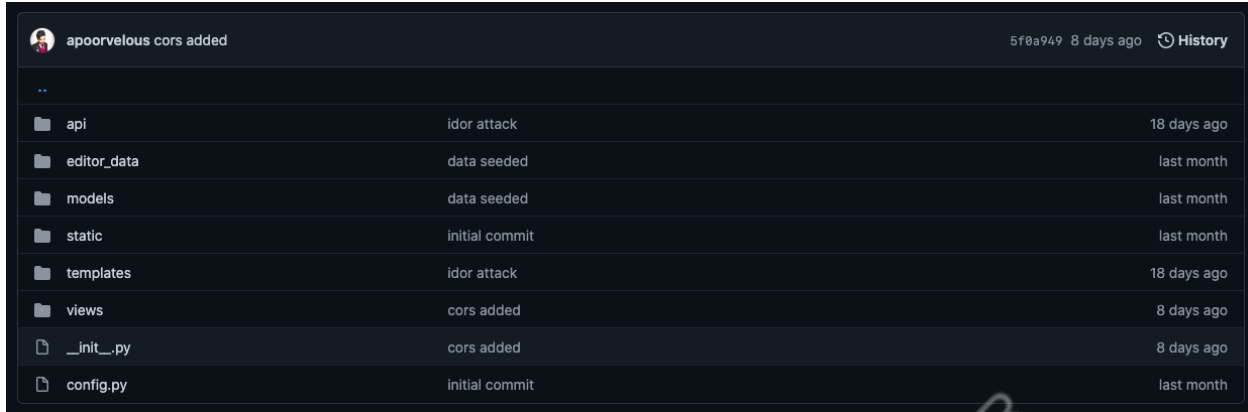
if __name__ == '__main__':
    cli()
```

12. Similarly, we have another command to run the application called -

a. **python manage.py run**

13. These commands are created with the **FlaskGroup()** function from earlier.

14. In the **app** folder, we have the following -



..		
api	idor attack	18 days ago
editor_data	data seeded	last month
models	data seeded	last month
static	initial commit	last month
templates	idor attack	18 days ago
views	cors added	8 days ago
__init__.py	cors added	8 days ago
config.py	initial commit	last month

15. The **config.py** file is used to set some configuration for our project. It is mostly just defining different configurations for different environments like **development**, **stage** or **production** and also to define the URL of the database or which database, URLs or Ports to use.
16. **__init__.py** file is the most important file of the project. It initialises the **app** folder so that it can work as a Python module.
17. At first, there are just some modules imported and some functions initialised -

```
import os
from flask import Flask, jsonify
from flask_cors import CORS
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

# instantiate the extensions
db = SQLAlchemy()
migrate = Migrate()
```

18. We are importing the **CORS** to allow cross origin requests in an application.
19. **SQLAlchemy** is used to write and execute SQL queries in Flask
20. **Migrate** is used to migrate data into the database when some change happens. It is understood that SQL is very particular about the columns and their types in a table, and if there is a new column added, or the type of a column is changed, then migrate is used to transfer the data from the older database to the newly constructed database without losing any data.

21. We then initialise our application -

```
def create_app(script_info=None):
    # instantiate the app
    app = Flask(__name__)
    cors = CORS(app)

    # set configz
    app_settings = os.getenv('APP_SETTINGS')
    app.config.from_object(app_settings)
    app.config['CORS_HEADERS'] = 'Content-Type'

    # set up extensions
    db.init_app(app)
    migrate.init_app(app, db)

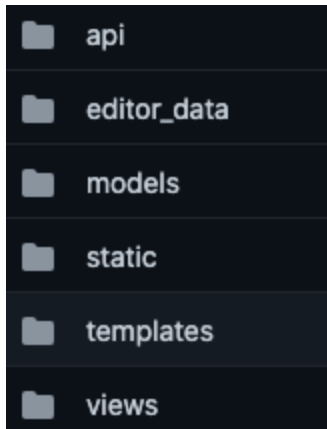
    # register blueprints
    from .views.views import views
    from .api.api import api

    app.register_blueprint(views)
    app.register_blueprint(api)
```

22. The code works in the following way -

- a. **create_app()** function is used to create the flask application and also the CORS is allowed in the application
- b. The **config.py** file is used to then import different configurations required to run the application.
- c. The database is initialised with the **db.init_app()** function
- d. We are registering our blueprints with the **register_blueprint()** function.
 - i. Blueprints are used to set different types of blueprints differently.
 - ii. In e-commerce application, there are different routes for frontend and backend. **Views** is used to set routes for frontend and **api** is used to set routes for backend.

23. Take a look at the folders in the **app** folder -



24. The **api** and the **views** folder contain code for routes for **APIs** and **Views** just similar to how you have learnt before in FLASK.
25. The **static** folder contains all the static images and files for the application, including the user uploaded **attachments**.
26. The **templates** folder contains the HTML, CSS and JavaScript files for the frontend pages of the application.
27. The **models** folder contains python files for all the tables, that contains the column names and their types, through which, the schema of each and every table in our database is defined.
28. Explore the code and based on all the vulnerabilities you have exploited before, find out where that piece of code is and how it could have been written differently to avoid getting hacked.

What's NEXT?

In the next class, we will review a secure version of the ecommerce app code and discuss how different vulnerabilities were addressed.

Expand Your Knowledge:

Explore more Python decorators [here](#)