University-Identity-Management-and-Access-Application

Design Manual

Submitted by

Rahul Vijaykumar
Satya Sai Deepak Velagapudi
Yu Liu

Table of Contents:

Introduction

The University-Identity-Management-and-Access-Application is a web-based application designed to manage courses. Developed using the Django framework, it seamlessly integrates with a MySQL database to handle data retrieval and dynamically generate web pages for display.

The system provides the functionalities for three kinds of users: Admin, Professor, and Student.

Admins can perform the following tasks:

- F1. Create a list of professors sorted by one of the following criteria chosen by the admin:
  (1) by name
  (2) by dept
  (3) by salary
- F2. Create a table of min/max/average salaries by the department.
- F3. Create a table of professor names, dept, and the total number of students taught by the professor in a given semester.

Professors can perform the following tasks:

- F4. Create the list of course sections and the number of students enrolled in each section that the professor taught in a given semester.
- F5. Create the list of students enrolled in a course section taught by the professor in a given semester.

Students can perform the following tasks:

- F6. Query the list of course sections offered by the department in a given semester and year.

Design Overview

The university's backend database is constructed using MySQL, with the Django web framework facilitating connectivity. Information retrieval for web page display is facilitated through Django's connection to MySQL. HTML is utilized for web page design, with CSS providing styling.

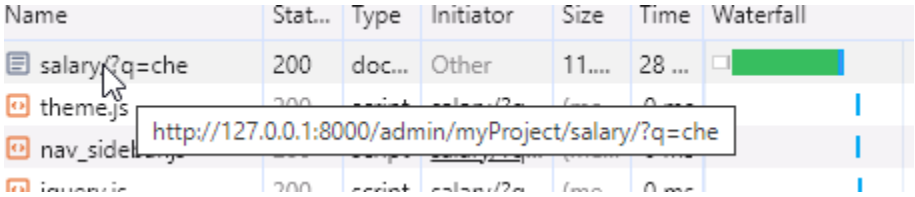
The application's core functionality involves retrieving data from the backend database and dynamically generating web pages tailored to specific user tasks. Python adheres to a structured directory and file organization, with the views.py file housing functions responsible for processing web page requests, backend data manipulation, and subsequent response generation.

Data extraction in the admin interface differs significantly from that of students and professors, as it leverages models to retrieve data from the database rather than relying on raw queries. Each model corresponds to a table in the database, streamlining migration processes. This approach ensures that deploying the code on a new system without the existing database will automatically create the necessary tables upon running a migration command. Moreover, the admin functionality adheres to the Model-View-Template (MVT) format, ensuring separation of concerns and maintainability. Below is the migration query to generate both the default Django tables and those essential for the UML functionality.

```
\University Identity and Access Management Application\EE46801_S24_University-Identity-Management-and-Access-Application> python manage.py migrate
```

The process of receiving the input data provided by the user from the web page to display the results include several steps as follows.

1. Input from the webpage is transmitted to the backend update through an HTTP request initiated from the browser.



2. The application establishes a connection with the local database via the settings.py file.

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "university",
        "USER": "root",
        "PASSWORD": "",
        "HOST": "localhost",
    }
}
```

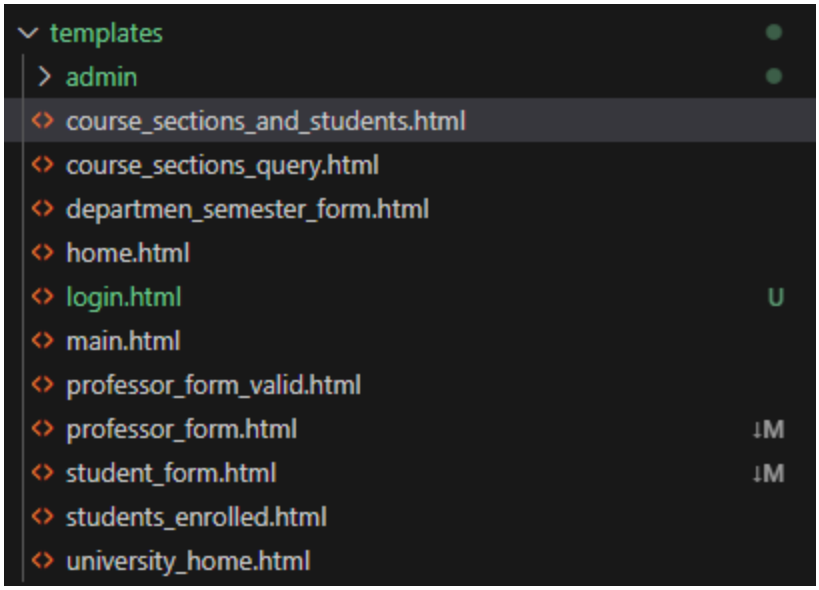
3. Model for each database that is extracted is created in model.py which is connected to the established database

```
    Codiumate: Options | Test this class
56  class Student(models.Model):
57      student_id = models.CharField(max_length=10, primary_key=True)
58      name = models.CharField(max_length=32)
59      dept_name = models.CharField(max_length=32)
60      total_credits = models.IntegerField()
61
    Codiumate: Options | Test this class
62  class Takes(models.Model):
63      student_id = models.ForeignKey(Student, on_delete=models.CASCADE)
64      course_id = models.CharField(max_length=10)
65      sec_id = models.CharField(max_length=5)
66      semester = models.CharField(max_length=10)
67      year = models.IntegerField()
68      grade = models.CharField(max_length=2)
69
```

4. Once the connection is established successfully, the background MySQL query will be executed using the “mycursor” and “query” command

```
if request.method == "POST":
    department = request.POST.get("department")
    semester = request.POST.get("semester")
    year = request.POST.get("year")
    print(department, year, semester)
    with connection.cursor() as cursor:
        cursor.execute(
            """
            SELECT c.title,s.course_id, s.sec_id, s.building, s.room, s.capacity
            FROM Section as s
            INNER JOIN Course as c
            ON t.course_id=c.course_id
            WHERE c.dept_name = %s AND t.year = %s AND t.semester = %s;
        """,
            [department, year, semester],
        )
        course_sections = cursor.fetchall()
    print(course_sections)
    return render(
        request, "course_sections_query.html", {"course_sections": course_sections}
    )
else:
    return render(request, "departmen_semester_form.html")
```

5. After extracting data from the database, it is subsequently passed to the corresponding templates. These templates then display the information on the webpage in table format, adhering to the specified requirements.



6. Each template is linked to its corresponding function within views.py via urls.py. Within urls.py, each action is directed to a specific webpage.

```
urlpatterns = [
    path("", views.home, name="home"),
    path("dashboard/", views.dashboard, name="dashboard"),
    path('professor/', views.professor, name='professor'),
    path('professor_valid/<str:professor_id>/', views.professor_valid, name='professor_valid'),
    path('student/', views.student, name='student'),
    path('department_semester_form/',views.department_semester_form,name='department_semester_form'),
    path('professor/logout/', views.professor_logout, name='prof_logout'),
    path('student/logout/', views.student_logout, name='stu_logout'),
    path('admin-page/', RedirectView.as_view(url='/admin/'),name="admin")
    ]
```

Database Schemas

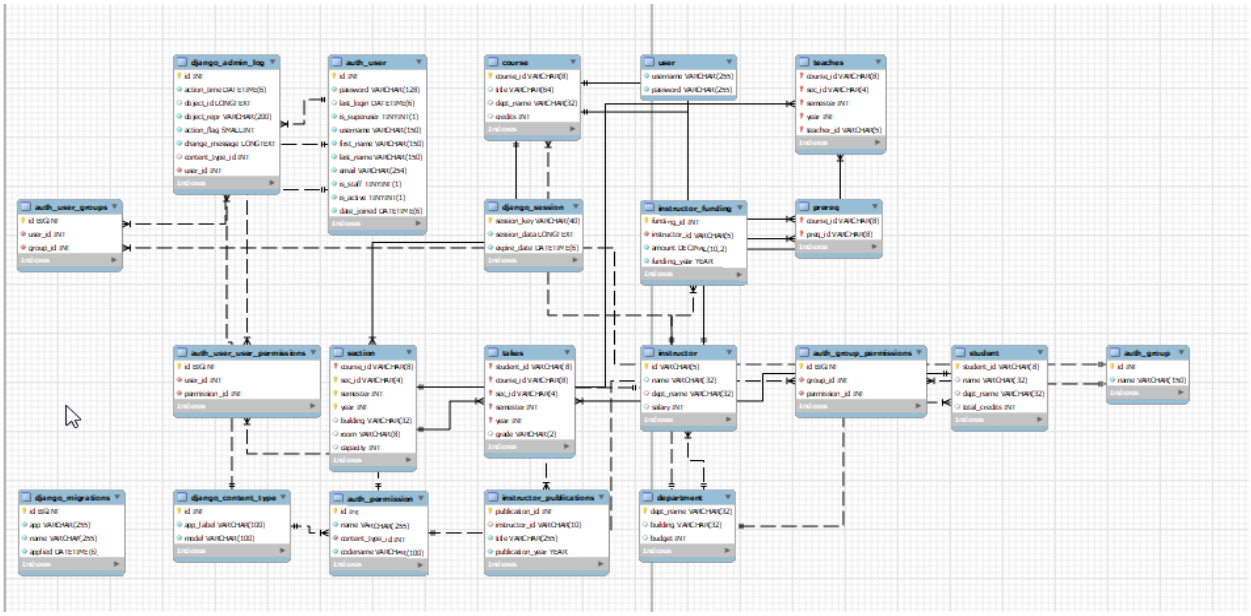
Following are the additional database schemas apart from the default django schemas created

1. *Student (ID, name, dep_name, tot_cred)*
2. *Instructor (id, name, dep_name, salary)*
3. *Department (name, building, budget)*
4. *Course (course_id, title, dep_name, credits)*
5. *Section (course id, sec id, semester, year, building, room number, time slot id)*
6. *Takes (ID, course_id, sec_id, semester, year, grade)*
7. *Teaches (ID, course_id, sec_id, semester, year)*
8. *Instructor_funding (funding_id, instructor_id, amount, funding_year)*
9. *Instructor_publications (publication_id, instructor_id, title, publication_year)*

Entity-relationship diagram (ERD)

Following is the ERD for the University database. The below diagram contains all the entities that are present in the database. We made use of the following entities from the University Database:

- Student
- Instructor
- Department
- Course
- Section
- Takes
- Teaches
- Instructor_funding
- Instructor_publications



Security

Authentication and Authorization:For admin, extra authentication code is not required as Django's built-in admin page already provides authentication functionality. The Django admin interface includes features such as login functionality, user authentication, and access control, making it inherently secure for managing administrative tasks related to the application's data model.

The code implements user authentication for both professors and students using Django's built-in authentication system. Users are required to provide valid credentials (ID and password) to access their respective functionalities. Additionally, the @user_passes_test decorator is used to restrict access to certain views based on user roles (prof and stu functions).

Authentication code For professor

```
def professor(request):
    if request.method == 'POST':
        professor_id = request.POST.get('professor_id')
        password = request.POST.get('password')

        # Check if the professor ID exists in the database
        with connection.cursor() as cursor:
            cursor.execute("SELECT * FROM Instructor WHERE id = %s", [professor_id])
            professor_exists = cursor.fetchone()

        if professor_exists:
            #authenticate and login the professor
            #default password is passwword
            user = authenticate(request, username='professor', password=password)
            if user is not None:
                login(request, user)
                return redirect('professor_valid', professor_id=professor_id)
            else:
                # Incorrect password
                return render(request, 'professor_form.html', {'invalid_details': True})
        else:
            return render(request, 'professor_form.html', {'invalid_details': True})
    else:
        return render(request, 'professor_form.html')


```

```
Codeium: Refactor | Explain | Generate Docstring | ×
def prof(user):
    if str(user)=="professor":
        return True
    return False
```

```
Codeium: Refactor | Explain | Generate Docstring | ×
@user_passes_test(prof, login_url="/professor/")
def professor_valid(request, professor_id):
    if request.method == 'POST':
        action = request.POST.get('action')
        if action == 'f4':
            # Fetch semester and year from the form
            semester = request.POST.get('semester')
            year = request.POST.get('year')
            print("HI")
            print(semester,year)
            # Fetch course sections and students enrolled for the professor in the chosen semester and year
            with connection.cursor() as cursor:
```

Authentication code For student.

```
def student(request):
    if request.method == 'POST':
        student_id = request.POST.get('student_id')
        password = request.POST.get('password')
        # Check if the student ID exists in the database
        with connection.cursor() as cursor:
            cursor.execute("SELECT * FROM Student WHERE student_id = %s", [student_id])
            student_exists = cursor.fetchone()


        if student_exists:
            #default password is passsword
            user = authenticate(request, username='student', password=password)
            if user is not None:
                login(request, user)
                return redirect('department_semester_form')
            else:
                # Incorrect password
                return render(request, 'student_form.html', {'invalid_details': True})
        else:
            return render(request, 'student_form.html', {'invalid_details': True})
    else:
        return render(request, 'student_form.html', {'invalid_details': False})
```

```
Codeium: Refactor | Explain | Generate Docstring | ×
def stu(user):
    if str(user)=="student":
        return True
    return False
```

```
Codeium: Refactor | Explain | Generate Docstring | ×
@user_passes_test(stu, login_url="/student/")
def department_semester_form(request):
    if request.method == 'POST':
        department=request.POST.get('department').upper()
        semester=request.POST.get('semester')
        year=request.POST.get('year')
        print(department,year,semester)
        with connection.cursor() as cursor:
            cursor.execute("""
```

Logout Functionality: Both professors and students have the ability to log out by clicking a "logout" button on the HTML webpage. When the logout button is clicked, it triggers a specific URL pattern within the application, which in turn invokes a corresponding view function in the views.py file. Within this view function, Django's logout() function is called to terminate the user's session and clear any session-related data. After successfully logging out, the view redirects the user to the application's homepage, represented by the home view.

```
myProject > urls.py > ...
1   from . import views
2   from django.urls import path
3   from django.views.generic import RedirectView
4
5
6   urlpatterns = [
7       path("", views.home, name="home"),
8       path("dashboard/", views.dashboard, name="dashboard"),
9       path('professor/', views.professor, name='professor'),
10      path('professor_valid/<str:professor_id>/', views.professor_valid, name='professor_valid'),
11      path('student/', views.student, name='student'),
12      path('department_semester_form/',views.department_semester_form,name='department_semester_form'),
13      path('professor/logout/', views.professor_logout, name='prof_logout'),
14      path('student/logout/', views.student_logout, name='stu_logout'),
15      path('admin-page/', RedirectView.as_view(url='/admin/'),name="admin")
16      ]
17
18
```

```
Codeium: Refactor | Explain | Generate Docstring | ×
def student_logout(request):
    logout(request)
    return redirect('home')
```

```
Codeium: Refactor | Explain | Generate Docstring | ×
def professor_logout(request):
    logout(request)
    return redirect('home')
```

Secure Password Handling: While the code doesn't explicitly show password hashing, it relies on Django's authentication system, which handles password hashing and verification securely. This ensures that passwords are not stored in plaintext in the database, enhancing security.

Session Management: The code manages user sessions securely by using Django's session management functionality. This helps maintain session state securely, ensuring that users remain authenticated throughout their interactions with the application.

Cross-Site Request Forgery (CSRF) Protection: Django provides built-in CSRF protection, which mitigates CSRF attacks by including a CSRF token in forms and validating it upon submission. This protection is crucial for preventing unauthorized requests from being executed on behalf of authenticated users.