UniSearch

University Browsing Platform

IT325 Web Services Project

Hamza Boukhatem

Senior BA/IT

Tunis Business School

January 2024

Abstract

The process of selecting a choice for higher education can be a very arduous task for high school graduates who have just finished with the stress of national examinations. What is more aggravating is that this search process uses archaic methods like long hard documents with the expectations of students taking their time to go through the overwhelming amount of information.

This is where UniSearch our Flask powered web application comes into play.With a vast array of filters and information gathered about numerous universities such as their description, the tuition, the degree and the speciality as well as tools to measure the students score , this API will allow students to have an easier more comfortable time making the best decision about the future prospects.

Keywords - API , Flask app , universities , UniSearch , students and score.

Contents

Chapter 1

Introduction

My final Project for the IT325 Web Services course this semester consists of a RESTful API developed using Flask ,Python and Visual Studio Code for web development.

A number of databases precisely four were designed and created through the implementation of Flask-SQLAlchemy and SQLite.

I have also used technologies such as Insomnia and Flasgger/swagger documentation as a tool to control the functionality of the the project and assuring its quality.

The nature of the project that i chose allowed me the freedom to implement dozens of CRUD operations that have been secured through user authentication methods and password hashing. These functions can be tested and using Insomnia endpoints tests and also through four web pages.

The inspiration behind this project is to optimize the new high school graduates university selection process by facilitating the discovery and research of higher education Establishments around the country .

Its main function is returning a filtered list of university based on a primary criteria like the state the student wishes to pursue his higher education in and other secondary ones like the type of degree or specialty to pursue or the type of university if it is private or public.

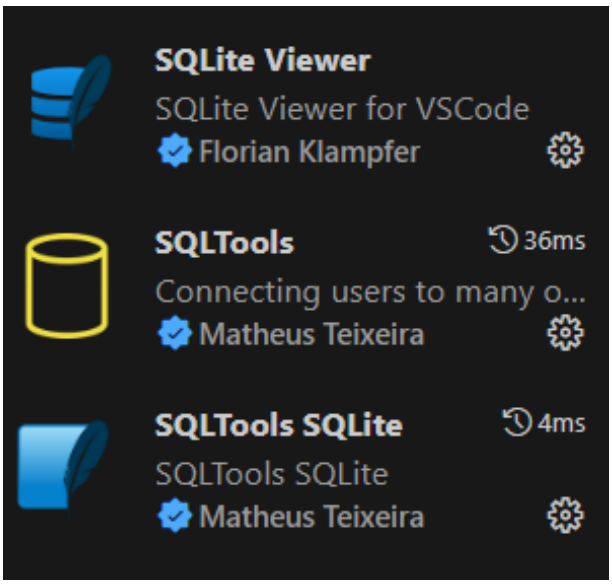
Chapter 2

Work Process and Technology Showcase

2.1 VSCode Contribution:

One of the most popular source-code editors, Visual Studio Code, commonly called VS Code, is very beginner-friendly.[9].Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add functionality.

I used VSCode mainly for its compatibility with different file types but mainly the extensions especially **SQLite tools/Viewer extension** .



2.2 Flask Contribution:

Flask is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.[1]

The Flask library was my main tool of choice to design my API. I used flask when coding my CRUD methods by implementing decorators and assigning specific routes to them , i used imports such as jasonify to convert Python dictionaries into JSON objects. I also implemented many flask extensions.

2.3 Python Contribution:

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

To design my Flask API, i decided on Python , a programming language that i am familiar with . I used python to develop the body of the methods i constructed ; other examples where i used python was when handling output response messages and error handling or when some methods included conditional functions.

2.4 SQLAchemy Contribution:

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.[7]

I used SQLAlchemy when designing my database models i have use it in the design of four databases models which are "state","university","user" and "subject" and the corresponding relationships between them through primary and foreign keys. There was an instance where i even defined inside "Subject" db a function that will update a parameter of "user" db .

Below is an example of a db model I created with SQLAlchemy import from flask-sqlachemy extension.

```python
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin

db = SQLAlchemy()

class State(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False)
    university_number = db.Column(db.Integer, default=0)
    universities = db.relationship('University', backref='state', lazy=True)

    def __repr__(self):
        return f"State(id={self.id}, name={self.name})"
```

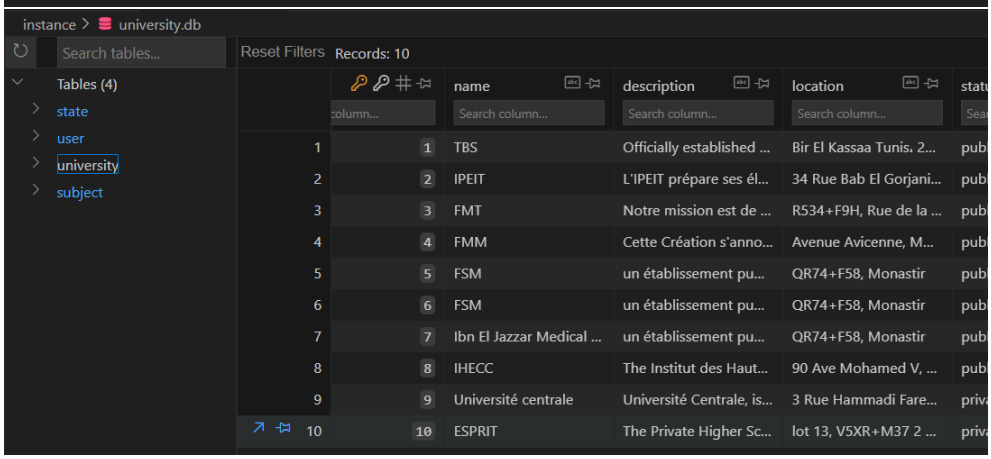
2.5 SQLite and SQLite Viewer Contribution:

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.[5]

After settling for my database models i chose to create them using SQLite which was the most flexible and accessible choice for me and further more its VSCode extension SQLite viewer helped me keep track of the db objects getting filled as intended .

Down below is the code used for initiating the creation of the db with "OS" import to pinpoint the creation path and also the interface of SQLite Viewer which highlights the tables columns .

```python
from flask import Flask, request, jsonify
import os

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///university.db'
app.config['SECRET_KEY'] = '0123456789'
app.instance_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'instance')

db.init_app(app)

with app.app_context():
    db.create_all()
```



2.6 Flask-login Contribution:

Flask-Login provides user session management for Flask. It handles the common tasks of logging in, logging out, and remembering your users' sessions over extended periods of time.[4]

I have used flask-login while developing users functionalities such as registration , logging in logging out and differentiation between normal users (students) and admin users . It also allowed me to create decorators

"@login_required" and "@admin_required" placed just before methods' body to achieve different levels of authorization for the two user roles.

Below are the methods to creating an admin exclusive decorator giving the right conditions and the other one is for managing the users and loading them.

```python
from flask_login import LoginManager, login_user, login_required, logout_user, current_user

login_manager = LoginManager(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

```python
def admin_required(fn):
    @wraps(fn)
    def wrapper(*args, **kwargs):
        if current_user.is_admin:
            return fn(*args, **kwargs)
        else:
            return jsonify({'error': 'Permission denied. Admin access required.'}), 403
    return wrapper
```

2.7 Fask-Bcrypt Contribution:

Flask-Bcrypt is a Flask extension that provides bcrypt hashing utilities for your application. Bcrypt is one of many advisable choice to better secure sensitive data that must be protected such as passwords.[2]

To add more security features to the user registration method, I resorted to the use of Bcrypt , another flask extension , that allowed me to add hashing to my passwords when a user is created. It was easy to add in my already developed app's first versions and a good addition to the user management methods and the way it works is :

from flask import Flask

from flask-bcrypt import Bcrypt

app = Flask(__name__)

bcrypt = Bcrypt(app)

`"Inside the method:"`

pw_hash = bcrypt.generate_password_hash('password')

bcrypt.check_password_hash(pw_hash, 'password')

2.8 The Implemented HTTP Methods:

2.8.1 GET Methods:

GET /users/`<string:username>`/NEgrade

This method was used to show the student type user his National Examination Grade (NEgrade) and the purpose of it will be highlighted later when discussion the web pages created .

```python
# Get NEgrade for a user by username
@app.route('/users/<string:username>/NEgrade', methods=['GET'])
#@login_required
def get_NEgrade(username):
    user = User.query.filter_by(username=username).first()

    if not user:
        return jsonify({'error': 'User not found.'}), 404

    return jsonify({'NEgrade': user.NEgrade}), 200
```

GET /states

This method will return the list of states recorded each having a name , and id , a number of universities within and a list of universities showing some of their credentials.

```python
# Get all states
@app.route('/states', methods=['GET'])
@login_required
def get_all_states():
    states = State.query.all()
    states_list = [{'id': state.id,
                    'name': state.name,
                    'university_number': state.university_number,
                    'universities': [{'id': uni.id,
                                      'name': uni.name,
                                      'status': uni.status,
                                      'specialty': uni.specialty} for uni in state.universities]}
                   for state in states]
    return jsonify(states_list)
```

GET /states/<string:state_name>/universities

This method allows the student to browse the universities available in each state. You can also see in the code the addition of a filter function that will come into play later when discussion the web pages development.

```python
# Get all universities in a given state.
@app.route('/states/<string:state_name>/universities', methods=['GET'])
#@login_required
def get_universities_in_state(state_name):
    state = State.query.filter_by(name=state_name).first()
    if not state:
        return jsonify({'error': 'State not found'}), 404

    universities_list = [{'id': uni.id,
                          'name': uni.name,
                          'status': uni.status,
                          'description': uni.description,
                          'location': uni.location,
                          'tuition_fee': uni.tuition_fee,
                          'specialty': uni.specialty,
                          'degree': uni.degree,
                          'student_capacity': uni.student_capacity,
                          'last_year_score': uni.last_year_score}
                         for uni in state.universities]
    # Get filter parameters from query string
    status_filter = request.args.get('status')
    specialty_filter = request.args.get('specialty')
    degree_filter = request.args.get('degree')

    # Apply filters if provided
    filtered_universities = [uni for uni in universities_list
                             if (not status_filter or uni['status'] == status_filter) and
                                (not specialty_filter or uni['specialty'] == specialty_filter) and
                                (not degree_filter or uni['degree'] == degree_filter)]

    return jsonify(filtered_universities)
```

GET /states/<string:state_name>/universities/(filers) :

1. **GET /states/<string:state_name>/universities/specialty/<string:specialty>**
2. **GET /states/<string:state_name>/universities/degree/<string:degree>**
3. **GET /states/<string:state_name>/universities/status/<string:status>**

These 3 methods are filters ; they allow students after choosing their desired state to filter the results either by degree type , by specialty or by university status.

I will provide one of the bodies of these 3 methods to showcase because they follow the same logic.

```python
# Get universities in a state by specialty
@app.route('/states/<string:state_name>/universities/specialty/<string:specialty>', methods=['GET'])
@login_required
def get_universities_in_state_by_speciality(state_name, specialty):
    state = State.query.filter_by(name=state_name).first()
    if not state:
        return jsonify({'error': 'State not found'}), 404

    universities = University.query.filter_by(state=state, specialty=specialty).all()
    universities_list = [{'id': uni.id,
                          'name': uni.name,
                          'status': uni.status,
                          'description': uni.description,
                          'location': uni.location,
                          'tuition_fee': uni.tuition_fee,
                          'degree': uni.degree,
                          'student_capacity': uni.student_capacity,
                          'last_year_score': uni.last_year_score}
                         for uni in universities]

    return jsonify(universities_list)
```

GET /universities/(filter)/(value)

1. **GET /universities/name/<string:uni_name>**
2. **GET /universities/degree/<string:degree>**
3. **GET /universities/status/<string:status>**
4. **GET /universities/specialty/<string:specialty>**

This is a second batch of filters this time they are not dependent on states and include all universities inside the university database and these methods filter them either by name, degree, specialty or status.

Like i did before i will show one of these for filers code to get a reference:

```python
# Get a university by name
@app.route('/universities/name/<string:uni_name>', methods=['GET'])
@login_required
def get_university_by_name(uni_name):
    university = University.query.filter_by(name=uni_name).first()
    if not university:
        return jsonify({'error': 'University not found'}), 404

    university_data = {
        'id': university.id,
        'name': university.name,
        'state': university.state.name,
        'status': university.status,
        'description': university.description,
        'location': university.location,
        'tuition_fee': university.tuition_fee,
        'specialty': university.specialty,
        'degree': university.degree,
        'student_capacity': university.student_capacity,
        'last_year_score': university.last_year_score
    }

    return jsonify(university_data)
```

2.8.2 POST Methods:

POST /register

This method register a user in the database; it takes the username, his email, his password, sector of his studies, NEgrade if he knows it if not it will be computed in another post method and is_admin a boolean

type variable that has default value of false meaning student not admin and is hidden.

```python
# register a user
@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()

    if not data or 'username' not in data or 'password' not in data:
        return jsonify({'error': 'Invalid request. Please provide username and password.'}), 400

    username = data['username']
    password = data['password']
    is_admin = data.get('is_admin', False)
    email = data.get('email', '')  # Default value is an empty string
    sector = data.get('sector', '')  # Default value is an empty string
    NEgrade = data.get('NEgrade', 0.0)  # Default value is 0.0

    if len(password) < 6:
        return jsonify({'error': 'Password must be at least 6 characters long.'}), 400

    existing_user = User.query.filter_by(username=username).first()
    if existing_user:
        return jsonify({'error': 'Registration failed. Please choose a different username.'}), 400

    hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')

    new_user = User(
        username=username,
        password=hashed_password,
        email=email,
        sector=sector,
        NEgrade=NEgrade,
        is_admin=is_admin
        )
    db.session.add(new_user)
    db.session.commit()

    return jsonify({'message': 'Registration successful'}), 201
```

POST /login

This method is implement at the beginning of testing process because almost every method has the @login_required decorator in it so this method allows the user to access and use the rest of the methods and it takes the username and password as inputs.

```python
#login
@app.route('/login', methods=['POST'])
def login():
    if current_user.is_authenticated:
        return jsonify({'message': 'User is already logged in'})

    data = request.json
    username = data.get('username')
    password = data.get('password')

    user = User.query.filter_by(username=username).first()

    if user and bcrypt.check_password_hash(user.password, password):
        login_user(user)
        return jsonify({'message': 'Login successful'})
    else:
        return jsonify({'error': 'Invalid credentials'}), 401
```

POST /logout

This method simply logs out the users after they have finished using the service and it can not be used until their is a login session working.

```python
#logout
@app.route('/logout', methods=['POST'])
@login_required
def logout():
    if not current_user.is_authenticated:
        return jsonify({'error': 'Unauthorized - User not logged in or already logged out'}), 401

    logout_user()
    return jsonify({'message': 'Logout successful'}), 200
```

POST /users/<string:username>/subjects

This method is only accessible after logging in and it is used by students to submit their exam results. this method also computes the NEgrade for users and updates it .

```python
# add subjects to user by name
@app.route('/users/<string:username>/subjects', methods=['POST'])
#@login_required
def add_subject(username):
    data = request.get_json()

    user = User.query.filter_by(username=username).first()

    if not user:
        return jsonify({'error': 'User not found.'}), 404

    new_subject = Subject(
        user_id=user.id,
        math=data.get('math', 0.0),
        science=data.get('science', 0.0),
        physics=data.get('physics', 0.0),
        french=data.get('french', 0.0),
        english=data.get('english', 0.0),
        arabic=data.get('arabic', 0.0),
        philosophy=data.get('philosophy', 0.0),
        computer_science=data.get('computer_science', 0.0),
        hist_geo=data.get('hist_geo', 0.0),
        economy=data.get('economy', 0.0),
        gestion=data.get('gestion', 0.0),
        technology=data.get('technology', 0.0),
        sport=data.get('sport', 0.0)
    )

    try:
        user.NEgrade = round(new_subject.calculate_NEgrade(user.sector), 2)
        db.session.add(new_subject)
        db.session.commit()
        return jsonify({'message': 'Subject added successfully'}), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({'error': str(e)}), 500
```

POST /states

This is an admin only method having the decorator @admin_required and it is used to add states.

```python
# Create a state
@app.route('/states', methods=['POST'])
@login_required
@admin_required
def create_state():
    data = request.json
    state_name = data.get('name')
    existing_state = State.query.filter_by(name=state_name).first()
    if existing_state:
        return jsonify({'error': 'State with the same name already exists'}), 400
    data = request.json
    new_state = State(name=data['name'])
    db.session.add(new_state)
    db.session.commit()
    return jsonify({'message': 'State added successfully', 'id': new_state.id}), 201
```

POST /states/<string:state_name>/universities

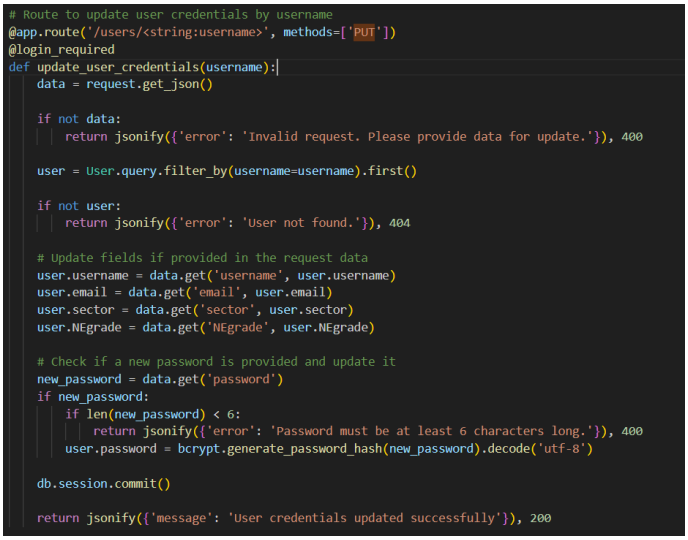
This method is also for admin only use having the decorator @admin_required and it allows the admin to add universities to his database.

```python
    if existing_university:
        return jsonify({'error': 'University with the same parameters already exists in the state'}), 400

    new_university = University(name=data['name'],
                                description=data.get('description'),
                                location=data.get('location'),
                                status=data.get('status'),
                                tuition_fee=data.get('tuition_fee'),
                                specialty=data.get('specialty'),
                                degree=data.get('degree'),
                                student_capacity=data.get('student_capacity'),
                                last_year_score=data.get('last_year_score'),
                                state=state)

    state.universities.append(new_university)
    state.university_number += 1

    db.session.add(new_university)
    db.session.commit()

    return jsonify({'message': 'University created successfully', 'id': new_university.id}), 201
```

2.8.3 PUT Methods:

PUT /users/`<string:username>`

This method enables users to make changes to their accounts and their credentials.



```python
# Route to update user credentials by username
@app.route('/users/<string:username>', methods=['PUT'])
@login_required
def update_user_credentials(username):
    data = request.get_json()

    if not data:
        return jsonify({'error': 'Invalid request. Please provide data for update.'}), 400

    user = User.query.filter_by(username=username).first()

    if not user:
        return jsonify({'error': 'User not found.'}), 404

    # Update fields if provided in the request data
    user.username = data.get('username', user.username)
    user.email = data.get('email', user.email)
    user.sector = data.get('sector', user.sector)
    user.NEgrade = data.get('NEgrade', user.NEgrade)

    # Check if a new password is provided and update it
    new_password = data.get('password')
    if new_password:
        if len(new_password) < 6:
            return jsonify({'error': 'Password must be at least 6 characters long.'}), 400
        user.password = bcrypt.generate_password_hash(new_password).decode('utf-8')

    db.session.commit()

    return jsonify({'message': 'User credentials updated successfully'}), 200
```

PUT /states/`<string:state_name>`

This is an admin exclusive method that can update state information.

```python
# Update a state
@app.route('/states/<string:state_name>', methods=['PUT'])
@login_required
@admin_required
def update_state(state_name):
    state = State.query.filter_by(name=state_name).first()
    if not state:
        return jsonify({'error': 'State not found'}), 404

    data = request.json
    state.name = data.get('name', state.name)

    db.session.commit()
    return jsonify({'message': 'State updated successfully'})
```

PUT /states/<string:state_name>/universities/(ID or Name)

1. **PUT /states/<string:state_name>/universities/<int:uni_id>**
2. **PUT /states/<string:state_name>/universities/<string:uni_name>**

This method is again an admin only method that allows him to modify with the desired modification the university's information either by selecting university by ID or by name.

```python
# Update a university by name
@app.route('/states/<string:state_name>/universities/<string:uni_name>', methods=['PUT'])
@login_required
@admin_required
def update_university_by_name(state_name, uni_name):
    state = State.query.filter_by(name=state_name).first()
    if not state:
        return jsonify({'error': 'State not found'}), 404

    university = University.query.filter_by(name=uni_name, state_id=state.id).first()
    if not university:
        return jsonify({'error': 'University not found'}), 404

    data = request.json
    university.name = data.get('name', university.name)
    university.description = data.get('description', university.description)
    university.location = data.get('location', university.location)
    university.status = data.get('status', university.status)
    university.tuition_fee = data.get('tuition_fee', university.tuition_fee)
    university.specialty = data.get('specialty', university.specialty)
    university.degree = data.get('degree', university.degree)
    university.student_capacity = data.get('student_capacity', university.student_capacity)  # Added
    university.last_year_score = data.get('last_year_score', university.last_year_score)  # Added

    db.session.commit()
    return jsonify({'message': 'University updated successfully'})
```

2.8.4 DELETE Methods:

DELETE /users/<string:username>

This is an admin only method that allows admin type users to delete students and their respective tow inside the subject database.

```python
# Admin-only delete users function
@app.route('/users/<string:username>', methods=['DELETE'])
@login_required
@admin_required
def delete_user(username):
    user = User.query.filter_by(username=username).first()

    if not user:
        return jsonify({'error': 'User not found.'}), 404

    Subject.query.filter_by(user_id=user.id).delete()

    db.session.delete(user)
    db.session.commit()

    return jsonify({'message': f'User {username} deleted successfully'}), 200
```

DELETE /states/<string:state_name>

This is an admin only methods that enables admins to delete states and with them their corresponding universities.

```python
#delete a state
@app.route('/states/<string:state_name>', methods=['DELETE'])
@login_required
@admin_required
def delete_state(state_name):
    state = State.query.filter_by(name=state_name).first()
    if not state:
        return jsonify({'error': 'State not found'}), 404

    # Delete all universities associated with the state
    for university in state.universities:
        db.session.delete(university)

    db.session.delete(state)
    db.session.commit()
    return jsonify({'message': 'State and associated universities deleted successfully'})
```

DELETE /states/<string:state_name>/universities/(ID or Name)

DELETE /states/<string:state_name>/universities/<int:uni_id>

DELETE /states/<string:state_name>/universities/<string:uni_name>

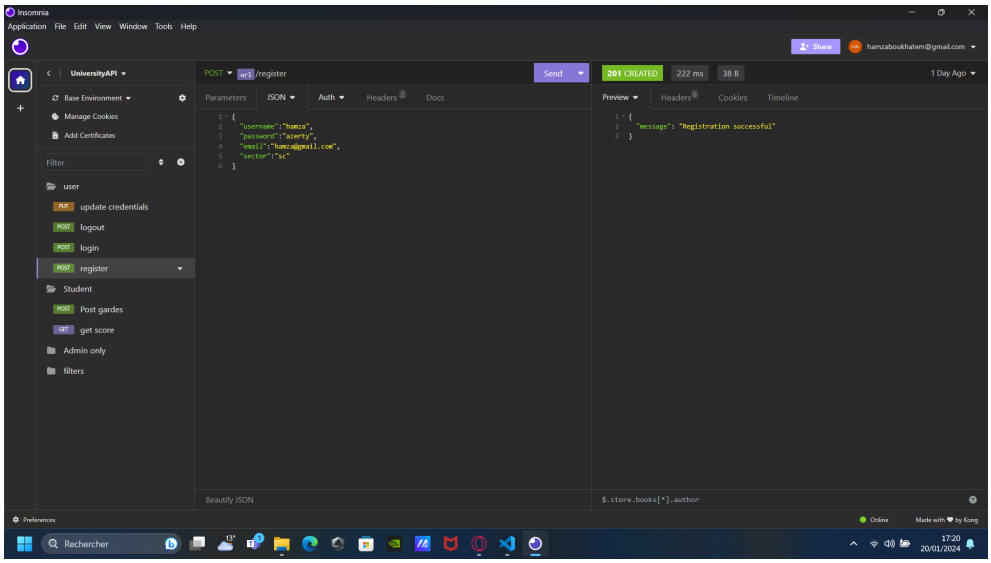
Methods that allow admin users only to delete universities referenced by ID or name and also the deleted university will decrement the state's number on universities by 1

```python
# Delete a university by id
@app.route('/states/<string:state_name>/universities/<int:uni_id>', methods=['DELETE'])
@login_required
@admin_required
def delete_university(state_name, uni_id):
    state = State.query.filter_by(name=state_name).first()
    if not state:
        return jsonify({'error': 'State not found'}), 404

    university = University.query.filter_by(id=uni_id, state_id=state.id).first()
    if not university:
        return jsonify({'error': 'University not found in the specified state'}), 404

    db.session.delete(university)

    # Ensure university_number is at a minimum of 0
    state.university_number = max(0, state.university_number - 1)

    db.session.commit()

    return jsonify({'message': 'University deleted successfully'})
```

2.9 Insomnia Contribution:

Insomnia is an open-source API testing tool that allows developers to test, debug, and document RESTful APIs .It is also a widely used API development tool that simplifies API design, testing, and management with its powerful features and user-friendly interface.[10]

Insomnia was a really helpful tool as it ,firstly, helped me with testing my endpoints and advance in my project but also through error handling queries made me single out coding mistakes and figure out how to best handle them

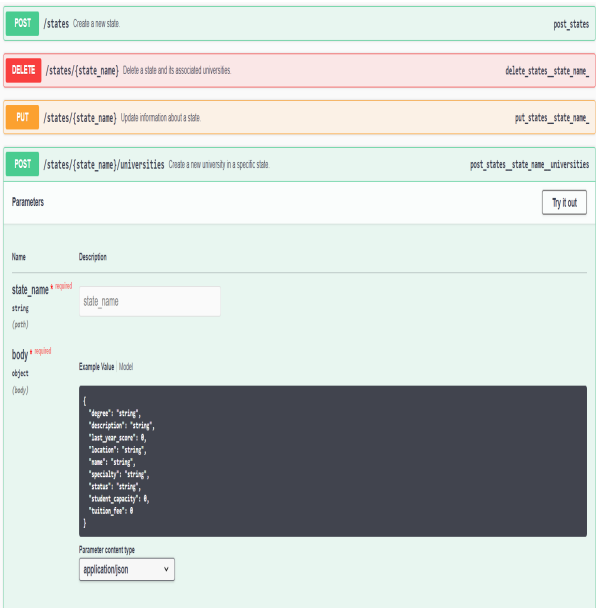


2.10 Swagger/Flasgger Contribution:

Swagger UI allows anyone to visualize and interact with the API's resources without having any of the implementation logic in place. It's automatically generated from your OpenAPI Specification, with the visual documentation making it easy for back end implementation and client side consumption.[6]

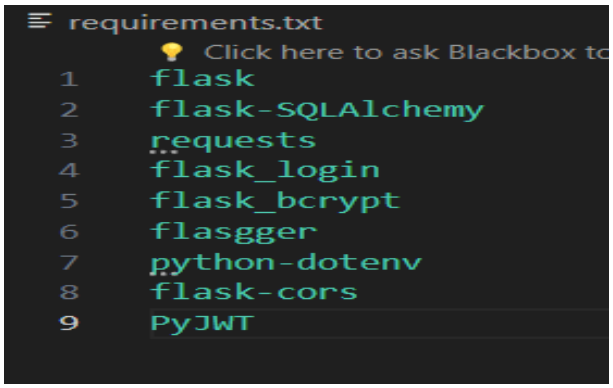
I used for this step a flask extension called flasgger which made making swagger documentation spontaneous when making my different methods , all it took was adding a detailed description right after defining the method like down below:

```
# register a user
@app.route('/register', methods=['POST'])
def register():
    """
    Register a new user.
    ---
    tags:
      - user
    parameters:
      - name: body
        in: body
        required: true
        schema:
          type: object
          properties:
            username:
              type: string
            password:
              type: string
            is_admin:
              type: boolean
    responses:
      201:
        description: User registered successfully
      400:
        description: Invalid request or username already taken
    """
```

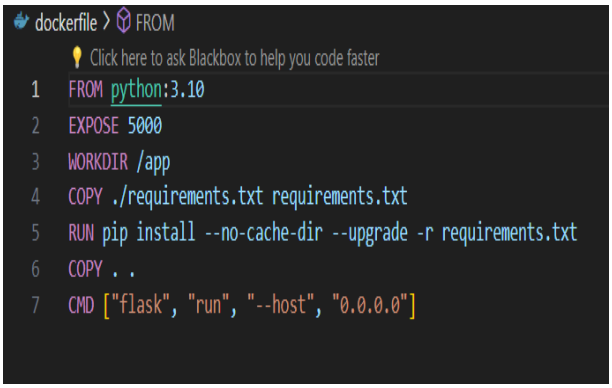


2.11 Docker Contribution:

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.The service has both free and premium tiers. The software that hosts the containers is called Docker Engine.[8] For the dockerization and containment of the project to further enhance its performance, I have created a requirement file where there are all the extensions and inputs that i used and prepared a dockerfile filled with the necessary parameters to build and run a docker image.



```
requirements.txt
Click here to ask Blackbox to
1  flask
2  flask-SQLAlchemy
3  requests
4  flask_login
5  flask_bcrypt
6  flasgger
7  python-dotenv
8  flask-cors
9  PyJWT
```

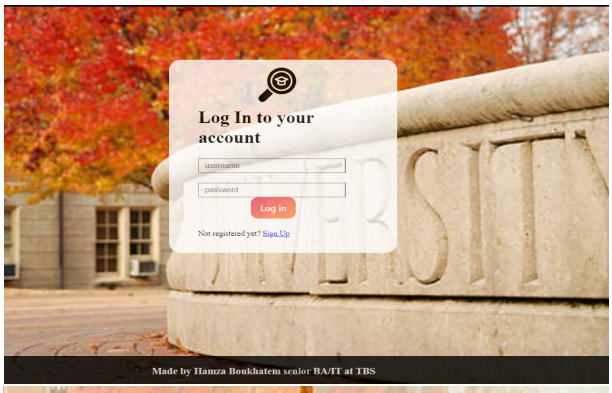


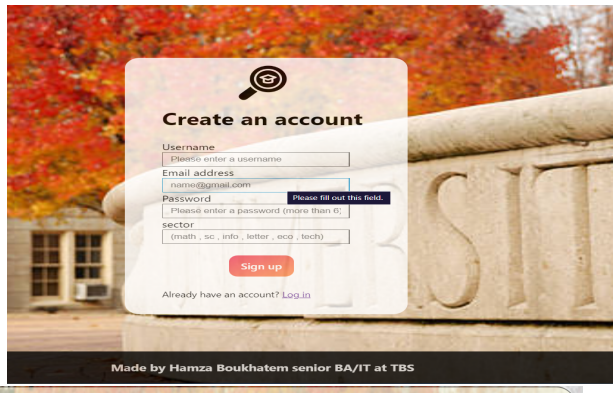
```
dockerfile > FROM
Click here to ask Blackbox to help you code faster
1  FROM python:3.10
2  EXPOSE 5000
3  WORKDIR /app
4  COPY ./requirements.txt requirements.txt
5  RUN pip install --no-cache-dir --upgrade -r requirements.txt
6  COPY . .
7  CMD ["flask", "run", "--host", "0.0.0.0"]
```

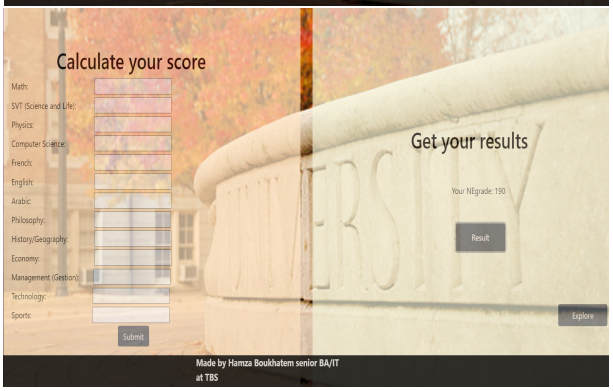
2.12 HTML/CSS/JS Contributions:

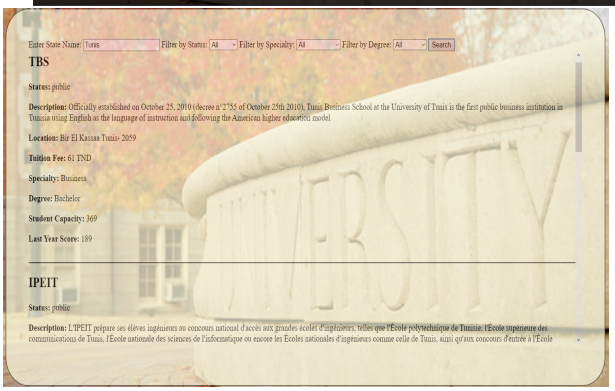
After making sure all my app functions are perfectly working and everything is smooth sailing, I decided to add to my project an interactive interface that will host some of these methods that i created and tried to enhance user experience. For that i used basic web development programming language HTML for page layout , CSS for the beautification of the page and JavaScript to make the link between my app methods and the input spaces and buttons on the web pages.

Below i will showcase the pages that i have developed:









2.13 Flask-CORS Contribution:

A Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible.This package has a simple philosophy: when you want to enable CORS, you wish to enable it for all use cases on a domain. This means no mucking around with different allowed headers, methods, etc.[3]

Flask-CORS has a very easy implementation "CORS(app)". That simple line allows your app to run on multiple domains at the same time the same way.

2.14 Data Structure:

I have use SQLite and SQLAlchemy to design my tables; there are 4 tables in total:

A User table , a Subject table , a State table and University table

2.14.1 States Table:

Contains 3 columns:

1. id : The state's ID
2. name : the state's name
3. number of universities : a value that is computed automatically corresponding to the sum of universities inside a stat .

2.14.2 Universities Table:

Contains 11 columns:

1. id : The university's ID
2. name : the university's name
3. description : description of university , mission and goals
4. location : university address
5. status : takes either public or private
6. tuition fee : value of tuition per year
7. specialty : the university main specialty
8. degree : the type of degree
9. student capacity : capacity of new students
10. last year score : the lowest score to attain the university last year
11. state id : ID of corresponding state (foreign key)

2.14.3 User Table:

Contains 7 columns:

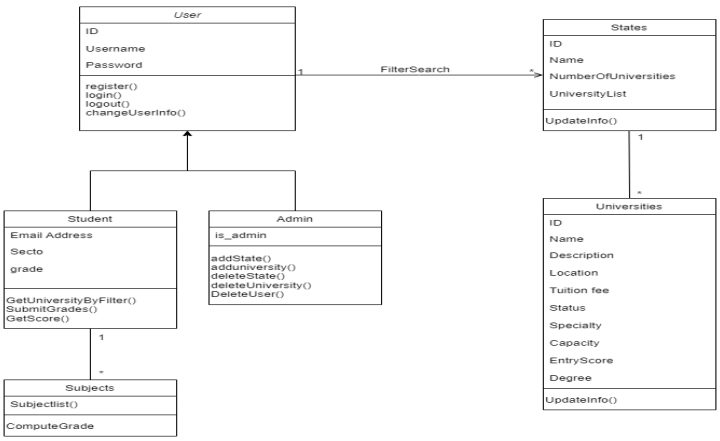
1. id : The user's ID
2. username : the user's name
3. password : hashed user password
4. email : user's email address
5. sector : sector taken in high school
6. NEgrade : National Examination Grade
7. is_admin : a hidden column to make the identification between admins and students

2.14.4 Subject Table:

Contains 15 columns:

this SQL database contains all the subjects that students sit for in the national examination, a column for ID and another for user_id (a foreign key)

Additionally the columns are not all required to be filled with non null values as they take a default value of 0 and will not have any effect on NEgrade calculations as there are conditional statement that select subjects appropriate for eash sector a student studies in .



Chapter 3

Challenges

3.1 Authentication and Authorization:

When designing my User authorization methods i wanted to include an authetication method with the login and logout methods that i have created . for that i tried to introduce tokens to my code, I downloaded the library needed which was "PyJWT" which allows you to create your token and customize its duration and its defining parameters.
But unfortunately it was not compatible with the authorization methods that uses decorators to set up the level of authorization for each method.
That is why i scrapped that idea and resorted to using basic authentication and authorization methods with more security to reinforce them such as hashing credentials using Bcrypt

3.2 BackEnd JavaScript Connections:

When i finished setting up my methods and after testing them, I decided to add to my project some web development aspects and create a user interface . The design part was smooth sailing with a lot of ideas like splitting the screen and using bootstrap for button and interactivity design.
The difficulties started however in the Script section ; making connections to the original applications was very hard in the beginning as multiple session availability wasn't working which prompt me to install "flask_CORS" but the most important difficulty was to maintain user authentication when redirecting to another page; the solution to that was to manipulate the redirect Href into taking the username modifier in his link which will allow the next page to have access to user info of the user that just logged in .

Chapter 4

Conclusion

At the of the project, I was overwhelmed with the amount of tools, resources and extensions presented before me. I was lost and settle for the first idea that came to mind , and that was forming a university library of some sort with very basic methods of management.

But at the end of each step i found myself adding more and more to the project, first was the numerous filters i implemented then it was maybe modifying the database models so i can have more freedom with my methods and functions.

After finishing with the core of the project and the thing that relates to the topic of university, my thought process drifted towards adding user functionalities such as registration , and from that came the duality of student only functions and admin only authority and prompted me to add login managers and authorization levels .

I found myself ,even with the constraint of time, enjoying this application of what i learned about API development and the new things I was discovering on the go: I learned about flask extensions and the different utilities it had , I also learned how to make the connections between the flask app and the web pages i found myself adding to my project. Overall it was an enjoyable experience and a good learning opportunity that will no doubt serve me well in the future.

Thank you for reading and thanks for the efforts you put into the class lesson, this course was very helpful and enriching .

Appendix A

Some Methods Responses:

A.1 POST Methods:

A.1.1 POST /register

"message": "Registration successful"

A.1.2 POST /login

"message": "Login successful"

A.1.3 POST /logout

"message": "Logout successful"

A.1.4 POST /states

"id": 6,

"message": "State added successfully"

A.1.5 POST /states/Tunis/univesities

"id": 12,

"message": "University created successfully"

A.2 PUT Methods:

A.2.1 PUT /users/student

"message": "User credentials updated successfully"

A.2.2 PUT /states/Tunis/universities/1

"message": "University updated successfully"

A.2.3 PUT /states/Tunis/universities/TBS

"message": "University updated successfully"

A.2.4 PUT /states/Monastir

"message": "State updated successfully"

A.3 DELETE Methods:

A.3.1 DELETE /states/Arianna/universities/unigabes

"message": "University deleted successfully"

A.3.2 DELETE /states/Tunis/universities/6

"message": "University deleted successfully"

A.3.3 DELETE /states/gabes

"message": "State and associated universities deleted successfully"

A.3.4 DELETE /users/hamza

"message": "User hamza deleted successfully"

A.4 GET Methods:

A.4.1 GET /users/student1/NEgrade

"NEgrade": 190.27

A.4.2 GET /states


```
{
  "id": 3,
  "name": "Sousse",
  "universities": [
    {
      "id": 7,
      "name": "Ibn El Jazzar Medical Faculty of Sousse",
      "specialty": "Medecine",
      "status": "public"
    }
  ],
  "university_number": 1
},
{
  "id": 4,
  "name": "Carthage",
  "universities": [
    {
      "id": 8,
      "name": "IHECC",
      "specialty": "Business",
      "status": "public"
    }
  ],
  "university_number": 1
},
```


A.4.3 GET /universities/name/FMT


```
{
  "degree": "Doctorat",
  "description": "Notre mission est de former des médecins,
des enseignants et des chercheurs compétents, sociétalement
responsables et répondant aux besoins en santé de la
communauté et de promouvoir l'excellence dans la recherche et
l'innovation en tenant compte des évolutions pédagogiques,
didactiques, scientifiques et techniques pertinentes.",
  "id": 3,
  "last_year_score": 186.65,
  "location": "R534+F9H, Rue de la Faculte de Medecine,
Tunis",
  "name": "FMT",
  "specialty": "Medecine",
  "state": "Tunis",
  "status": "public",
  "student_capacity": 327,
  "tuition_fee": 50.0
}
```

Bibliography

[1] *About Flask and its documentation.* https://flask.palletsprojects.com/en/3.0.x/.

[2] *About Flask-Bcrypt and its documentation.* https://flask-bcrypt.readthedocs.io/en/1.0.1/.

[3] *About Flask-CORS and its documentation.* https://flask-cors.readthedocs.io/en/latest/.

[4] *About Flask-login and its documentation.* https://flask-login.readthedocs.io/en/latest/.

[5] *About SQLite.* https://www.sqlite.org/about.html.

[6] *About Swagger and its Documentation.* https://swagger.io/tools/swagger-ui/.

[7] *SQLAlchemy documentation.* https://www.sqlalchemy.org.

[8] docker.com. what is a container? 2019-05-19. https://www.docker.com/resources/what-container/.

[9] Stalon Lee. "how to run code in vs code". *Alphr.*, 2022-06-02. https://www.citedrive.com/overleaf.

[10] Ahmed Waheed. Overview of insomnia api and the best insomnia app alternative. *APIDOG*, 2023-10-19. https://apidog.com/blog/insomnia-api/.