CSE370 : Database Systems

Project Report

Project Title : Resource website of bracu

Group No: 1, CSE370 Lab Section :8, Summer 2023		
ID	**Name**	**Contribution**
20301443	Hamim Ibne Nasim	**Frontend (HTML,CSS,JS):** ● Login+registration, ● add Faculty information ● calendar **Backend(Django,Python):** ● Implement search method ● table implementation (add_resources) ● error handling while logging in or registering. ● Handles Migration error ● Rendering to page and variable passing
20301357	Fateha Jannat Printia	**Frontend (HTML,CSS,JS):** ● Home page

		● Logout, ● Add_resource ● Searched show Results ● popup window ● Base page **Backend(Django,Python):** ● Registration and login implementation with authentication ● Handles Migration error ● Query handling and passing to HTML ● Class method for complex data
20201074	Nafiz Iqbal Duke	**Frontend (HTML):** ● Course levels (CSE, MNS, and COD) ● faculty informations ● Course information **Backend(Django,Python):** ● table implementation (Faculty information) ● uploaded course orientation ● Redirecting to page ● Formating data into desired values in dictionary

Table of Contents

Introduction

We, "Team Confused," worked to make the BRAC University Resource Website. Our interface makes it easy for you to move through your educational journey. You can find the tools you need by using filters for course level and department. Explore a wide range of materials, faculty routines, and courses that will help you learn more outside of the classroom. You can help the community by easily uploading academic tools, which encourage collaboration and the sharing of knowledge. Connect students with teacher information through our specialised page, which will make it easier to go for consultation and get help. Your private login gives you a personal way to access learning tools. Newcomers can easily sign up with just a few pieces of information. On the Resource View Page, you can find a lot of tools that students have added. Join us in this new age of empowering students in universities.

Project Features

Homepage:

- Login/Register: Students can register by providing their email, ID, and password.
- Search Button: Students can search for academic resources using various filters.
- Course Level Selection: Students can select their desired course level (e.g., 100, 200, etc.).
- Department Selection: Students can choose their department (e.g., CSE, EEE, etc.).
- Course Sections: Based on the selected course level and department, the available course sections (e.g., CSE110, CSE111) will be displayed on the left side of the homepage.

- Resources Section: In the middle, there will be various resources such as events, sessions, guidelines for competitive programming, therapy, etc.
- Upload Button: Students can upload academic resources by clicking on this button, which will open a form.

Upload Resource Page/Form:

Form Fields: The form will require the following details for resource submission:

- Course Code
- Email
- Department
- Course Level
- PDF/images(where the resource is stored)
- Description (a brief description of the resource)

Faculty Information Page:

Faculty Initial Input: Students can input the faculty's initials to access their information.

Faculty Information Popup: Upon entering the faculty's initials, a popup page will appear with the following details:

- Faculty Name
- Faculty Email
- Contact Information
- Office/Room Number
- Routine (if applicable)

Login Page:

- Login Credentials: Students can log in using their registered id and password.

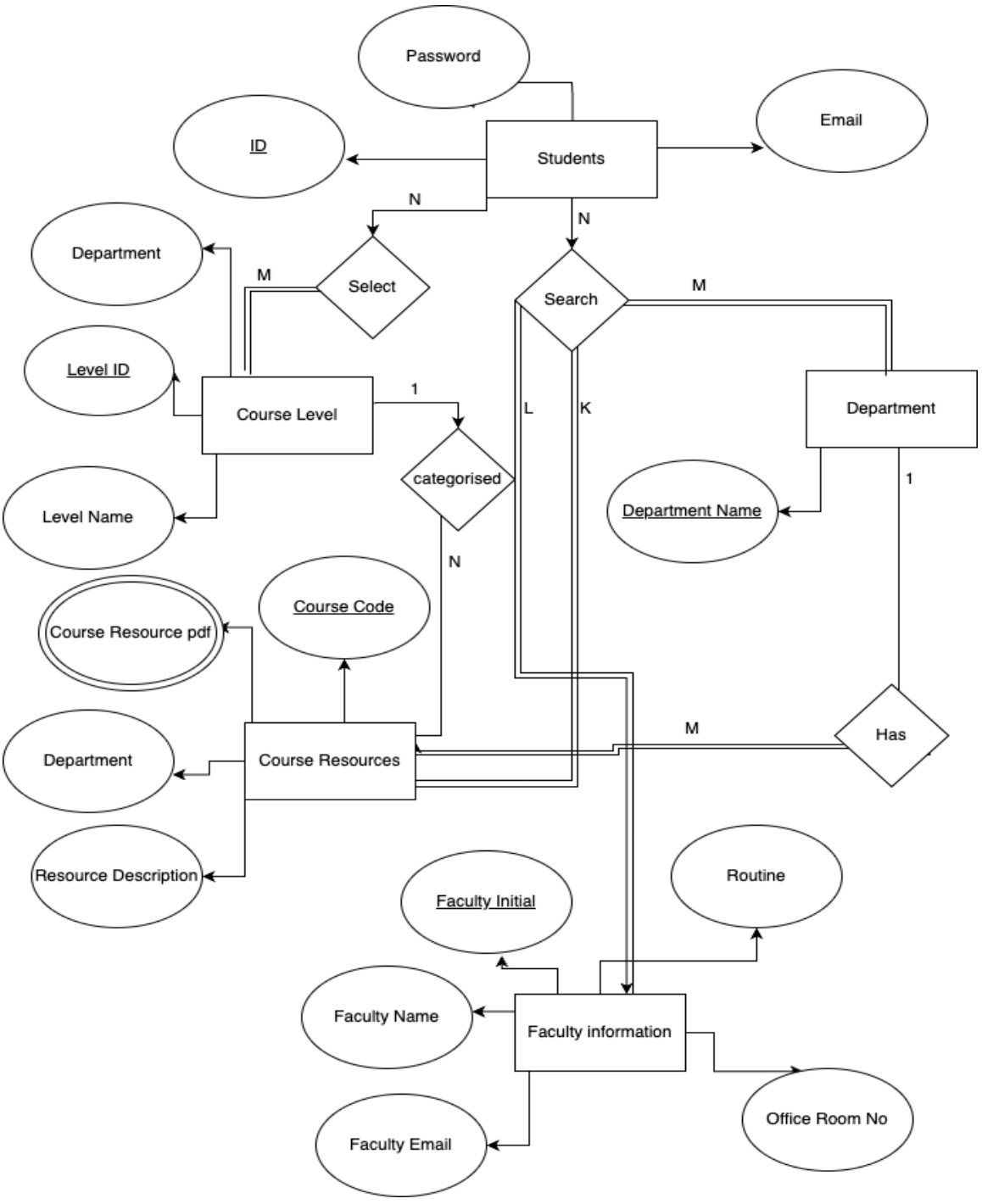
Registration Page:

- Registration Form: Students need to provide their email, ID, and create a password to register.

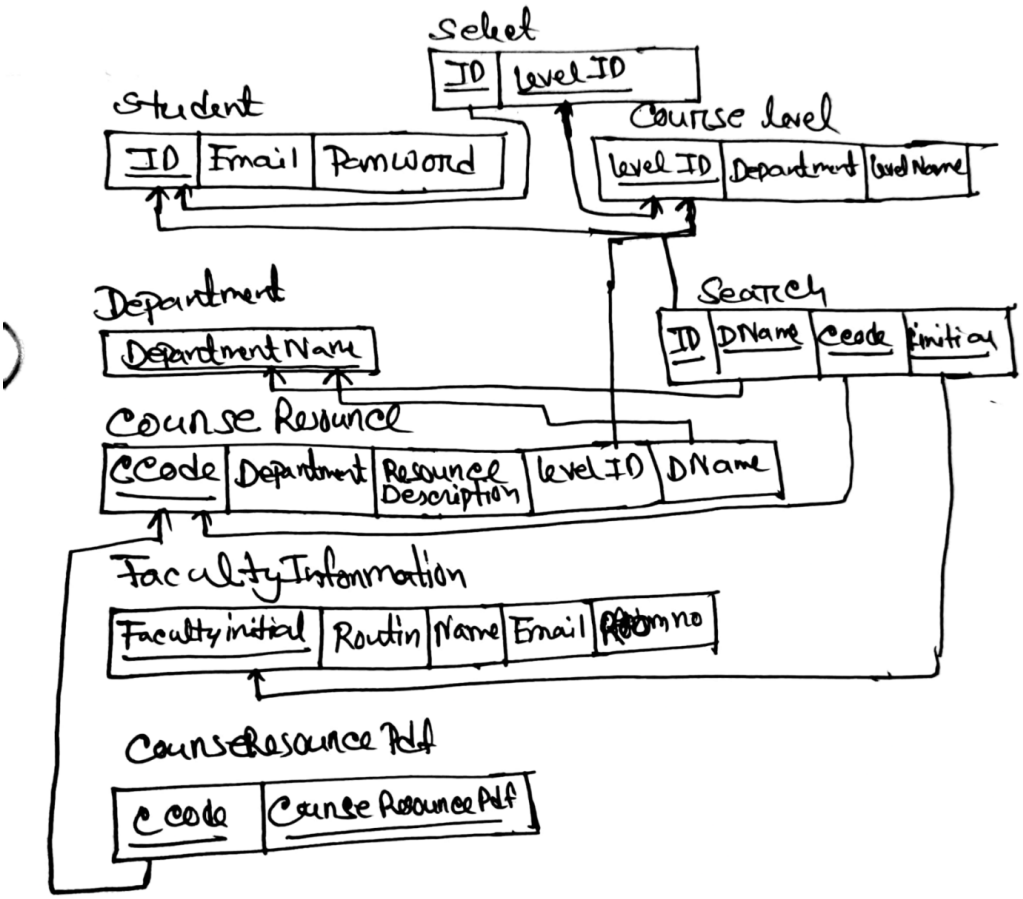
Resource View Page:

- Listed Resources: All the resources uploaded by students will be displayed here, along with the relevant details such as course code, department, etc.

ER/EER Diagram



Schema Diagram



Frontend Development

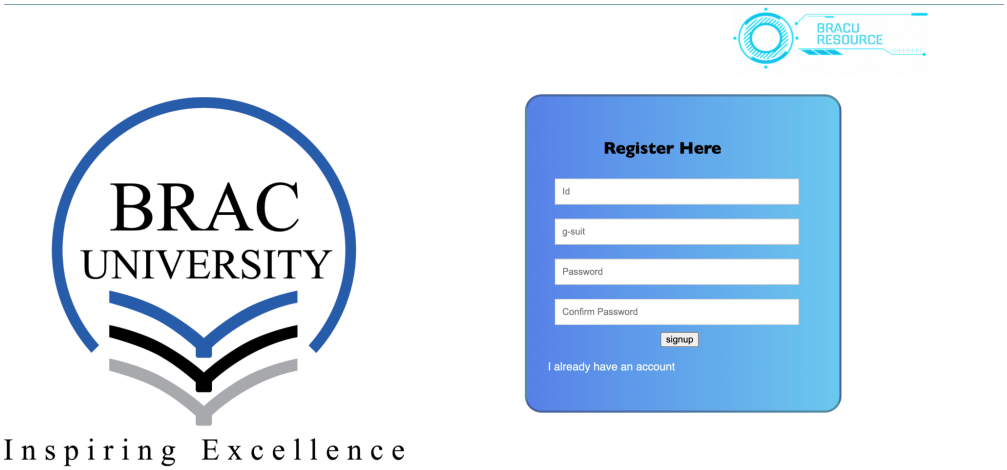
For the frontend, we used HTML, CSS, and JS. We used the form tag to create the registration page. Thus, we used four input boxes, four labels, and a placeholder to show what kind of information the website wants from the user. After registering, it will direct you to the login page via hyperlink. Then there are again two input boxes in a form where you can put the values of your login information. The button colour and gif we used through direct picture CSS gradient, colour, and button, and the forms are imported by HTML tags.

After the authentication is complete in the backend, it will direct you to the home page if the authentication is valid. There is one navbar on every page, which we have done it by Html and css. There is a slidebar where we can find more options, like level, course info, and department. The home page, or base.html, is connected with the home.css file through a link. Thus, we are creating every transaction font and colour by defining every element with a class or id. except that the slide window will show up after clicking the 3 bar icon. That is also done in home.css by displaying after a click. The slide bar has some options like 100 level and 200 level, which are implemented through the <ul> tag's < li> and after clicking those hyperlinks in the li (tag), it will direct you to a page that has all the courses or resources that correspond to that page. The image is going through a loop, and there could be as much as can be stored. In the navbar, there are 4 options. home, add resources, faculties, and logout. Logout will take your authentication and redirect you to the login page that has been implemented in the backend.
Add resources and faculties have the same form, which collects data or input from the user that will be uploaded to the database. except that the form is generated by the Django class. where every input has a particular object variable that can be accessed in the backend. The same goes for faculties forms. The last input on both pages is a file, which is also implemented in the backend. and from the class name, the HTML is showing what type of data it is collecting.

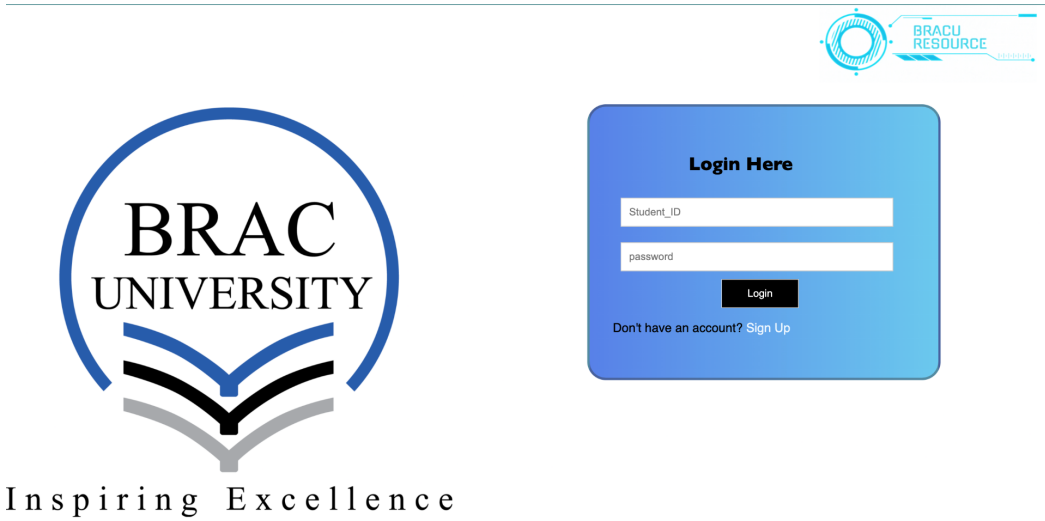
Then there is a pop-up window when the course resource has been shown. This is implemented by JS, as the courses are shown as pictures with the course code. Thus, we use the image as the pop-up button. Thus, we took 3 variables in JS. image, close button, popup display, or which things should be there after the popup? as popup has a class and CSS. The JS function is changing the CSS. When we click the image, JS takes the input and changes

the popup display block so that the popup shows up, and when we click the cross button, the popup display disappears. At the end, there is a search button, which is also an input, and the button is redirecting to the backend as the input has been used in the backend to show results.

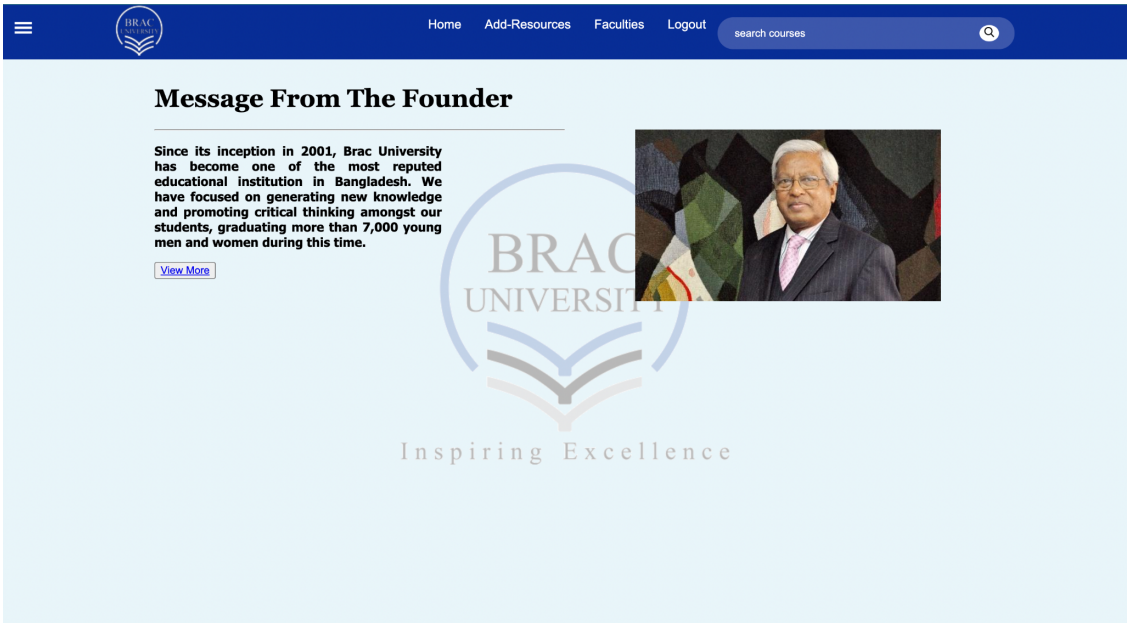
Registration Page



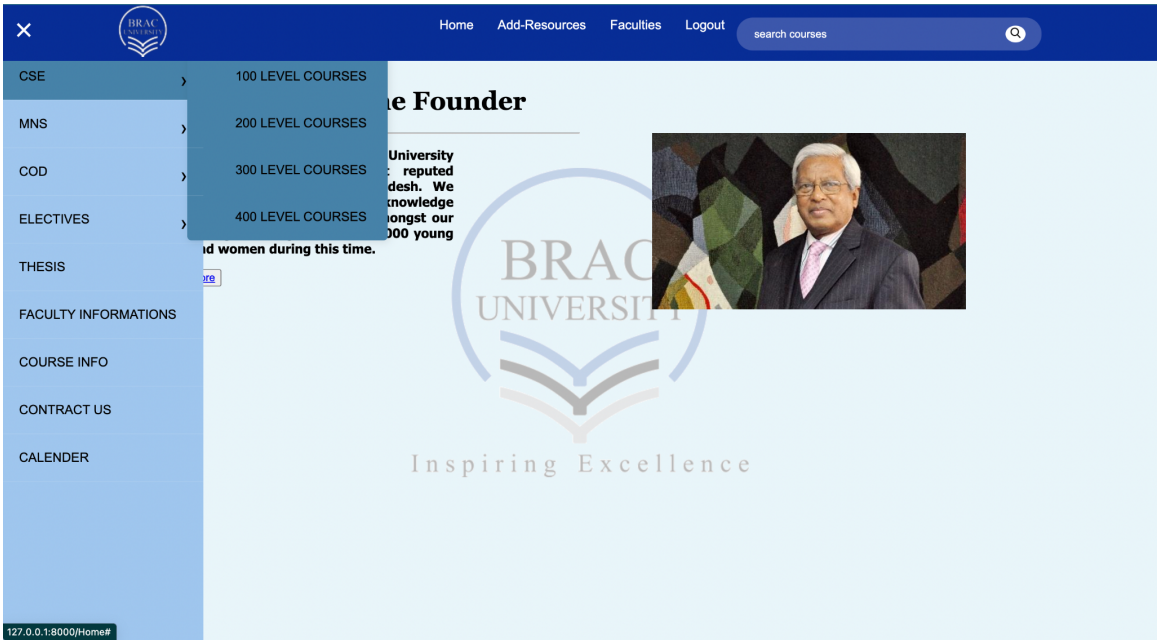
Login Page



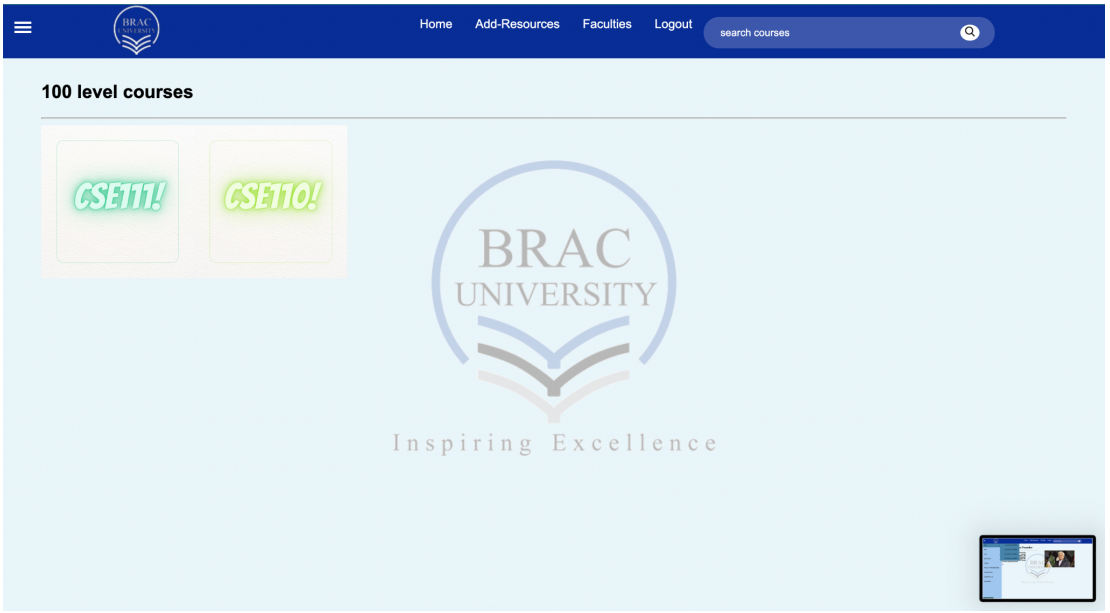
Home Page



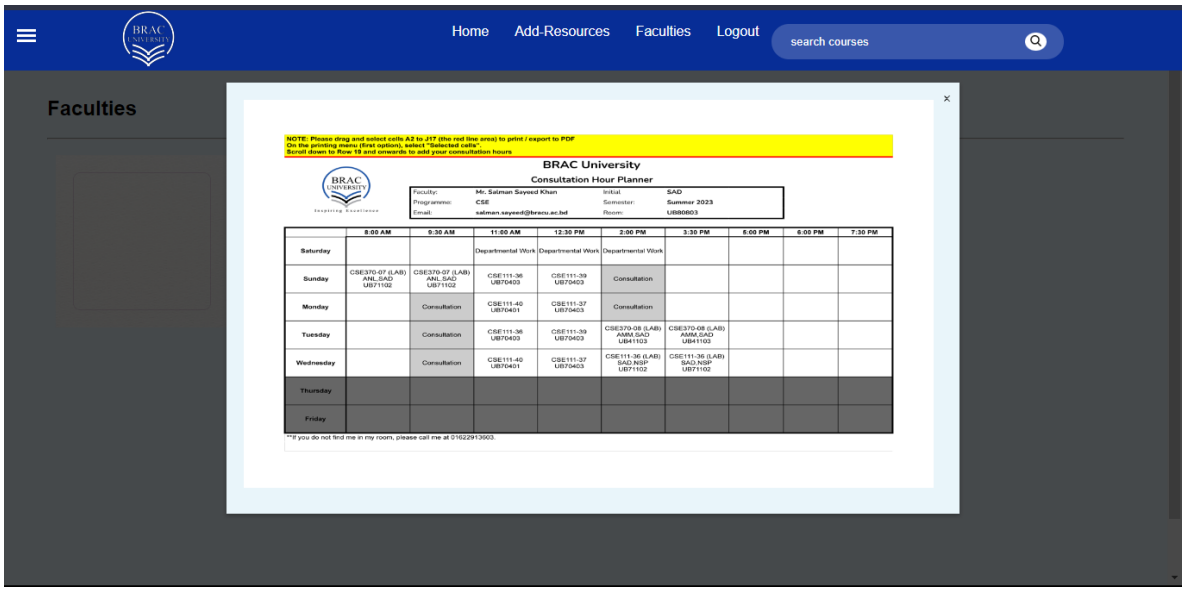
Side Bar



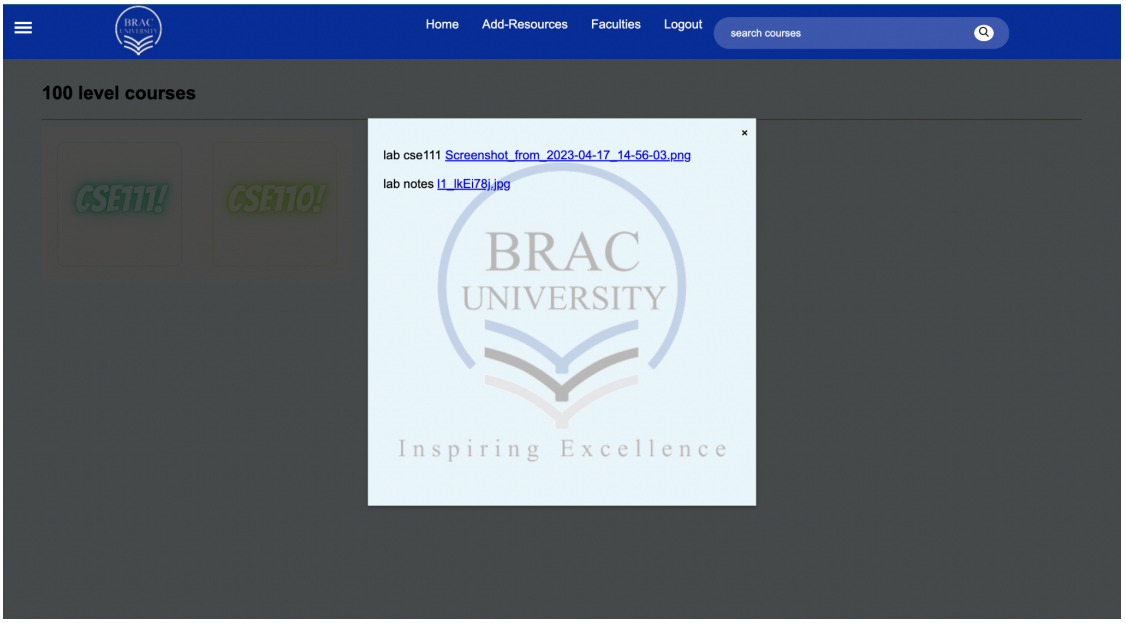
Course Level Page



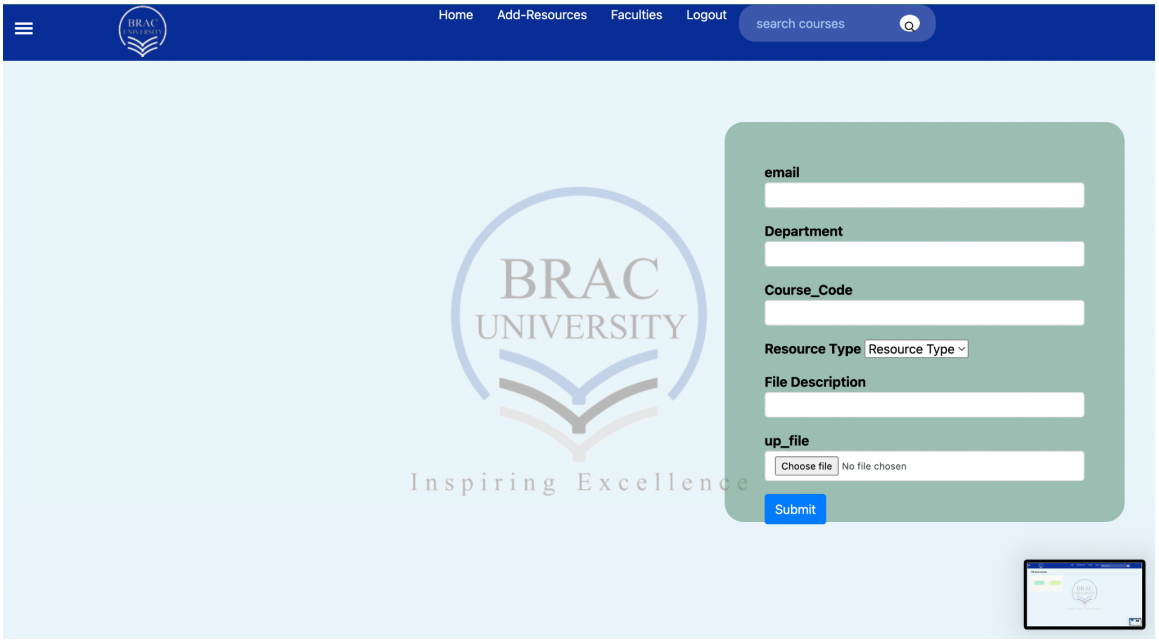
Pop up window for faculty information



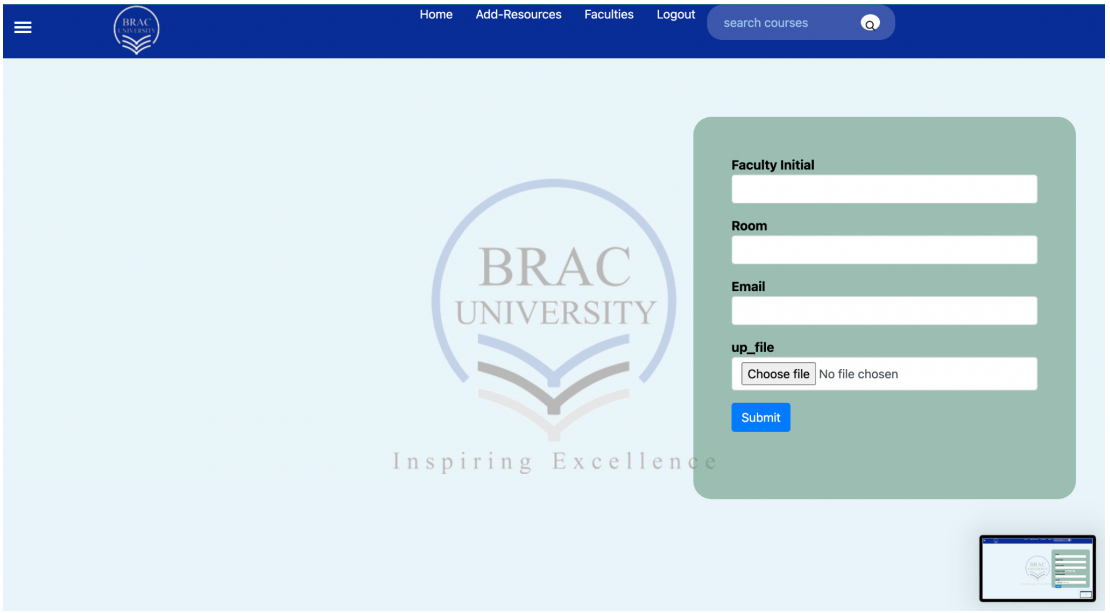
Pop up window for course resources



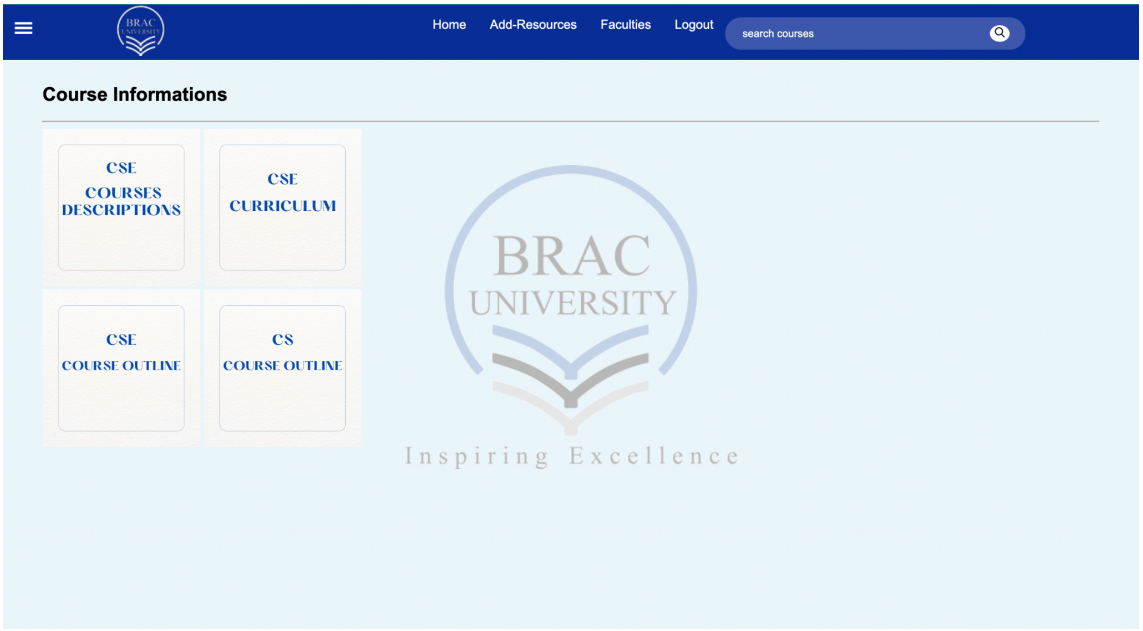
Resource Upload page



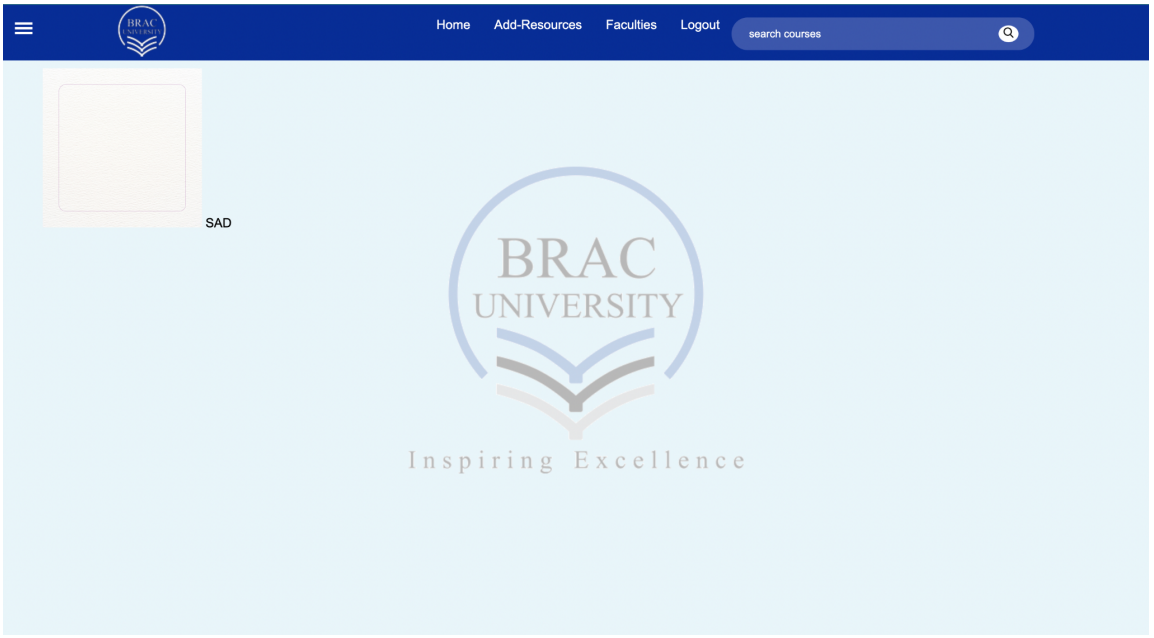
Faculty information upload page



Academic information page



Faculty info page after searching by initial



Backend Development

Contribution of ID : 20301443, Name : Hamim Ibne Nasim

- **Implement search method:**

The search method takes a value by using the query function. which is giving an indication that someone used the search button. This line is entering the condition and taking what is being questioned.

query=request.GET['query']

After getting the query first, it determines if the user is searching for faculty information or not by importing all the values from the faculty table where the search query is present or contained by using icons in the initial column of the database of the faculty. If it is giving any value, it returns a set of queries where the value is present, and it returns the whole set to an HTML file to show up. Otherwise, that will be a query, of course. Thus, it is importing all the course values that contain that particular query string. then it is manipulating the data using dictionary form so that we can show the value in HTML. If the condition is fullfiled, it is redirecting to the search.html page by the render function of Django; otherwise, it will take us to the home page.

```
def searchfunc(request):

    if request.method == 'GET':
        query=request.GET['query']

        facul=faculty.objects.filter(initial__icontains=query)
        if len(facul)>0:
            context = {
            'data':facul
            }
            return render (request,'search2.html',context)
        else:
            course_list = add_resources.objects.filter(Course__icontains=query)
            c = {}
            for i in course_list:
                if i.Course in c:
                    c[i.Course]+=[i]
                else:
                    c[i.Course]=[i]
            print(c)

            context={
                'data':c
            }
            return render (request,'search.html',context)
    return render(request, 'index.html')
```

- **Table implementation (add_resources)**

The table implementation part has two parts: one is database creation, and the other is saving data. The saving of data is part of the backend. When the user clicks the submit button, it is passing a request to the add_resource function in view.py. If the condition is fulfilled, we are capturing the request and passing it to the model class by using this line.

form = UploadFileFrom(request.POST, request.FILES)

We are gathering all the data in the inputs or the form by using cleand_data [variable]. After that, we are passing these values to the class so they can be saved. After save is done and no error happens, it will show a https response uploaded." Otherwise, it will show an error.

```
    context={
        'data':c
    }
    return render (request,'mns1.html',context)
def Add_Resources(request):
    if request.method == 'POST':
        form = UploadFileFrom(request.POST, request.FILES)
        if form.is_valid():
            email = form.cleaned_data['email']
            Department = form.cleaned_data['Department']
            Course = form.cleaned_data['Course']
            texxt = form.cleaned_data['texxt']
            type=form.cleaned_data['type']

            file = form.cleaned_data['file']

            new_resource = add_resources(email=email, Department=Department,type=type,
            new_resource.save()
            return HttpResponse('upload')
        return HttpResponse('error')
    else:
        form = UploadFileFrom()

    return render(request, 'Add_resources.html', {'form': form})
def show_file(request):
    # this for testing
```

- **Error handling while logging in or registering.**

The error handling is shown in the upper screenshots, where we can see that if the forms contain faulty data, it will give an https response warning. but for the primary keys like Student ID for login, Django will show a built-in error as the primary value cannot repeat twice.

IntegrityError at /

UNIQUE constraint failed: auth_user.username

- **Handles Migration error**

The migration error happened when the model got migrated while coding for database connection or there was a mismatch in any kind of redirection or variable name. that has been handled in the terminal and migration files by changing values and modifying the responding function.

- **Rendering to page and variable passing**

Every function is rendering, taking us to a different page. But some pages need data, like after searching the page, which should give us either the values of faculties or the values of courses. Thus, the right data should be passed. that has been done through queries and manipulating particular data structures.

```
    facul=faculty.objects.filter(initial__icontains=query)
    if len(facul)>0:
        context = {
        'data':facul
        }
        return render (request,'search2.html',context)
    else:
        course_list = add_resources.objects.filter(Course__icontains=query)
        c = {}
        for i in course_list:
            if i.Course in c:
                c[i.Course]+=[i]
            else:
                c[i.Course]=[i]
        print(c)

        context={
            'data':c
        }
        return render (request,'search.html',context)
return render(request, 'index.html')
```

Contribution of ID : 20301357, Name : Fateha Jannat Printia

- **Registration and login implementation with authentication**

The registration has 4 inputs and 1 button. The buttons send a request to the function, as the HTML has an action method that connects the function. After the function has been called, it is getting the variable POST," which comes from the HTML response. after the page is checked to ensure the value has been passed perfectly. As we have the request, we are collecting the values that the user has provided by

POST.get('username')

After that, we check to see if the values are faulty or not. We also check if, as it is a student platform, the corresponding ID and G Suite have been given and are valid or not by seeing

certain criteria. If there is any error, the https response shows a message to the user that the data is faulty. Then we are also checking if the password and confirmation password are the same or not. If everything is fine, the value has been passed to

User.objects.create_user #user table

so that we can save them and redirect them to the login page as soon as the registration is successful.

```
def registration(request):
    if request.method=='POST':
        uname=request.POST.get('username')
        email=request.POST.get('mail')
        if "@g.bracu.ac.bd" not in email:
            return HttpResponse("Enter Valid BracU g-suit !!!!")

        if len(uname)!=8:
            return HttpResponse("Enter Valid BracU Student Id!!!!")
        if uname[:1]>'23' or uname[2] not in '123' or uname[3:5] not in ['01','04'] :
            return HttpResponse("Enter Valid BracU Student Id!!")
        pass1=request.POST.get('password1')
        pass2=request.POST.get('password2')

        if pass1!=pass2:
            return HttpResponse("Your password and confrom password are not Same!!")
        else:

            my_user=User.objects.create_user(uname,email,pass1)
            my_user.save()
            return redirect('login')

    return render(request, 'registration.html')
```

- **Handles Migration error**

The migration errors that have been reported come from when one of the tables has already been created and connected via migration. They have been solved by modifying and tracking the function pages of Django, the migration pages, and the user class table. A false migration has also been created to delete the faulty databases and create the correct ones by using **Python Manage. pymigrate --false**

```
class User(AbstractUser):
    objects: UserManager[User]

class AnonymousUser:
    id: Any = ...
    pk: Any = ...
    username: str = ...
    is_staff: bool = ...
    is_active: bool = ...
    is_superuser: bool = ...
    def save(self) -> Any: ...
    def delete(self) -> Any: ...
    def set_password(self, raw_password: str) -> Any: ...
    def check_password(self, raw_password: str) -> Any: ...
    @property
    def groups(self) -> EmptyManager: ...
    @property
    def user_permissions(self) -> EmptyManager: ...
    def get_user_permissions(self, obj: Optional[_AnyUser] = ...) -> Set[str]: ...
    def get_group_permissions(self, obj: Optional[_AnyUser] = ...) -> Set[Any]: ...
    def get_all_permissions(self, obj: Optional[_AnyUser] = ...) -> Set[str]: ...
    def has_perm(self, perm: str, obj: Optional[_AnyUser] = ...) -> bool: ...
    def has_perms(
        self, perm_list: Collection[str], obj: Optional[_AnyUser] = ...
    ) -> bool: ...
    def has_module_perms(self, module: str) -> bool: ...
```

- **Query handling and passing to HTML**

The login page needs proper authentication so that we can authenticate users. Thus, the two inputs of the login page have been fetched through

request.POST.get

After that, the built-in method of Django authenticate has been used and passes the values so that it can match without fetching the passwords for security issues. This function will pass a token, and if the user is found or the password is wrong, it will pass a none. then we are redirecting the user to the home page if a token has been found, or else it will get you back to the login page.

```
        if pass1!=pass2:
            return HttpResponse("Your password and confrom password are not Same!!")
        else:
            my_user=User.objects.create_user(uname,email,pass1)
            my_user.save()
            return redirect('login')

    return render(request, 'registration.html')
def loginUser(request):
    if request.method=="POST":
        username = request.POST.get('username')
        password = request.POST.get('password')
        print(username, password)
        user = authenticate(username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect("/Home")

        else:
            return render(request, 'login.html')

    return render(request, 'login.html')

def logoutUser(request):
    logout(request)
    return redirect("/login")
```

- **Class method for complex data**

The models that have been implemented for the database have some complex values, like

RESOURCE_CHOICES = [
(1, 'Resource Type'),
(2, 'Theory'),
(3, 'Lab'),
]

which indicated choice. This cannot be fetched through an object. The class method has been implemented so that we can get the desired output from HTML code by calling these functions or methods by objects.

```
class add_resources(models.Model):
    RESOURCE_CHOICES = [
        (1, 'Resource Type'),
        (2, 'Theory'),
        (3, 'Lab'),
    ]
    email=models.CharField( max_length=122)
    Department=models.CharField( max_length=12,default='None')
    type = models.IntegerField(choices=RESOURCE_CHOICES, default=1)
    Course=models.CharField(max_length=9,default='Not label')  #file name
    file= models.FileField(upload_to='',default='No file given')  #uploaded file
    texxt=models.CharField( max_length=200)



    def __str__(self):
        return self.Course
    def getType(self):
        if self.type==2:
            return 'Theory'
        else:
            return 'lab'
```

Contribution of ID : 20201074 , Name : Nafiz Iqbal Duke

- **table implementation (Faculty information)**

  The main parts of putting the faculty form table into place are making the database layout and storing the data. The backend code is where the data is stored. When a user clicks the submit button to start the submission process, a request is sent to the view.py file's add_resource method. When certain conditions are met, the request that comes in is caught and then sent to the model class. The following line of code shows how this process works:

  form equals FacultyForm(request.POST, request.FILES).

  The cleaned_data [variable] is used to get the information from the form's input fields or from the form itself. The numbers that were gathered are then sent to the appropriate class to be stored.

```
            return 'Theory'
        else:
            return 'lab'


class faculty(models.Model):
    g_suit=models.CharField( max_length=122)
    initial=models.CharField( max_length=12,default='None')
    room_no=models.CharField(max_length=9,default='NO room assigned')
    file= models.FileField(upload_to='faculty',default='No file given')

    def __str__(self):
        return self.initial
```

- **uploaded course orientation**

The course level needs a certain course that would show up in the desire page as the course has a certain level, like CSE 111, which belongs to the 100-level course of the CSE Department. and we can ensure that if the column contains "cse1" in its values, The following data has been feted by icons:

add_resources.objects.filter(Course__icontains='cse3')

and manupulate it in context so that it can be passed to an HTML file.

```
def cse3(request):
    course_list = add_resources.objects.filter(Course__icontains='cse3')

    c = {}
    for i in course_list:
        if i.Course in c:
            c[i.Course]+=[i]
        else:
            c[i.Course]=[i]
    print(c)
    context={
        'data':c
    }
    return render (request,'cse3.html',context)

def cse4(request):

    course_list = add_resources.objects.filter(Course__icontains='cse4')

    c = {}
    for i in course_list:
        if i.Course in c:
            c[i.Course]+=[i]
        else:
            c[i.Course]=[i]
    print(c)
    context={
        'data':c
    }
    return render (request,'cse4.html',context)
```

- **Redirecting to page**

The urls.py file contains all the redirects for the page.

path('admin/', admin.site.urls),

This path contains three parameters: name, url, and name of the url. where if any function renders any of the urls, it or any HTML asks for access to the action urls.py, which will give the direction of the particular function.

```python
from django.urls import path,include
from home import views
from django.conf import settings
from django.conf.urls.static import static



urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.index, name="home"),
    path('index',views.index, name="home"),
    path('Courses', views.Courses,name='Courses'),
    path('CourseInformation', views.CourseInformation,name='CourseInformation'),
    path('Add_resources', views.Add_Resources,name='Add_resources'),
    path('login',views.loginUser, name="login"),
    path('logout',views.logoutUser, name="logout"),
    path('cse1',views.cse1,name='cse1'),
    path('cse2',views.cse2,name='cse2'),
    path('cse3',views.cse3,name='cse3'),
    path('cse4',views.cse4,name='cse4'),
    path('mns1',views.mns1,name='mns1'),
    path('search',views.searchfunc,name='search'),

    path('view', views.show_file, name="view"),
    path('faculty', views.Faculty, name="faculty"),
    path('fac', views.show_file1, name="fac")
]

urlpatterns += static(settings.STATIC_URL, document_root = settings.STATIC_ROOT)
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Conclusion

In the end, the BRAC University Resource Website made by Team Confused shows how important it is to use new ideas and work together in education. As we start this journey together, we believe that education isn't just something that happens in classrooms; it's a dynamic, interactive process that should be open to everyone and improve their lives. Through our platform, we want to connect students with resources and build a group that thrives on sharing information, ideas, and support. We invite you to embark on a transformative journey of exploration, contribution, and connection. Thank you for joining us on this fun trip where there are endless ways to learn.

References

CodeWithHarry. (2020, April 1). *Django Tutorial In Hindi* [Video]. YouTube.
https://www.youtube.com/watch?v=JxzZxdht-XY