**ICS 2203: INTERNET APPLICATION PROGRAMMING PROJECT**

**PROJECT NAME:** MyUniversity-Portal

**GITHUB LINK:** <https://github.com/d3q0w/MyUniversity-Portal>

**PROJECT OVERVIEW**

* This project takes shape of a university portal where students can create university student accounts. It mimics top university portals such as JKUAT Student portal.

**DISTINCTIVENESS AND COMPLEXITY**

**(I) DISTINCTIVENESS OF THE PROJECT**

* This is not just a mere university portal, it is an adaptive, self-optimizing portal that utilizes eco-friendly attributes like Green-Docker for containerization and power-efficient Database queries that shape needs, compliance enforcement and promotes technology growth.
* This MY\_UNIVERSITY\_PORTAL project shines light to the current trend in how modern websites are created, managed and operated in this sprawling technological world, highlighting the need for future improvements on the current, monolithic websites with normal login pages and university student accounts.
* What makes this project different from the other groups is that it incorporates cutting-edge technologies beyond the class reference. In the course work, we only utilized one Django application, where everything like models and templates were hosted in one folder. However, this uses more than one Django application. Each feature: Enrollment, Attendance, Grading, Reporting and User Management: uses its own self-contained Django application, hence each behaves like a domain specific “mini-service” which makes addition of plugins for the future.

**COMPLEXITIES INVOLVED**

**1. Modular, Event-driven architecture**

* The project has independent Django applications for enrollment, attendance, grading, reporting and user management. It also has a clear service interface that supports easy plugins of new features, example: Financial Aid.

**2. Real-Time Processing**

* The student dashboard is powered by Web-sockets that ensure real-time processing of activities, example: Attendance check-ins.

**3. Security and Compliance**

* There is an immutable journaling of any changes in the system, such as: who or when a student came and left the class. In addition, all of the student’s Personally Identifiable Information (PII) such as name, date of birth and admission numbers, are encrypted. Runtime encryption is also involved to block unethical privilege escalation.

**4. Device-Agnostic User Experience**

* It is able to engage offline support and compatibility on any device.

**DESIGN APPROACH**

* We followed a **Domain-Driven Design (DDD),** which has its foundation on modular architecture.
* We focused on building each model as a modular “micro-service” which enables each module (the Django apps) to independently run.
* By following this approach, each service: Enrollment, Attendance, Grading, Reporting and User Management: lives in its own “micro-application”.
* Each application has a domain layered with entities, value-attributes and important domain services.
* The Web Layer has Django views and REST serializers to create a dynamic flow of data and a security layers between requests.
* In addition, the web application incorporates infrastructure laying, example: Caching. This helps with accessing frequently accessed data without directly querying the same data over and over again.

**Steps involved in the design approach:**

1. We identified the services and entities of the web application after coming up with the idea. The services included the Attendance, Grading, Reporting and User Management.
2. The entities were then drafted and consolidated to build work flow and data flow diagrams to visualize the web application, since it was no ordinary web application.
3. Each of the members then took a service to work on and build it service independently by implementing independent Django application before consolidating everything as one major web application service.
4. Consolidation was then taken up by wiring inter-application communication so that they communicate amongst each other and know what each service does within the system.
5. Lastly, each unit was tested and debugged before submission of the work.

**Why this Design Approach:**

1. **There is neat separation of concerns**: Each member was able to research and design their “micro-app” independently and work parallel ways, hence reducing conflict and excessive overlap.
2. **Easy debugging and testing**: By working on each service independently, we were able to easy configure, debug and test each unit on its own, making work easier. In a point of failure, tracing the problem becomes a small issue and is quickly solved.
3. **Plug-and-Play extensibility**: This design approach allows us to add more plugins and services without modifying the core. Example: Financial Aid service. This is a plugin that will be easily added upon scalability of the web application.
4. **Easy to Maintain**: Since each service is built independently, the code-base for each application is smaller hence making it easier to manage and maintain the whole web application even for the future.

**WHAT IS CONTAINED IN EACH FILE**

* There was a total of 7 program folders and one other file, manage.py. The folders collectively include:
* **settings.py** – for configuration and installation of the applications.
* **urls.py** – for setting up paths for communication between the
* **admin.py**- which is responsible for the super user and admin clauses of the web application.
* **apps.py** – which has the base configurations of all the applications built
* **models.py** – which uses Django to import models used in the project for creation of Database. The database takes student details, stores them and utilizes efficient queries.
* **tests.py** – which imports TestCase for testing.
* **views.py** – which contains all the necessary views needed for the web application.
* **manage.py** – which is a command line interface utility used for creation of the apps before installation. It also enables running the local environment server, applying database migrations, creation of super users and running tests.
* **forms.py** – which has Meta data from the models.py created for the students.
* **Assets**: This is folder which has a couple of HTML, CSS and JavaScript files, like index.html, animate.html, style.css, script.js. Those are just but a few in many static files available in this folder.
* **Templates**: This is also another folder implicitly for html files and Django forms. It has the base.html, authentication.html, login.html among others. They are created to display the relevant User Interface to the students as they navigate the website.

**STEPS TO RUN THE WEB APPLICATION ON YOUR LOCAL MACHINE.**

* Straight from one of our members GitHub account, below is the documented procedure on how to run the application on your local machine:

Follow the steps below to get the project up and running on your local machine:

**1. Clone the Repository**

git clone https://github.com/d3q0w/MyUniversity-Portal.git

cd MyUniversity-Portal

**2. Create a Virtual Environment**

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

**3. Install Dependencies**

pip install -r requirements.txt

**4. Set Up Database**

Make sure to set up your database (e.g., PostgreSQL) and update the DATABASES configuration in MyUniversity-Portal\*\*/settings.py.

**5. Run Migrations**

python manage.py migrate

**6. Create Superuser**

python manage.py createsuperuser

if this command fails, you can run this:

winpty python manage.py createsuperuser

**7. Run the Application**

python manage.py runserver

Visit http://localhost:8000 to access the app.

**Configuration**

You'll need a .env file for environment-specific configurations. Example:

DEBUG=True

SECRET\_KEY='your\_secret\_key'

DATABASE\_URL=postgres://user:password@localhost:5432/your\_db\_name

Make sure to configure settings like database, static files, and email backend properly for production.