DB-backed Web Application Design Manual
Rowan Rishe, Xavier Lora, Ryan Converse, Kolade Ayeni
4/29/2024

# Introduction

This manual is intended to serve as a comprehensive guide for developers, administrators, and users to understand and effectively interact with the web application designed for educational institutions. It covers system architecture, user functionalities, and technical specifications to ensure ease of use and maintenance.

**Target Audience:**

- System Administrators
- Professors and Academic Staff
- Students
- Developers maintaining and enhancing the application

**High-level Architecture:**

The system is built using Django, a high-level Python web framework that encourages rapid development and clean pragmatic design. The architecture consists of:

- **Models:** Define the database schema and data interactions.
- **Views:** Handle the business logic and interact with the models to pass data to templates.
- **Templates:** Manage the presentation layer which users interact with through their web browser.

**Components of the System:**

- **Views File (‘Views.py’):** Contains functions that define the logic for each user interaction
  and data processing.
- **Models File (‘models.py’):** Specifies the database schema with Django models.
- **Templates Directory:** Holds HTML files for rendering the information on the web page

## User Roles and Permissions

**Admin:**

- Can access and manipulate professor listings and salary data.
- Able to view and analyze detailed performance metrics of professors.
- Manage and execute system-wide updates and configurations.

**Professors:**

- View and manage course sections they are teaching.
- Access enrollment lists and interact with student data for their courses.

**Students:**

- Query available course sections based on departmental offerings for specific academic terms.
- Interact with the course data to plan their academic schedules.

# Views and Logic:

This section of the manual elaborates on the views handling and logic for the login functionality and then each user functionality. It explains how the web application processes data and generates responses based on user requests.

## Login:

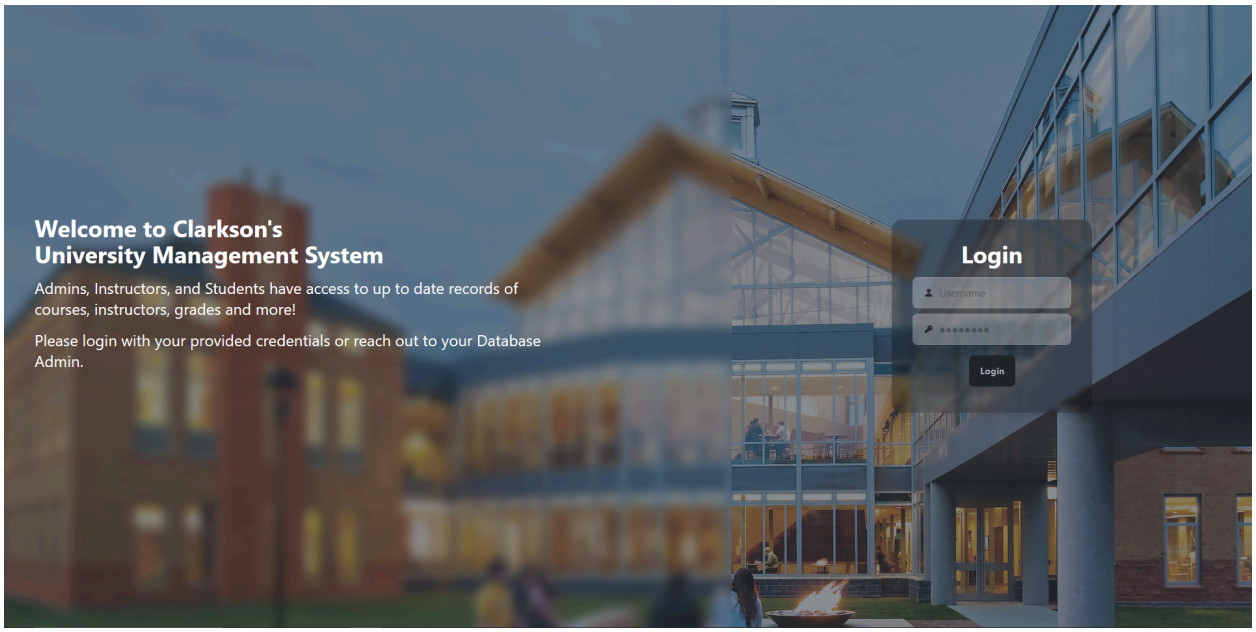
**Function:** 'index'

**Logic:**

- **Action Recognition:** The function checks for POST request indicating login submission
- **Parameter Retrieval:** Utilizes .get() to retrieve the username and password from POST request
- **Database Querying:** Using Django's built-in authentication we authenticate the user and return user type for proper dashboard redirect
- **Data Processing:** Based on user type, the template changes to the proper dashboard using the Group.objects.filter() function from the built-in Django authentication
- **Rendering:** The returned render templates are admin_dashboard.html, instructor_dashboard.html, and student_dashboard.html
- **Code Snippet:**

```python
def index(request):
    if request.method == 'POST':
        # Handle login form submission
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            if user.is_superuser:
                template = loader.get_template('dbApp/admin_dashboard.html')
                context = {}
                return HttpResponse(template.render(context, request))
            else:
                if Group.objects.filter(user=user, name='instructors').exists():
                    template = loader.get_template('dbApp/instructor_dashboard.html')
                    context = {}
                    return HttpResponse(template.render(context, request))
                elif Group.objects.filter(user=user, name='students').exists():
                    template = loader.get_template('dbApp/student_dashboard.html')
                    context = {}
                    return HttpResponse(template.render(context, request))
        else:
            print("Invalid login attempt")
            messages.error(request, 'Invalid username or password.')
            template = loader.get_template('dbApp/login.html')
            context = {'error': True}
            return HttpResponse(template.render(context, request))
    else:
        # Render the login form
        template = loader.get_template('dbApp/login.html')
        context = {}
        return HttpResponse(template.render(context, request))
```

## Preview:




## General Structure of Features

**Common Pattern Across Features:**
Each feature in the application follows a consistent format that simplifies understanding and maintaining the code. Here's the common workflow for each feature:

1. **Action Recognition:** Each view function begins by identifying the action requested by the user, typically passed as a parameter in a GET request.
2. **Parameter Retrieval:** After recognizing the action, the function retrieves necessary parameters from the GET request.
3. **Database Querying:** Using the retrieved parameters, the function executes relevant database queries. This might involve sorting data or fetching specific records.
4. **Data Processing:** Depending on the feature, the function may sort the results or append them to lists for further use.
5. **Rendering:** Finally, the function renders an HTML template, passing the processed data as context to be displayed on the web page.

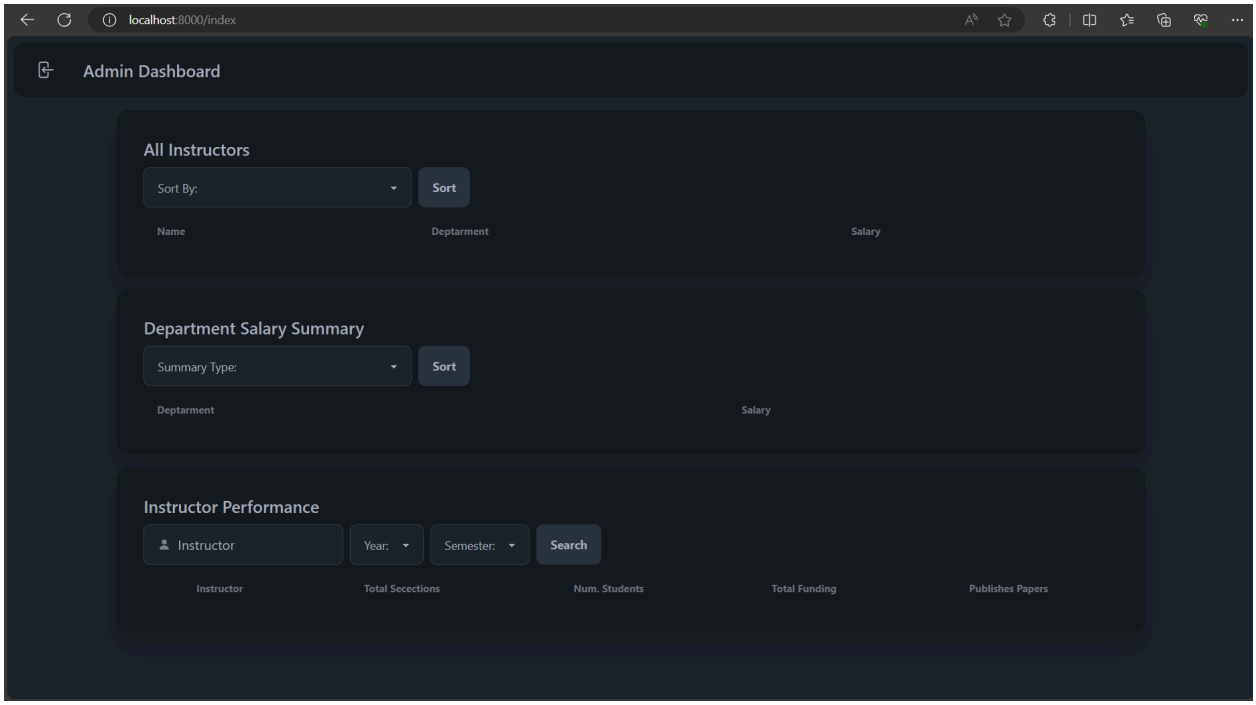
**F1: Roster Management (Admin)**
**Endpoint:** '/admin_dashboard_view?action=sort_instructor&sort_by=<criteria>'
**Function:** 'admin_dashboard_view'
**Logic:**

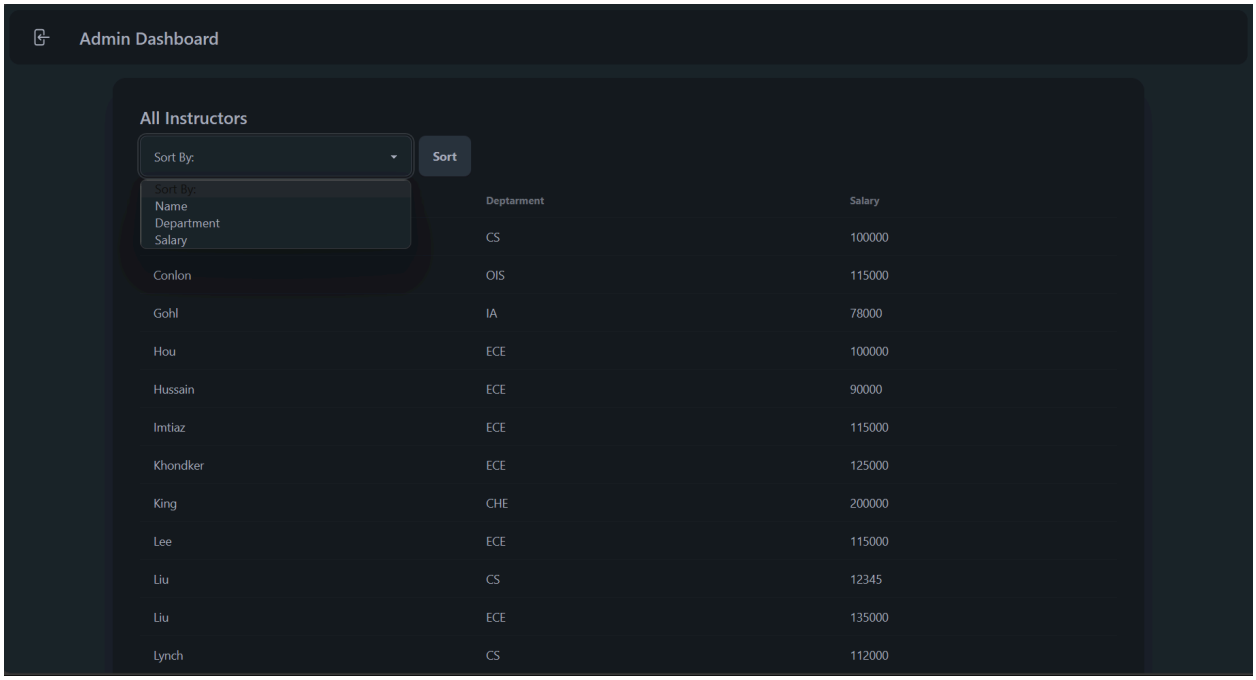
- **Action Recognition:** The function identifies the 'sort_instructors' action from the GET request.
- **Parameter Retrieval:** It fetches the 'sort_by' parameter from the GET request to determine the sorting criterion (name, department, or salary).

- **Database Querying:** Executes a query on the ‘Instructor’ model to retrieve all instructors using ‘Instructor.objects.all()’ and orders them based on the sorting criteria using ‘order_by(sort_by)’.
- **Data Processing:** No significant data processing outside of sorting is required for this feature.
- **Rendering:** The sorted list of instructors is passed to the ‘admin_dashboard.html’ template for display.
- **Code Snippet:**

```
def admin_dashboard_view(request):
    if request.method == 'GET':
        if 'action' in request.GET:
            action = request.GET.get('action')
            if action == 'sort_instructors':
                sort_by = request.GET.get('sort_by')
                instructors = Instructor.objects.all().order_by(sort_by)
                context = {'instructors': instructors}
                template = loader.get_template('dbApp/admin_dashboard.html')
                return HttpResponse(template.render(context, request))
```

- **Preview:**





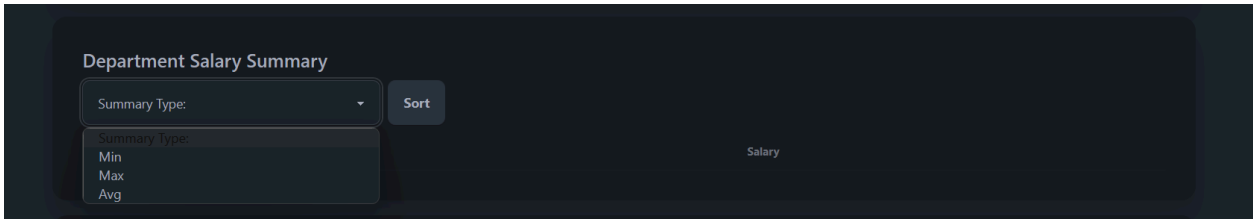

**F2: Salary Overview (Admin)**
**Endpoint:** /admin_dashboard_view?action=sort_salary&sort_by=<MIN|MAX|AVG>'
**Function:** 'admin_dashboard_view'
**Logic:**

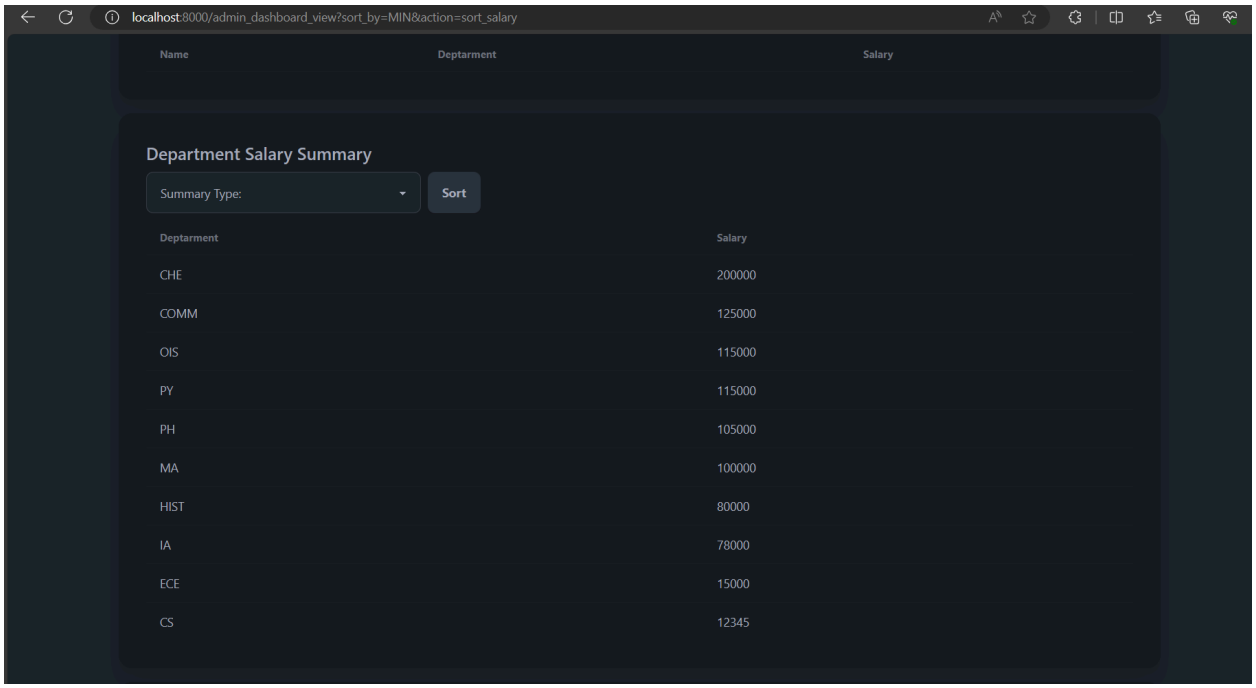
- **Action Recognition:** Detects the 'sort_salary' action based on the query parameters.
- **Parameter Retrieval:** Retrieves the 'sort_by' parameter to determine the type of salary aggregation (Min, Max, Avg).
- **Database Querying:** Uses the 'annotate()' function combined with Min, Max, or Avg to aggregate salaries by department.
- **Data Processing:** Aggregated data is sorted in descending order to prioritize higher values for display using 'instructors.order_by('-salary').
- **Rendering:** The aggregated and sorted salary data is displayed using the 'admin_dashboard.html' template.
- **Code Snippet:**

```
elif action == 'sort_salary':
    sort_by = request.GET.get('sort_by')
    if sort_by == 'MIN':
        instructors = Instructor.objects.values('dept_name').annotate(salary=Min('salary'))
    elif sort_by == 'MAX':
        instructors = Instructor.objects.values('dept_name').annotate(salary=Max('salary'))
    elif sort_by == 'AVG':
        instructors = Instructor.objects.values('dept_name').annotate(salary=Avg('salary'))
```

```
instructors = instructors.order_by('-salary')
context = {'salaries': instructors}
template = loader.get_template('dbApp/admin_dashboard.html')
return HttpResponse(template.render(context, request))
```

- **Preview:**



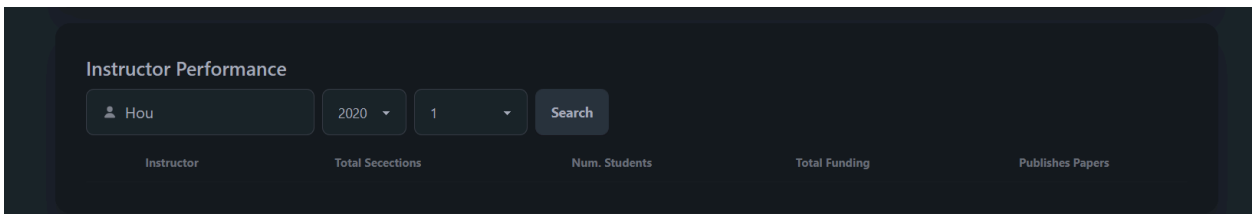

**F3: Professor Performance (Admin)**
**Endpoint:**
'/admin_dashboard_view?action=performance&instructor=<name>&semester=<semester>&year=<year>'
**Function:** 'admin_dashboard_view'
**Logic:**

- **Action Recognition:** The function recognizes the 'performance' action from the GET request.
- **Parameter Retrieval:** Fetches parameters for the instructor name, semester, and year from the GET request.

- **Database Querying:** Uses ‘instructor = Instructor.objects.get(name=instructor.name)’ to query the database and retrieve the instructor object based on the GET parameter. Queries multiple models such as ‘Teaches’, ‘Takes’, ‘Funding’, and ‘Papers’ to gather a comprehensive set of performance data.
- **Data Processing:** Compiles the performance data including instructor’s name, course count, enrollment count, total funding, and total papers in a list ‘instructor_performance’.
- **Rendering:** The compiled performance data is rendered using the ‘admin_dashboard.html’ template.
- **Code Snippet:**

```
elif action == 'performance':
    try:
        instructor_name = request.GET.get('instructor')
        instructor_semester = request.GET.get('semester')
        instructor_year = request.GET.get('year')
        print(instructor_name, instructor_semester, instructor_year)
        instructor = Instructor.objects.get(name=instructor_name)
        instructor_id = instructor.id
        instructor_performance = []
        # Query to get courses taught by the instructor in a specific semester and year
        courses_taught = Teaches.objects.values('teacher_id', 'course_id', 'sec_id', 'semester', 'year')
        courses_taught_filter = courses_taught.filter(teacher_id=instructor_id, semester=instructor_semester, year=instructor_year)
        courses_taught_filter_count = courses_taught.filter(teacher_id=instructor_id, semester=instructor_semester, year=instructor_year).count()
        enrollment_count = []
        for course in courses_taught_filter:
            enrollment_count = Takes.objects.filter(course_id=course['course_id'], sec_id=course['sec_id'], semester=course['semester'], year=course['year']).count()

        instructor_funding = Funding.objects.values('name', 'semester', 'funding', 'year')
        instructor_funding_filter = instructor_funding.filter(name=instructor_name, year=instructor_year, semester = instructor_semester)
        instructor_funding_sum = instructor_funding_filter.aggregate(total_funding=Sum('funding'))
        total_funding_sum = instructor_funding_sum['total_funding'] if instructor_funding_sum['total_funding'] else 0

        instructor_papers = Papers.objects.values('name', 'semester', 'papers', 'years')
        print(instructor_papers)
        instructor_papers_filter = instructor_papers.filter(name=instructor_name, years=instructor_year, semester = instructor_semester)
        print(instructor_papers_filter)
        instructor_papers_sum = instructor_papers_filter.aggregate(total_papers=Sum('papers'))
        papers_count = instructor_papers_sum.get('total_papers', 0)
        instructor_performance.append((instructor.name, courses_taught_filter_count, enrollment_count, total_funding_sum, papers_count))
        print(instructor_performance)
        context = {'instructor_performance': instructor_performance}
        template = loader.get_template('dbApp/admin_dashboard.html')
        return HttpResponse(template.render(context, request))
    except Instructor.DoesNotExist:
        print("Invalid Instructor Input attempt")
        messages.error(request, 'Invalid instructor passed.')
        template = loader.get_template('dbApp/admin_dashboard.html')
        context = {'error': True}
        return HttpResponse(template.render(context, request))
```

- **Preview:**



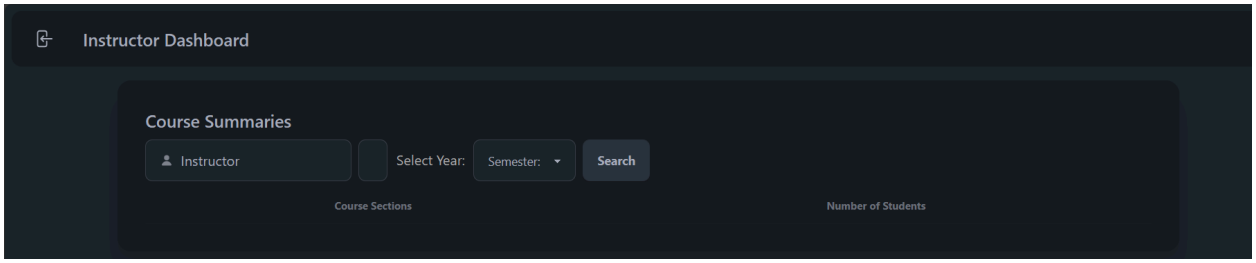

**F4: Course Summary (Professor)**

- **Endpoints:**
  ‘/instructor_dashboard_view?action=course_summary&instructor=<name>&semester=<semester>&year=<year>’

- **Function:** ‘instructor_dashboard_view’
- **Logic:**
- **Action Recognition:** The function identifies the ‘course_summary’ action from the GET request.
- **Parameter Retrieval:** It fetches the value of the parameter ‘instructor’, ‘semester’, and ‘year’ from the GET request.
- **Database Querying:** The function queries the ‘Teaches’ model for values ‘teacher_id’, ‘course_id’, ‘sec_id’, ‘semester’, and ‘year’ and filters to select records where these fields match the GET parameters. This ensures only the courses taught by the instructor in the specified time frame are retrieved.
- **Data Processing:** Constructs a list of courses ‘course_enrollments’ along with the number of students enrolled in each by querying the ‘Takes’ model.
- **Rendering:** Passes the course list with enrollment details to the ‘instructor_dashboard.html’ for display.
- **Code Snippet:**

```python
def instructor_dashboard_view(request):
    if request.method == 'GET':
        if 'action' in request.GET:
            action = request.GET.get('action')
            if action == 'course_summary':
                try:
                    instructor_name = request.GET.get('instructor')
                    semester = request.GET.get('semester')
                    year = request.GET.get('year')

                    # Query to get instructor ID based on name
                    instructor = Instructor.objects.get(name=instructor_name)
                    instructor_id = instructor.id
                    # Query to get courses taught by the instructor in a specific semester and year
                    courses_taught = Teaches.objects.values('teacher_id', 'course_id', 'sec_id', 'semester', 'year')
                    courses_taught_filter = courses_taught.filter(teacher_id=instructor_id, semester=semester, year=year)

                    print(courses_taught_filter)

                    # Query to get student enrollments for each course
                    course_enrollments = []
                    for course in courses_taught_filter:
                        enrollment_count = Takes.objects.filter(course_id=course['course_id'], sec_id=course['sec_id'], semester=course['semester'], year=course['year']).count()
                        course_enrollments.append((f"{course['course_id']} - {course['sec_id']}", enrollment_count))

                    context = {'course_enrollments': course_enrollments}
                    template = loader.get_template('dbApp/instructor_dashboard.html')
                    return HttpResponse(template.render(context, request))
                except Instructor.DoesNotExist:
                    print("Invalid Instructor Input attempt")
                    messages.error(request, 'Invalid instructor passed.')
                    template = loader.get_template('dbApp/instructor_dashboard.html')
                    context = {'error': True}
                    return HttpResponse(template.render(context, request))
```

- **Preview:**

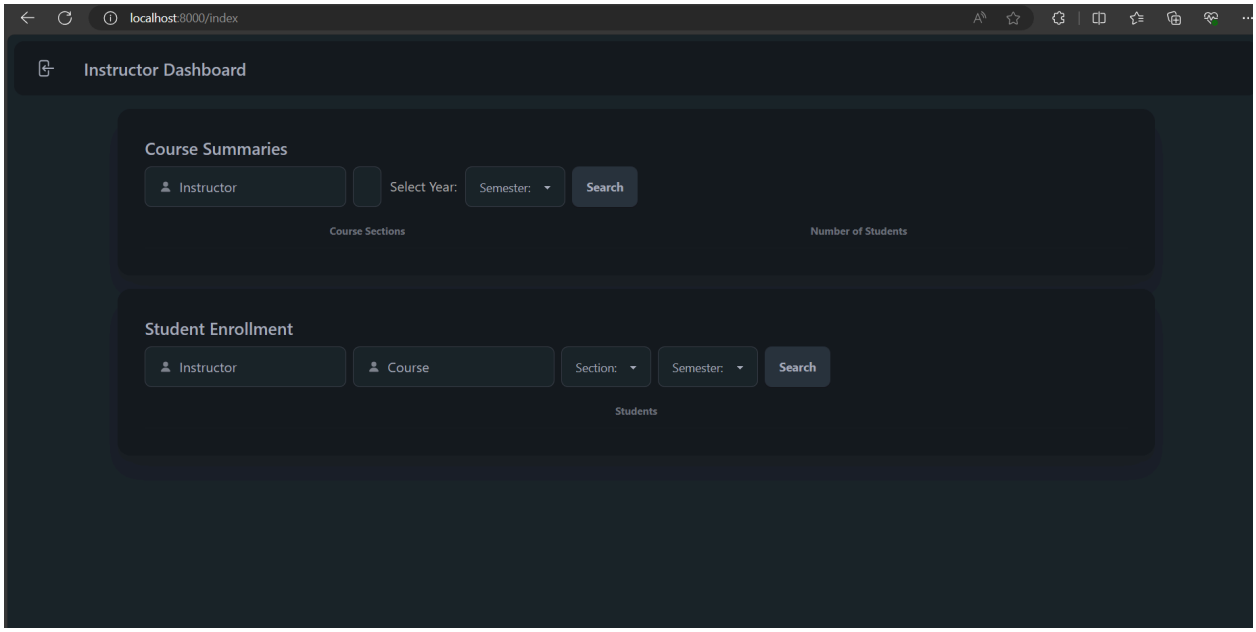


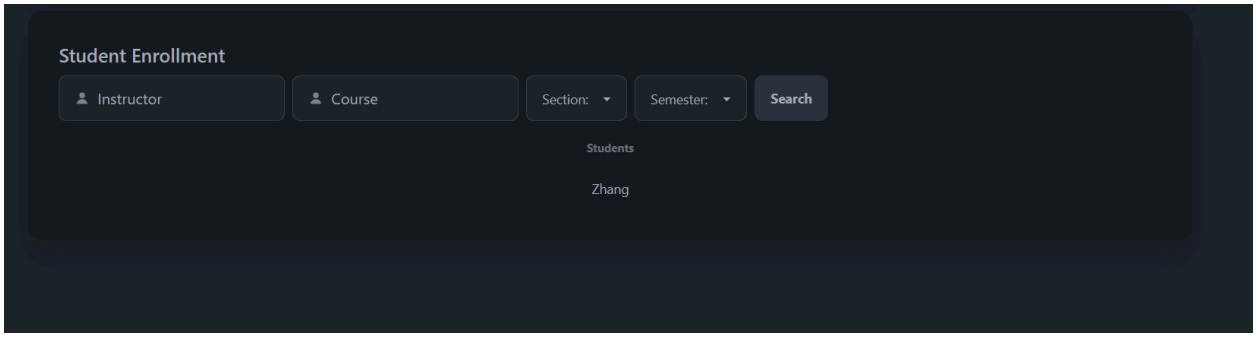
**F5: Student Enrollment Details (Professor)**
**Endpoint:**
'/instructor_dashboard_view?action=student_enrollment&instructor=<name>&course=<course_id>&section=<sec_id>&semester=<semester>&year=<year>'
**Function:** 'instructor_dashboard_view'

- **Action Recognition:** Detects the 'student_enrollment' action from the query parameters.
- **Parameter Retrieval:** Retrieves 'instructor', 'course', 'section', 'semester', and 'year' parameters from the GET request.
- **Database Querying:** Queries the database to get the instructor id based on the name provided in the GET request using 'instructor = Instructor.objects.get(name=instructors_name)'. Queries the 'Teaches' model to retrieve all records using '.values()' to specify the fields to retrieve from the database. Uses '.filter()' to filter the query set of courses taught to only include those taught by the instructor with the specified ID ('instructor_id'), in the specified semester and year requested by the GET parameters.
- **Data Processing:** Compiles a list of enrolled students 'student_enrollments' to store the enrollment data.
- **Rendering:** Sends this list to the 'instructor_dashboard.html' for rendering.
- **Code Snippet:**

```
elif action == 'student_enrollment':
    try:
        instructor_name = request.GET.get('instructor')
        course_semester = request.GET.get('semester')
        course_section = request.GET.get('section')
        course = request.GET.get('course')
        # Query to get instructor ID based on name
        instructor = Instructor.objects.get(name=instructor_name)
        instructor_id = instructor.id
        # Query to get courses taught by the instructor in a specific semester and year
        students_taught = Takes.objects.values('student_id', 'course_id', 'sec_id', 'semester', 'year')
        students_taught_filter = students_taught.filter(course_id = course, sec_id = course_section, semester = course_semester)
        student_ids = [student['student_id'] for student in students_taught_filter]
        students_names = Students.objects.values('name', 'student_id')
        students_names_filter = students_names.filter(student_id__in=student_ids)
        print(students_names_filter)
        # You can further process students_names_filter if needed

        context = {'students_names_filter': students_names_filter}
        template = loader.get_template('dbApp/instructor_dashboard.html')
        return HttpResponse(template.render(context, request))
    except (Instructor.DoesNotExist, Takes.DoesNotExist):
        print("Invalid Instructor Input attempt")
        messages.error(request, 'Invalid instructor passed.')
        template = loader.get_template('dbApp/instructor_dashboard.html')
        context = {'error': True}
        return HttpResponse(template.render(context, request))
```

- **Preview:**



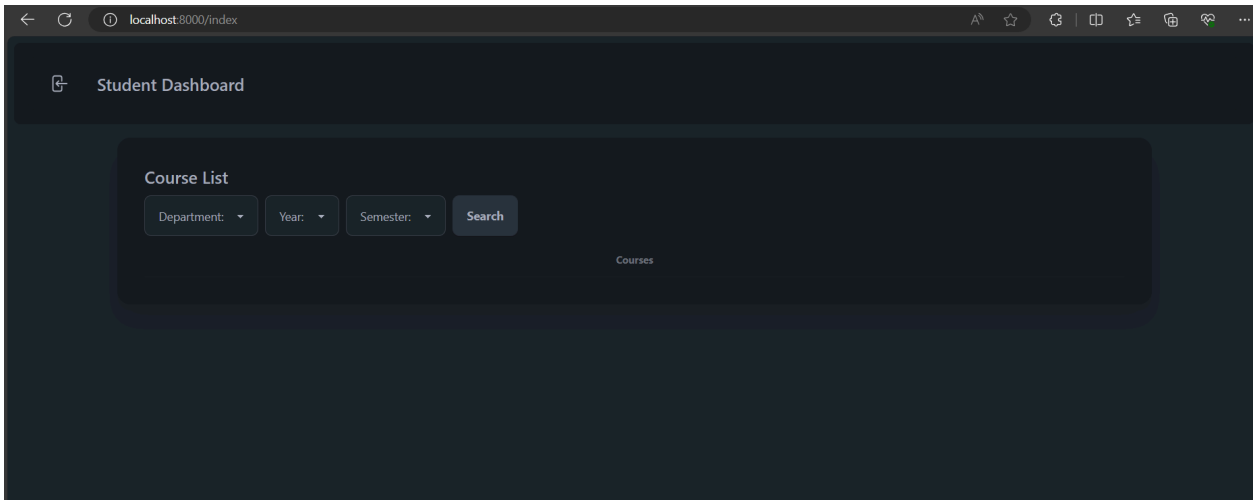




**F6: Course Query (Student)**

- **Endpoint:** ‘/student_dashboard_view?dept=<dept_name>&year=<year>&semester=<semester>’
- **Function:** ‘student_dashboard_view’
- **Logic:**
- **Action Recognition:** Identifies the action to query course sections based on department, year, and semester.
- **Parameter Retrieval:** Extracts ‘dept’, ‘year’, and ‘semester’ from the GET request.
- **Database Querying:** Looks up all sections in the specified semester and year using ‘all_sections = Section.objects.filter(semester=semester, year=year)’.
- **Data Processing:** Filters these sections based on whether the department name ‘dept_name’ is contained within the ‘course_id’ of each section.
- **Rendering:** Creates a context dictionary containing the filtered list of course IDs using ‘context = {‘course_list’:filtered_sections}’. Renderes the ‘student_dashboard.html’ template with this context.
- **Code Snippet:**

```python
def student_dashboard_view(request):
    if request.method == 'GET':
        dept_name = request.GET.get('dept')
        year = request.GET.get('year')
        semester = request.GET.get('semester')
        print(dept_name, year, semester)
        all_sections = Section.objects.filter(semester=semester, year=year)

        # Filter sections in Python code based on course_id containing dept_name
        filtered_sections = [section.course_id for section in all_sections if dept_name in section.course_id]
        print(filtered_sections)
        template = loader.get_template('dbApp/student_dashboard.html')
        context = {'course_list': filtered_sections}
        return HttpResponse(template.render(context, request))
    else:
        template = loader.get_template('dbApp/student_dashboard.html')
        context = {}
        return HttpResponse(template.render(context, request))
```

- **Preview:**




## Database Models:

This section outlines the database schema, highlighting the relationships and key attributes of each model involved in the application.

- **Instructor:** Stores information about professors, including their department and salary.
- **Teaches:** Represents the relationship between instructors and the courses they teach.
- **Takes:** Captures the relationship between students and the courses they enroll in.
- **Funding:** Records details about research funding secured by professors.
- **Papers:** Keeps track of academic papers published by instructors.
- **Students:** Contains student details.
- **Section:** Represents individual course offerings, including details about the semester and academic year.

1. **Instructor**

- id (CharField): Unique identifier for each instructor, maximum length of 5 characters.
- name (CharField): Name of the instructor, maximum length of 32 characters.
- dept_name (CharField): Department to which the instructor belongs, maximum length of 32 characters.
- salary (IntegerField): Annual salary of the instructor.
- Meta:
  db_table: 'instructor'

2. **Teaches**

- course_id (CharField): Course identifier, maximum length of 8 characters.
- sec_id (CharField): Section identifier, maximum length of 4 characters.
- semester (IntegerField): Numeric representation of the semester.
- year (IntegerField): Academic year.
- teacher_id (CharField): Instructor's ID, linked to the Instructor model, maximum length of 5 characters.
- Meta:
  db_table: 'teaches'
- unique_together: Ensures uniqueness across course_id, sec_id, semester, year, and teacher_id.

3. **Takes**

- student_id (CharField): Student's identifier, maximum length of 8 characters.
- course_id (CharField): Course identifier, maximum length of 8 characters.
- sec_id (CharField): Section identifier, maximum length of 4 characters.
- semester (IntegerField): Numeric representation of the semester.
- year (IntegerField): Academic year.
- grade (CharField): Grade achieved in the course, maximum length of 2 characters.
- Meta:
  db_table: 'takes'
- unique_together: Ensures uniqueness for each enrollment record.

4. **Students**

- student_id (CharField): Unique identifier for each student, maximum length of 8 characters.
- name (CharField): Name of the student, maximum length of 32 characters.
- dept_name (CharField): Department to which the student belongs, maximum length of 32 characters.
- total_credit (IntegerField): Total credit hours accumulated by the student.
- Meta:
  db_table: 'student'

5. **Papers**

- name (CharField): Name of the instructor associated with the papers, maximum length of 32 characters.
- semester (IntegerField): Numeric representation of the semester.
- papers (IntegerField): Number of papers published.
- years (IntegerField): Academic year.
- Meta:
  db_table: 'papers'

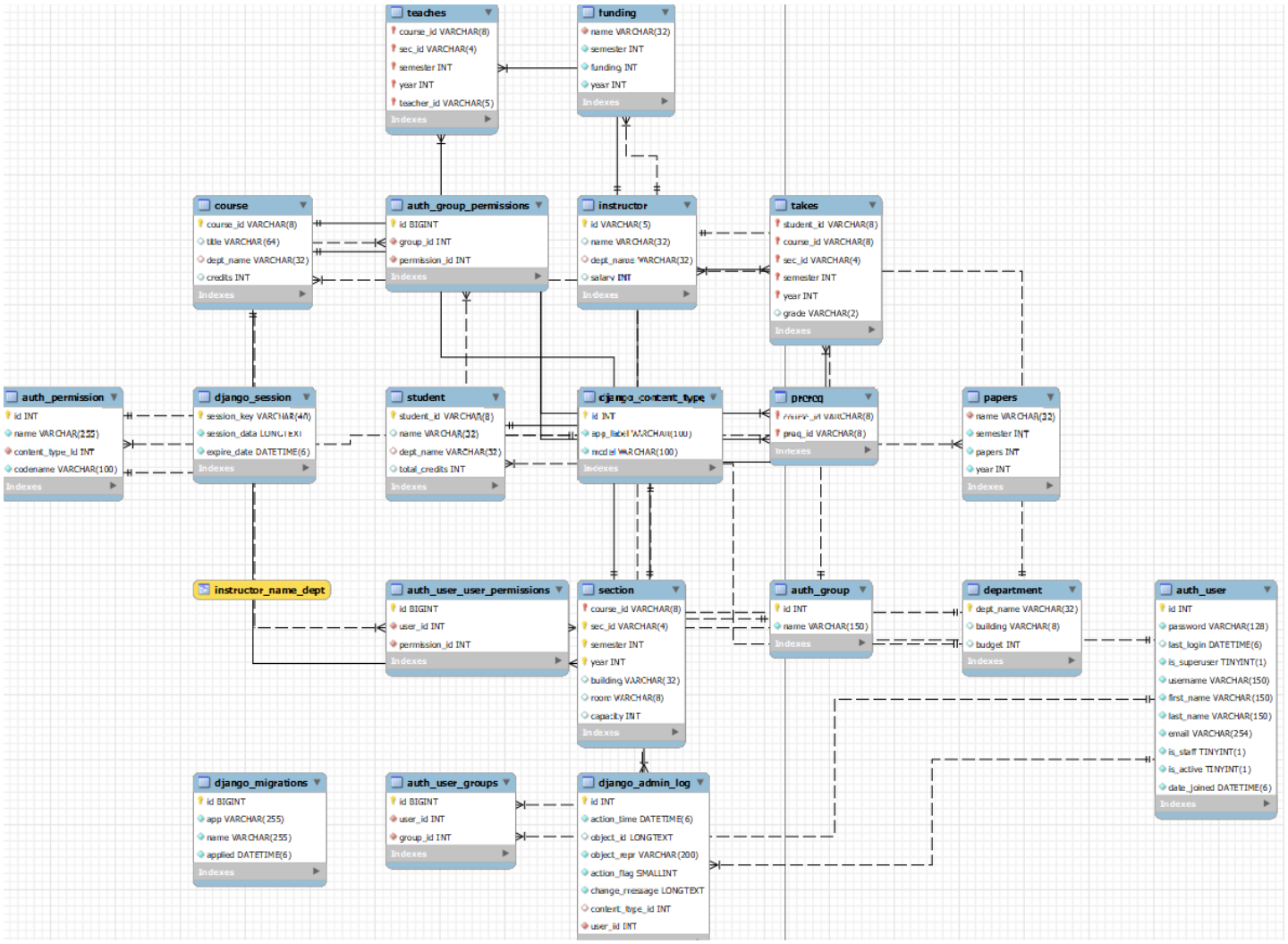
**6. Funding**

- name (CharField): Name of the instructor receiving funding, maximum length of 32 characters.
- semester (IntegerField): Numeric representation of the semester.
- funding (IntegerField): Amount of funding received.
- year (IntegerField): Academic year.
- Meta:
  db_table: 'funding'

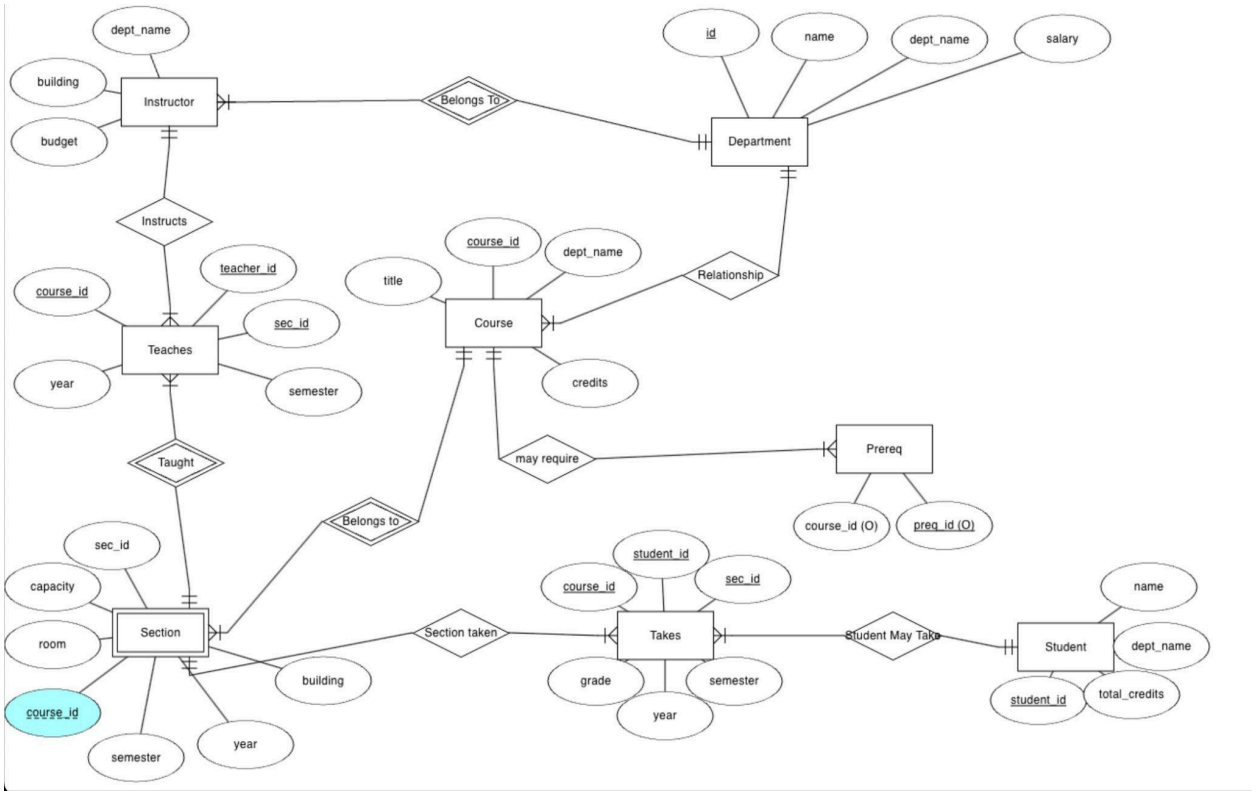
**7. Section**

- course_id (CharField): Course identifier, serves as a primary key, maximum length of 8 characters.
- sec_id (CharField): Section identifier, maximum length of 4 characters.
- semester (IntegerField): Numeric representation of the semester.
- year (IntegerField): Academic year.
- building (CharField): Building where the course section is held, maximum length of 32 characters.
- room (CharField): Room number, maximum length of 8 characters.
- capacity (IntegerField): Maximum number of students the section can accommodate.
- Meta:
  db_table: 'section'

## ER Diagram to Illustrate Database Schema:



### ER Diagram Constructed using ERPlus




Although MySQLWorkbench's reverse engineer feature generates an ER Diagram well enough, we also decided to provide an ER Diagram, constructed using ERPlus, to showcase weak entities, unique/primary attributes, and relationships more clearly.

## Templates and User Interface

This section describes the HTML templates used to render information in the web browser, detailing how the design and user interactions are facilitated. All of the pages utilize DaisyUI, a tailwind CSS component library which streamlined the design of the page

- **Login Page ('login.html'):** Welcome page that allows users to login with provided credentials assuming the database has implemented the django authentication models.
    - **Code Snippet:**

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

```
   <title>Login Page</title>
   <link href="https://cdn.jsdelivr.net/npm/daisyui@4.10.1/dist/full.min.css"
rel="stylesheet" type="text/css" />
   <script src="https://cdn.tailwindcss.com"></script>
 </head>
 <body>
   <div class="w-full flex flex-col place-items-center text-center justify-center">
     <div class="hero min-h-screen" style="background-image:
url(https://www.clarkson.edu/sites/default/files/2024-02/StudentCenterwfireplaceexterior%20
2000x1363.jpg);">
       <div class="hero-overlay bg-opacity-55"></div>
         <div class="w-full flex flex-row h-full">
         <div class="flex-1 flex flex-col justify-center backdrop-blur h-full
overflow-hidden">
           <h1 class="text-4xl font-bold text-left ml-10 p-2 text-white z-10">Welcome to
Clarkson's<br> University Management System</h1>
           <h1 class="text-2xl text-left ml-10 p-2 mr-60 text-white z-10">Admins,
Instructors, and Students have access to up to date records of courses, instructors, grades
and more!</h1>
           <h1 class="text-2xl text-left ml-10 p-2 mr-60 pb-20 text-white z-10">Please
login with your provided credentials or reach out to your Database Admin.</h1>
         </div>
         <div class="hero-content text-center text-neutral-content w-5/12">
           <div class="max-w-lg">
             <div class="card backdrop-blur-sm bg-black/30 shadow-lg transition-all
duration-500 hover:scale-110 hover:shadow-2xl">
               <div class="card-body items-center text-center">
                 <h2 class="card-title pb-2 text-4xl text-white font-bold">Login</h2>
                 <form method="post" action="{% url 'index' %}" class="flex flex-col
gap-2">
                   {% csrf_token %}
                 <label class="input input-bordered flex items-center gap-2 bg-white/30
text-black">
                   <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16"
fill="white/30" class="w-4 h-4 opacity-70"><path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 0
6ZM12.735 14c.618 0 1.093-.561.872-1.139a6.002 6.002 0 0 0-11.215 0c-.22.578.254 1.139.872
1.139h9.47Z" /></svg>
                   <input type="text" name="username" class="grow placeholder-black/30"
placeholder="Username" />
                 </label>
                 <label class="input input-bordered flex items-center gap-2 bg-white/30
```

```
text-black/30">
                    <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16"
fill="white/40" class="w-4 h-4 opacity-70"><path fill-rule="evenodd" d="M14 6a4 4 0 0
1-4.899 3.899l-1.955 1.955a.5.5 0 0 1-.353.146H5v1.5a.5.5 0 0 1-.5.5h-2a.5.5 0 0
1-.5-.5v-2.293a.5.5 0 0 1 .146-.353l3.955-3.955A4 4 0 1 1 14 6Zm-4-2a.75.75 0 0 0 0 1.5.5.5
0 0 1 .5.5.75.75 0 0 0 1.5 0 2 2 0 0 0-2-2Z" clip-rule="evenodd" /></svg>
                    <input type="password" name="password" class="grow placeholder-black/30"
value="password" />
                  </label>
                  <div class="card-actions flex justify-center w-full p-2">
                    <button type="submit" class="btn btn-netural font-bold">Login</button>
                  </div>
                </form>
              </div>
            </div>
            </div>
          </div>
        </div>
        {% if error %}
          <div role="alert" class="alert alert-error absolute bottom-10 w-10/12 mx-10
px-10">
            <svg xmlns="http://www.w3.org/2000/svg" class="stroke-current shrink-0 h-6 w-6"
fill="none" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round"
stroke-width="2" d="M10 14l2-2m0 0l2-2m-2 2l-2-2m2 2l2 2m7-2a9 9 0 11-18 0 9 9 0 0118 0z"
/></svg>
            <span>Invalid Login, please try again.</span>
          </div>
        {% endif %}
      </div>
    </div>
  </body>
</html>
```

- **HTML Structure Description:** The page utilizes a large hero container with a Clarkson background image and then two smaller container ‘div’ elements. One that contains the welcome message on the left and another that contains the login form on the right. The form submission utilizes ‘name’ tags to distinguish the values passed by the form to Django. It also utilizes a csrf_token for form validation with Django. At the bottom of the page, there is also a Django templates if statement that checks if the views.py script returned an error indicating an invalid login attempt which will be discussed in the next section.

- **Admin Dashboard (‘admin_dashboard.html’):** Displays sorted lists of instructors, salary statistics, and professor performance metrics. Includes interactive elements like dropdowns and buttons to trigger different views based on user-selected criteria.
    - **Code Snippet:**

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Admin Dashboard</title>
   <link href="https://cdn.jsdelivr.net/npm/daisyui@4.10.1/dist/full.min.css" rel="stylesheet"
type="text/css" />
   <script src="https://cdn.tailwindcss.com"></script>
 </head>
 <body>
   <div class="flex flex-col w-full place-items-center px-2">
     <div class="navbar w-full bg-base-300 m-2 rounded-xl">
       <a href="{% url 'index' %}">
         <button class="btn btn-ghost text-xl">
           <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-width="1.5"
stroke="currentColor" class="w-6 h-6">
             <path stroke-linecap="round" stroke-linejoin="round" d="M15.75 9V5.25A2.25 2.25 0 0 0
13.5 3h-6a2.25 2.25 0 0 0-2.25 2.25v13.5A2.25 2.25 0 0 0 7.5 21h6a2.25 2.25 0 0 0 2.25-2.25V15M12 9l-3
3m0 0 3 3m-3-3h12.75" />
           </svg>
         </button>
       </a>
       <button class="btn btn-ghost text-xl">Admin Dashboard</button>
     </div>
     <div class="card m-2 w-10/12 bg-base-300 shadow-xl">
       <div class="card-body">
         <h2 class="card-title">All Instructors</h2>
         <form method="get" action="{% url 'admin_dashboard' %}" class="flex gap-2 items-center">
           <select name="sort_by" class="select select-bordered w-full max-w-xs">
             <option disabled selected>Sort By:</option>
             <option value="name">Name</option>
             <option value="dept_name">Department</option>
             <option value="salary">Salary</option>
           </select>
```

```
        <input type="hidden" name="action" value="sort_instructors">
        <button type="submit" class="btn btn-neutral font-bold">Sort</button>
      </form>
      <div class="overflow-x-auto">
        <table class="table">
          <!-- head -->
          <thead>
            <tr>
              <th>Name</th>
              <th>Deptarment</th>
              <th>Salary</th>
            </tr>
          </thead>
          <tbody>
            <!-- row 1 -->
            {% for instructor in instructors %}
            <tr>
                <td>{{ instructor.name }}</td>
                <td>{{ instructor.dept_name }}</td>
                <td>{{ instructor.salary }}</td>
            </tr>
            {% endfor %}
          </tbody>
        </table>
      </div>
    </div>
  </div>
  <div class="card m-2 w-10/12 bg-base-300 shadow-xl">
    <div class="card-body">
      <h2 class="card-title">Department Salary Summary</h2>
      <form method="get" action="{% url 'admin_dashboard' %}" class="flex gap-2 items-center">
        <select name="sort_by" class="select select-bordered w-full max-w-xs">
          <option disabled selected>Summary Type:</option>
          <option value="MIN">Min</option>
          <option value="MAX">Max</option>
          <option value="AVG">Avg</option>
        </select>
        <input type="hidden" name="action" value="sort_salary">
        <button type="submit" class="btn btn-neutral font-bold">Sort</button>
      </form>
      <div class="overflow-x-auto">
```

```
          <table class="table">
            <!-- head -->
            <thead>
              <tr>
                <th>Deptarment</th>
                <th>Salary</th>
              </tr>
            </thead>
            <tbody>
              <!-- row 1 -->
              {% for salary in salaries %}
              <tr>
                  <td>{{ salary.dept_name }}</td>
                  <td>{{ salary.salary }}</td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>
    <div class="card m-2 w-10/12 bg-base-300 shadow-xl">
      <div class="card-body">
        <h2 class="card-title">Instructor Performance</h2>
        <form method="get" action="{% url 'admin_dashboard' %}" class="flex gap-2 items-center">
        <label class="input input-bordered flex items-center gap-2">
          <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16" fill="currentColor" class="w-4
h-4 opacity-70"><path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 6ZM12.735 14c.618 0 1.093-.561.872-1.139a6.002
6.002 0 0 0-11.215 0c-.22.578.254 1.139.872 1.139h9.47Z" /></svg>
          <input type="text" name="instructor" class="grow" value="Instructor" />
        </label>
        <select name="year" class="select select-bordered w-fit max-w-xs">
          <option disabled selected>Year:</option>
          <option>2019</option>
          <option>2020</option>
        </select>
        <select name="semester" class="select select-bordered w-fit max-w-xs">
          <option disabled selected>Semester:</option>
          <option>1</option>
          <option>2</option>
        </select>
```

```
        <input type="hidden" name="action" value="performance">
        <button type="submit" class="btn btn-neutral font-bold">Search</button>
      </form>
        <div class="overflow-x-auto">
          <table class="table text-center">
            <!-- head -->
            <thead>
              <tr>
                <th>Instructor</th>
                <th>Total Sections</th>
                <th>Num. Students</th>
                <th>Total Funding</th>
                <th>Publishes Papers</th>
              </tr>
            </thead>
            <tbody>
              <!-- row 1 -->
              {% for instructor in instructor_performance %}
              <tr>
                <td>{{ instructor.0 }}</td>
                <td>{{ instructor.1 }}</td>
                <td>{{ instructor.2 }}</td>
                <td>{{ instructor.3 }}</td>
                <td>{{ instructor.4 }}</td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>
    {% if error %}
        <div role="alert" class="alert alert-error absolute bottom-10 w-10/12 mx-10 px-10">
          <svg xmlns="http://www.w3.org/2000/svg" class="stroke-current shrink-0 h-6 w-6" fill="none"
viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M10
14l2-2m0 0l2-2m-2 2l-2-2m2 2l2 2m7-2a9 9 0 11-18 0 9 9 0 0118 0z" /></svg>
          <span>Invalid Instructor Name, please try again.</span>
        </div>
      {% endif %}
  </div>
 </body>
```

```
</html>
```

- **HTML Structure Description:** This page utilizes a navbar at the top which has a redirect back to the login page. It has one large ‘flex div’ that's a column and each functionality requirement is its own card on the page. It utilizes ‘value’ tags in ‘select’ tags that allow for increased security which is discussed further in the next slide as well as give the parameter the proper id to pass to the views.py. Each card utilizes an invisible input to dictate which python function to run in views.py. A template for loop is utilized to display the data that is returned from the query. Lastly there is a template if statement at the bottom for if a user inputs an invalid name for the Instructor Name.

- **Instructors Dashboard (‘instructor_dashboard.html’):** Shows courses taught by the professor and lists of enrolled students, providing detailed information relevant to the logged-in professor.
  - **Code Snippet:**

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Instructor Dashboard</title>
   <link href="https://cdn.jsdelivr.net/npm/daisyui@4.10.1/dist/full.min.css" rel="stylesheet"
type="text/css" />
   <script src="https://cdn.tailwindcss.com"></script>
 </head>
 <body>
   <div class="flex flex-col w-full place-items-center px-2">
     <div class="navbar w-full bg-base-300 m-2 rounded-xl">
       <a href="{% url 'index' %}">
         <button class="btn btn-ghost text-xl">
           <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-width="1.5"
stroke="currentColor" class="w-6 h-6">
             <path stroke-linecap="round" stroke-linejoin="round" d="M15.75 9V5.25A2.25 2.25 0 0 0
13.5 3h-6a2.25 2.25 0 0 0-2.25 2.25v13.5A2.25 2.25 0 0 0 7.5 21h6a2.25 2.25 0 0 0 2.25-2.25V15M12
9l-3 3m0 0 3 3m-3-3h12.75" />
           </svg>
         </button>
       </a>
       <button class="btn btn-ghost text-xl">Instructor Dashboard</button>
```

```
    </div>
    <div class="card m-2 w-10/12 bg-base-300 shadow-xl">
      <div class="card-body">
        <h2 class="card-title">Course Summaries</h2>
        <form method="get" action="{% url 'instructor_dashboard' %}" class="flex gap-2
items-center">
        <label class="input input-bordered flex items-center gap-2">
          <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16" fill="currentColor" class="w-4
h-4 opacity-70"><path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 6ZM12.735 14c.618 0
1.093-.561.872-1.139a6.002 6.002 0 0 0-11.215 0c-.22.578.254 1.139.872 1.139h9.47Z" /></svg>
          <input type="text" name="instructor" class="grow" value="Instructor" />
        </label>
        <select name="year" class="select select-bordered w-fit max-w-xs">
          <option disabled selected>Year:</option>
          <option value="2019">2019</option>
          <option value="2020">2020</option>
        </select>
        <select name="semester" class="select select-bordered w-fit max-w-xs">
          <option disabled selected>Semester:</option>
          <option value="1">1</option>
          <option value="2">2</option>
        </select>
        <input type="hidden" name="action" value="course_summary">
        <button type="submit" class="btn btn-neutral font-bold">Search</button>
      </form>
        <div class="overflow-x-auto">
          <table class="table text-center">
            <!-- head -->
            <thead>
              <tr>
                <th>Course Sections</th>
                <th>Number of Students</th>
              </tr>
            </thead>
            <tbody>
              <!-- row 1 -->
              {% for course in course_enrollments %}
              <tr>
                <td>{{ course.0 }}</td> <!-- Course Sections -->
                <td>{{ course.1 }}</td> <!-- Number of Students -->
              </tr>
```

```
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>
    <div class="card m-2 w-10/12 bg-base-300 shadow-xl">
      <div class="card-body">
        <h2 class="card-title">Student Enrollment</h2>
        <form method="get" action="{% url 'instructor_dashboard' %}" class="flex gap-2
items-center">
        <label class="input input-bordered flex items-center gap-2">
          <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16" fill="currentColor" class="w-4
h-4 opacity-70"><path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 6ZM12.735 14c.618 0
1.093-.561.872-1.139a6.002 6.002 0 0 0-11.215 0c-.22.578.254 1.139.872 1.139h9.47Z" /></svg>
          <input type="text" name="instructor" class="grow" value="Instructor" />
        </label>
        <label class="input input-bordered flex items-center gap-2">
          <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 16 16" fill="currentColor" class="w-4
h-4 opacity-70"><path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 6ZM12.735 14c.618 0
1.093-.561.872-1.139a6.002 6.002 0 0 0-11.215 0c-.22.578.254 1.139.872 1.139h9.47Z" /></svg>
          <input type="text" name="course" class="grow" value="Course" />
        </label>
        <select name="section" class="select select-bordered w-fit max-w-xs">
          <option disabled selected>Section:</option>
          <option value="01">01</option>
          <option value="02">02</option>
        </select>
        <select name="semester" class="select select-bordered w-fit max-w-xs">
          <option disabled selected>Semester:</option>
          <option value="1">1</option>
          <option value="2">2</option>
        </select>
        <input type="hidden" name="action" value="student_enrollment">
        <button type="submit" class="btn btn-neutral font-bold">Search</button>
      </form>
        <div class="overflow-x-auto">
          <table class="table text-center">
            <!-- head -->
            <thead>
              <tr>
```

```
                <th>Students</th>
              </tr>
            </thead>
            <tbody>
              <!-- row 1 -->
              {% for student in students_names_filter %}
              <tr>
                <td>{{ student.name }}</td> <!-- Student Enrollment -->
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>
    {% if error %}
        <div role="alert" class="alert alert-error absolute bottom-10 w-10/12 mx-10 px-10">
          <svg xmlns="http://www.w3.org/2000/svg" class="stroke-current shrink-0 h-6 w-6"
fill="none" viewBox="0 0 24 24"><path stroke-linecap="round" stroke-linejoin="round" stroke-width="2"
d="M10 14l2-2m0 0l2-2m-2 2l-2-2m2 2l2 2m7-2a9 9 0 11-18 0 9 9 0 0118 0z" /></svg>
          <span>Invalid Instructor Input, please try again.</span>
        </div>
      {% endif %}
  </div>
 </body>
```

- **HTML Structure Description:** This page utilizes a navbar at the top which has a redirect back to the login page. It has one large ‘flex div’ that's a column and each functionality requirement is its own card on the page. It utilizes ‘value’ tags in ‘select’ tags that allow for increased security which is discussed further in the next slide as well as give the parameter the proper id to pass to the views.py. Each card utilizes an invisible input to dictate which python function to run in views.py. A template for loop is utilized to display the data that is returned from the query. Lastly there is a template if statement at the bottom for if a user inputs an invalid name for the Instructor Name or Course.

- **Student Dashboard (‘student_dashboard.html’):** Allows students to query available courses based on department, year, and semester, displaying a list of relevant course sections.
  - **Code Snippet:**

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Student Dashboard</title>
   <link href="https://cdn.jsdelivr.net/npm/daisyui@4.10.1/dist/full.min.css" rel="stylesheet"
type="text/css" />
   <script src="https://cdn.tailwindcss.com"></script>
 </head>
 <body>
   <div class="flex flex-col w-full place-items-center px-2">
     <div class="navbar w-full bg-base-300 m-2 rounded-lg">
       <div class="navbar w-full bg-base-300 m-2 rounded-xl">
         <a href="{% url 'index' %}">
           <button class="btn btn-ghost text-xl">
             <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
stroke-width="1.5" stroke="currentColor" class="w-6 h-6">
               <path stroke-linecap="round" stroke-linejoin="round" d="M15.75 9V5.25A2.25 2.25 0
0 0 13.5 3h-6a2.25 2.25 0 0 0-2.25 2.25v13.5A2.25 2.25 0 0 0 7.5 21h6a2.25 2.25 0 0 0
2.25-2.25V15M12 9l-3 3m0 0 3 3m-3-3h12.75" />
             </svg>
           </button>
         </a>
         <button class="btn btn-ghost text-xl">Student Dashboard</button>
       </div>
     </div>
     <div class="card m-2 w-10/12 bg-base-300 shadow-xl">
       <div class="card-body">
         <h2 class="card-title">Course List</h2>
         <form method="get" action="{% url 'student_dashboard' %}" class="flex gap-2
items-center">
           <select name="dept" class="select select-bordered w-fit max-w-xs">
             <option disabled selected>Department:</option>
             <option value="CS">CS</option>
             <option value="EE">EE</option>
           </select>
         <select name="year" class="select select-bordered w-fit max-w-xs">
           <option disabled selected>Year:</option>
```

```
          <option value="2019">2019</option>
          <option value="2020">2020</option>
        </select>
        <select name="semester" class="select select-bordered w-fit max-w-xs">
          <option disabled selected>Semester:</option>
          <option value="01">01</option>
          <option value="02">02</option>
        </select>
        <button type="submit" class="btn btn-neutral font-bold">Search</button>
      </form>
        <div class="overflow-x-auto">
          <table class="table text-center">
            <!-- head -->
            <thead>
              <tr>
                <th>Courses</th>
              </tr>
            </thead>
            <tbody>
              <!-- row 1 -->
              {% for course in course_list %}
              <tr>
                <td>{{ course }}</td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
 </body>
</html>
```

- **HTML Structure Description:** This page utilizes a navbar at the top which has a redirect back to the login page. It has one large ‘flex div’ that's a column and each functionality requirement is its own card on the page. It utilizes a select drop down for just EE and CS departments as those were the only departments that had data useful for the student list. This could/should be expanded for full departments.

## Error Handling and Security

Due to the use of Models within our project, we decided to use the built in functions like filter(), values(), etc. to eliminate the possibility of SQL injection. Since we are just pulling the objects from the database and filtering the array we create from there, we do not have to manually inject an SQL query resulting in returning more data than is supposed to be returned or making harmful changes to the database.

**Post Authentication for User Login:**

As mentioned earlier, we also utilize a csrf token to verify source and avoid malicious attacks. Furthermore if no user object is returned (‘user is None’), authentication fails and an error is output to the terminal indicating an invalid login attempt. The application uses Django’s ‘messages’ framework to display an error message to the user.

/views.py

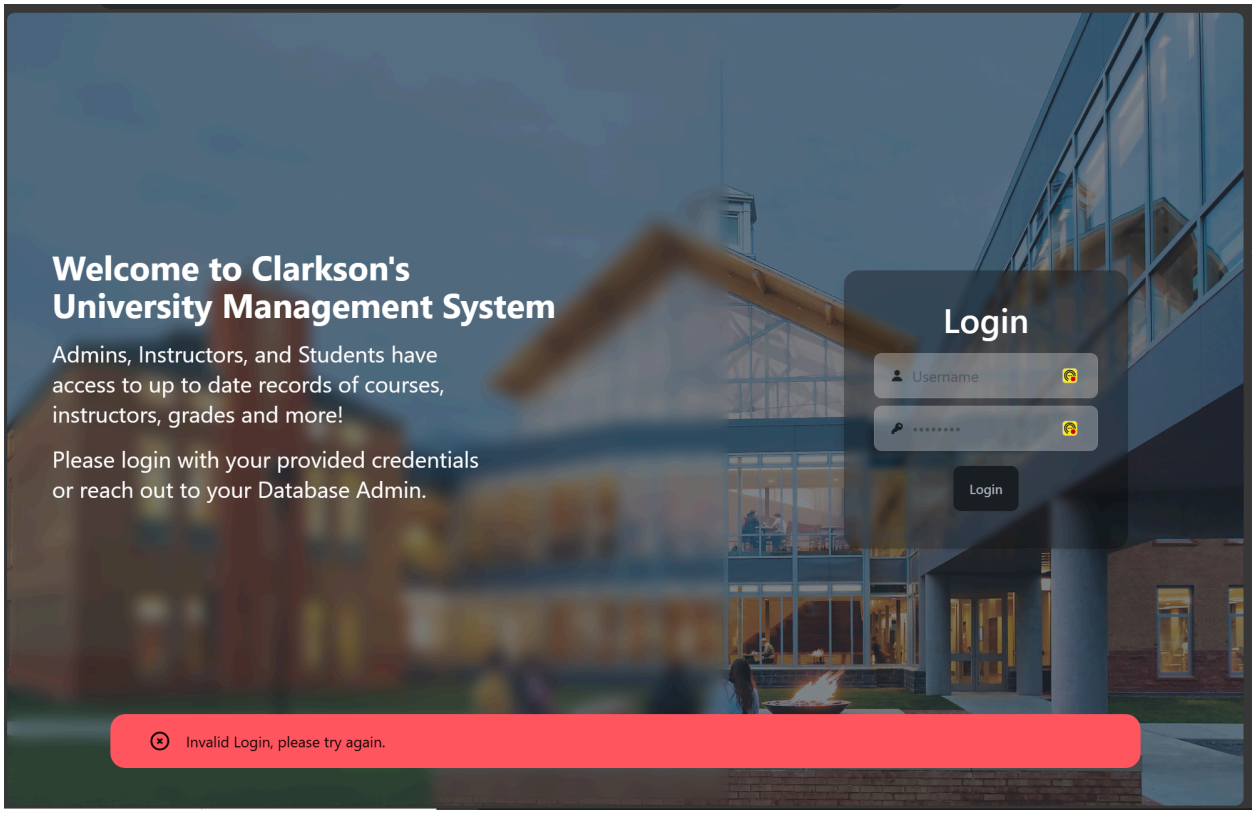
```
else:
    print("Invalid login attempt")
    messages.error(request, 'Invalid username or password.')
    template = loader.get_template('dbApp/login.html')
    context = {'error': True}
    return HttpResponse(template.render(context, request))
```

/templates/login.html

‘Invalid Login, please try again.’ is a text span that contains the error message printed to the interface.

```
{% if error %}
  <div role="alert" class="alert alert-error absolute bottom-10 w-10/12 mx-10 px-10">
    <svg xmlns="http://www.w3.org/2000/svg" class="stroke-current shrink-0 h-6 w-6" fill="none" viewBox="0 0 24 24"><path
    stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M10 14l2-2m0 0l2-2m-2 2l-2-2m2 2l2 2m7-2a9 9 0 11-18 0 9 9 0
    0118 0z" /></svg>
    <span>Invalid Login, please try again.</span>
  </div>
{% endif %}
</div>
</div>
</body>
</html>
```

**Preview:**




**Text Input Error Handling:**
To alleviate potential problems with a teacher or course name not found within a search we have implemented try and except cases within our admin_dashboard and instructor_dashboard functions.

This function by assigning an empty set to the context variable if the action condition is not met by the contents of the GET request. This essentially renders the dashboard without any data. If we wanted to include a specific error message to be displayed to the user we could use the following formatting:

/views.py

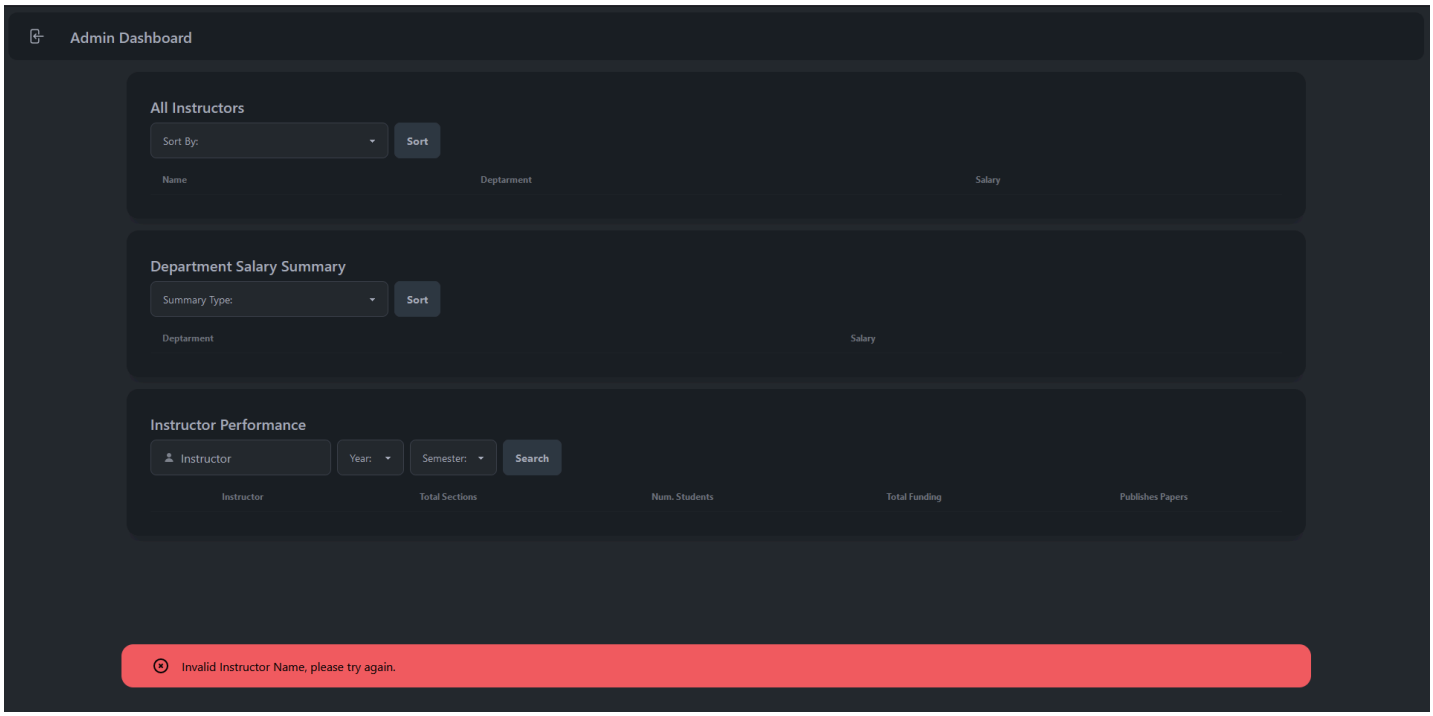
```
def admin_dashboard_view(request):
    if request.method == 'GET':
```

```
        if 'action' in request.GET:
            action = request.GET.get('action')
            if action == 'sort_instructors':
                sort_by = request.GET.get('sort_by')
                instructors = Instructor.objects.all().order_by(sort_by)
                context = {'instructors': instructors}
                template = loader.get_template('dbApp/admin_dashboard.html')
                return HttpResponse(template.render(context, request))
            elif action == 'sort_salary':
                sort_by = request.GET.get('sort_by')
                if sort_by == 'MIN':
                    instructors =
Instructor.objects.values('dept_name').annotate(salary=Min('salary'))
                elif sort_by == 'MAX':
                    instructors =
Instructor.objects.values('dept_name').annotate(salary=Max('salary'))
                elif sort_by == 'AVG':
                    instructors =
Instructor.objects.values('dept_name').annotate(salary=Avg('salary'))
                else:
                    # Default behavior if sort_by is not recognized
                    instructors = Instructor.objects.all()

                instructors = instructors.order_by('-salary')
                context = {'salaries': instructors}
                template = loader.get_template('dbApp/admin_dashboard.html')
                return HttpResponse(template.render(context, request))
            elif action == 'performance':
                try:
                    instructor_name = request.GET.get('instructor')
                    instructor_semester = request.GET.get('semester')
                    instructor_year = request.GET.get('year')
                    print(instructor_name, instructor_semester, instructor_year)
                    instructor = Instructor.objects.get(name=instructor_name)
                    instructor_id = instructor.id
                    instructor_performance = []
                    # Query to get courses taught by the instructor in a specific semester and
year
                    courses_taught = Teaches.objects.values('teacher_id', 'course_id', 'sec_id',
'semester', 'year')
                    courses_taught_filter = courses_taught.filter(teacher_id=instructor_id,
```

```
semester=instructor_semester, year=instructor_year)
                    courses_taught_filter_count = courses_taught.filter(teacher_id=instructor_id,
semester=instructor_semester, year=instructor_year).count()
                    enrollment_count = []
                    for course in courses_taught_filter:
                        enrollment_count = Takes.objects.filter(course_id=course['course_id'],
sec_id=course['sec_id'], semester=course['semester'], year=course['year']).count()

                    instructor_funding = Funding.objects.values('name', 'semester', 'funding',
'year')
                    instructor_funding_filter = instructor_funding.filter(name=instructor_name,
year=instructor_year, semester = instructor_semester)
                    instructor_funding_sum =
instructor_funding_filter.aggregate(total_funding=Sum('funding'))
                    total_funding_sum = instructor_funding_sum['total_funding'] if
instructor_funding_sum['total_funding'] else 0

                    instructor_papers = Papers.objects.values('name', 'semester', 'papers',
'years')
                    print(instructor_papers)
                    instructor_papers_filter = instructor_papers.filter(name=instructor_name,
years=instructor_year, semester = instructor_semester)
                    print(instructor_papers_filter)
                    instructor_papers_sum =
instructor_papers_filter.aggregate(total_papers=Sum('papers'))
                    papers_count = instructor_papers_sum.get('total_papers', 0)
                    instructor_performance.append((instructor.name, courses_taught_filter_count,
enrollment_count, total_funding_sum, papers_count))
                    print(instructor_performance)
                    context = {'instructor_performance': instructor_performance}
                    template = loader.get_template('dbApp/admin_dashboard.html')
                    return HttpResponse(template.render(context, request))
                except Instructor.DoesNotExist:
                    print("Invalid Instructor Input attempt")
                    messages.error(request, 'Invalid instructor passed.')
                    template = loader.get_template('dbApp/admin_dashboard.html')
                    context = {'error': True}
                    return HttpResponse(template.render(context, request))
else:
       template = loader.get_template('dbApp/admin_dashboard.html')
       context = {}
```

```
        return HttpResponse(template.render(context, request))
```

**Preview:**




/views.py

```
def instructor_dashboard_view(request):
   if request.method == 'GET':
       if 'action' in request.GET:
           action = request.GET.get('action')
           if action == 'course_summary':
               try:
                   instructor_name = request.GET.get('instructor')
                   semester = request.GET.get('semester')
                   year = request.GET.get('year')

                   # Query to get instructor ID based on name
                   instructor = Instructor.objects.get(name=instructor_name)
                   instructor_id = instructor.id
                   # Query to get courses taught by the instructor in a specific semester and
year
                   courses_taught = Teaches.objects.values('teacher_id', 'course_id', 'sec_id',
'semester', 'year')
```

```
                    courses_taught_filter = courses_taught.filter(teacher_id=instructor_id,
semester=semester, year=year)

                    print(courses_taught_filter)

                    # Query to get student enrollments for each course
                    course_enrollments = []
                    for course in courses_taught_filter:
                        enrollment_count = Takes.objects.filter(course_id=course['course_id'],
sec_id=course['sec_id'], semester=course['semester'], year=course['year']).count()
                        course_enrollments.append((f"{course['course_id']} - {course['sec_id']}",
enrollment_count))

                    context = {'course_enrollments': course_enrollments}
                    template = loader.get_template('dbApp/instructor_dashboard.html')
                    return HttpResponse(template.render(context, request))
                except Instructor.DoesNotExist:
                    print("Invalid Instructor Input attempt")
                    messages.error(request, 'Invalid instructor passed.')
                    template = loader.get_template('dbApp/instructor_dashboard.html')
                    context = {'error': True}
                    return HttpResponse(template.render(context, request))
            elif action == 'student_enrollment':
                try:
                    instructor_name = request.GET.get('instructor')
                    course_semester = request.GET.get('semester')
                    course_section = request.GET.get('section')
                    course = request.GET.get('course')
                    # Query to get instructor ID based on name
                    instructor = Instructor.objects.get(name=instructor_name)
                    instructor_id = instructor.id
                    # Query to get courses taught by the instructor in a specific semester and
year
                    students_taught = Takes.objects.values('student_id', 'course_id', 'sec_id',
'semester', 'year')
                    students_taught_filter = students_taught.filter(course_id = course, sec_id =
course_section, semester = course_semester)
                    student_ids = [student['student_id'] for student in students_taught_filter]
                    students_names = Students.objects.values('name', 'student_id')
                    students_names_filter = students_names.filter(student_id__in=student_ids)
                    print(students_names_filter)
```

```
                # You can further process students_names_filter if needed

                context = {'students_names_filter': students_names_filter}
                template = loader.get_template('dbApp/instructor_dashboard.html')
                return HttpResponse(template.render(context, request))
            except (Instructor.DoesNotExist, Takes.DoesNotExist):
                print("Invalid Instructor Input attempt")
                messages.error(request, 'Invalid instructor passed.')
                template = loader.get_template('dbApp/instructor_dashboard.html')
                context = {'error': True}
                return HttpResponse(template.render(context, request))
    else:
        template = loader.get_template('dbApp/instructor_dashboard.html')
        context = {}
        return HttpResponse(template.render(context, request))
```

**Preview:**