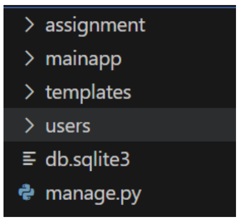
REPORT

BACKEND ASSIGNMENT FOR PROSPACE - **AI-Powered Oral Conversation Learning Management System**

By – Ishan Patni (22B0377)
Date – 25/03/2024

Setting up Django project

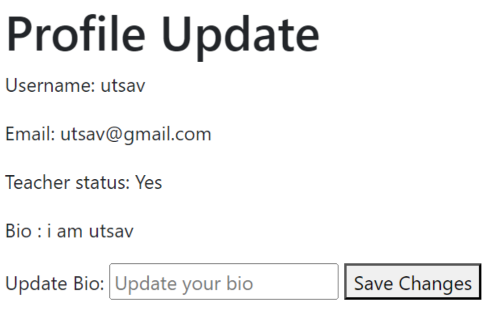
- I have created a project structure which contains main file with name assignment and two app folders – one for users and other the mainapp for other requirements.
- Users app contains code for defining the users and user authentication
- Mainapp contains code for defining other requirements like classes, assignments and questions (as asked for in the assignment)
- I have also created a template folder which contains html templates picked up from internet open source for displaying some of the work front end.



```
> assignment
> mainapp
> templates
> users
db.sqlite3
manage.py
```

Models

1. User Model and authentication
    - I have used default user model provided by django.contrib.auth which helps us to easily create users and also helps in easy authentication using user-id and password.
    - I have then created a custom user model named profile which is one to one mapped to the default user model and also contains ability to generate extra fields based on requirement. For demonstration, in profile model after maping with default user model I have created an extra field like user bio (other fields like profile image, phone number, teacher access status etc can also be created based on requirement)



Profile Update

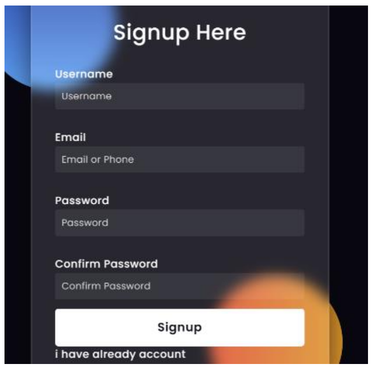
Username: utsav

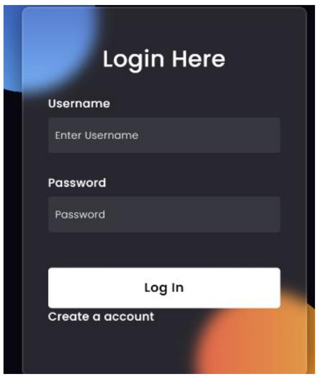
Email: utsav@gmail.com

Teacher status: Yes

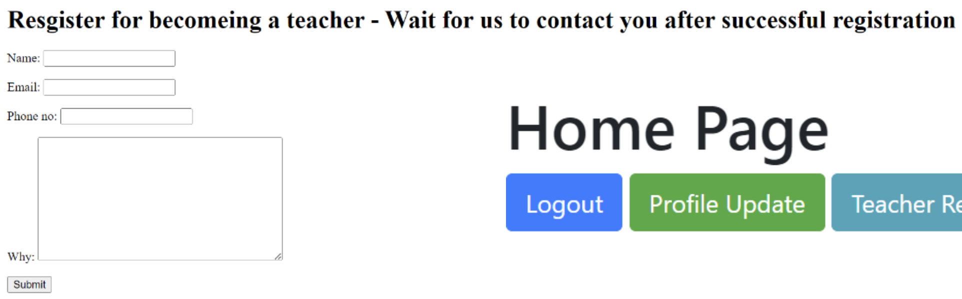
Bio : i am utsav

Update Bio: Update your bio [Save Changes]





- The login signup and login pages have been taken from internet for demonstration purpose
- The method I am using here to distinguish if a person is teacher is or not is a Boolean in his profile model. When user registers the status is "NO" by default but if he wishes to register as a teacher he can fill another form named teacher registration.
- For this I have created another model with name requirements. This helps in asking user his additional info to become a teacher and send it to the admin. Admin can then verify the additional information and change the Boolean manually to "YES" thus giving teacher access to the user which will then reflect on his profile.


Resgister for becomeing a teacher - Wait for us to contact you after successful registration

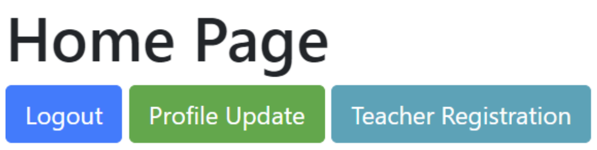
Name:

Email:

Phone no:

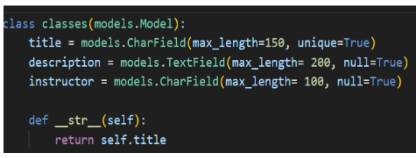
Why:

Submit

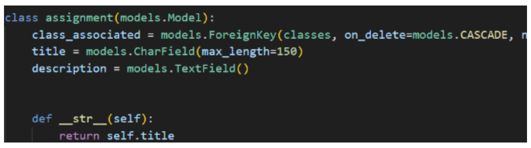


- This is how I have processed the user authentication and efficiently distinguished between user and teacher.

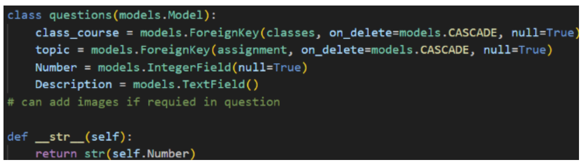
2. Other Models
    - I have created models for classes, assignments and questions as well in other app named main app.
    - These models can have various fields as required, for demonstration purpose I have mainly used fields as titles and descriptions



```python
class classes(models.Model):
    title = models.CharField(max_length=150, unique=True)
    description = models.TextField(max_length= 200, null=True)
    instructor = models.CharField(max_length= 100, null=True)

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



```python
class assignment(models.Model):
    class_associated = models.ForeignKey(classes, on_delete=models.CASCADE, n
    title = models.CharField(max_length=150)
    description = models.TextField()


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

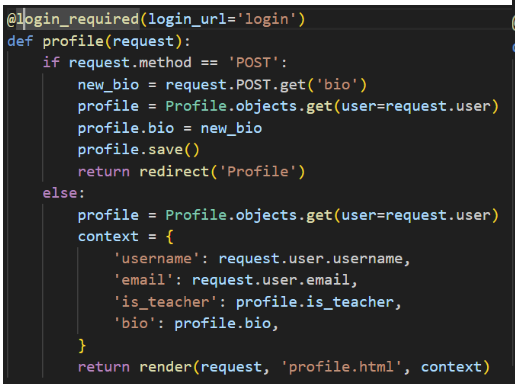


```python
class questions(models.Model):
    class_course = models.ForeignKey(classes, on_delete=models.CASCADE, null=True
    topic = models.ForeignKey(assignment, on_delete=models.CASCADE, null=True)
    Number = models.IntegerField(null=True)
    Description = models.TextField()
# can add images if requied in question

def __str__(self):
    return str(self.Number)
```

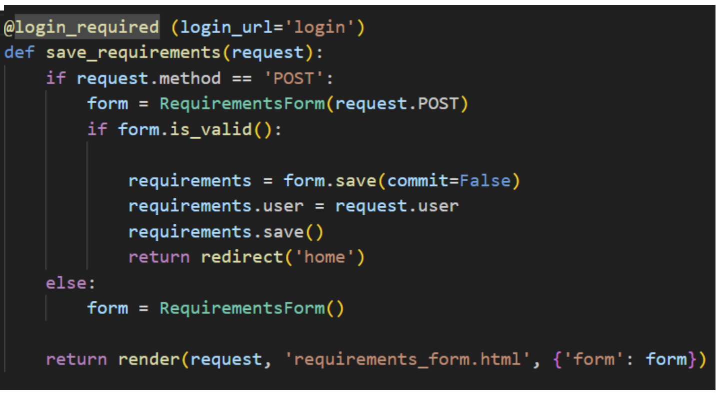
- I have used foreign key (use for one to may kind of relations) to link classes to assignments and assignments to questions because a course can contain multiple assignments and assignment can contain multiple questions.

Views for users

- For user registration I have used condition that whenever a person fill form we use POST methos to collect the data and for a new user. By default Django user model does not allow users with same name thus all users have unique username through which they can login after registration
- For login when person enters the registered username and password we use POST method to fetch it and GET method to get the username and password that exist in database then we authenticate it using the Django default authentication.
- We have used decorators over all other views so that user can not use them without login.
- Again we use conditional POST and GET for user profile. If request method is POST then send the additional added information to database else use GET to display the retrieved existing information from Django database.
- For teacher registration we simply use POST method to send data to the database.



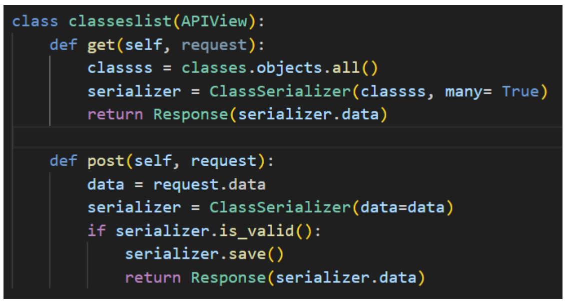
```python
@login_required(login_url='login')
def profile(request):
    if request.method == 'POST':
        new_bio = request.POST.get('bio')
        profile = Profile.objects.get(user=request.user)
        profile.bio = new_bio
        profile.save()
        return redirect('Profile')
    else:
        profile = Profile.objects.get(user=request.user)
        context = {
            'username': request.user.username,
            'email': request.user.email,
            'is_teacher': profile.is_teacher,
            'bio': profile.bio,
        }
        return render(request, 'profile.html', context)
```



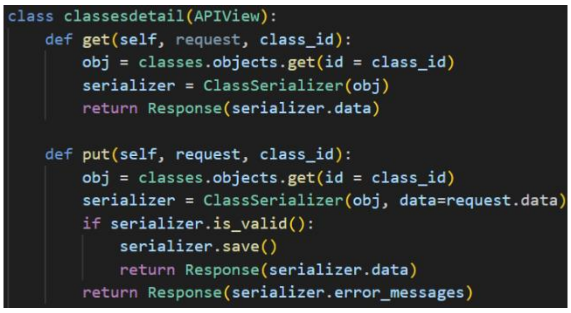
```python
@login_required (login_url='login')
def save_requirements(request):
    if request.method == 'POST':
        form = RequirementsForm(request.POST)
        if form.is_valid():

            requirements = form.save(commit=False)
            requirements.user = request.user
            requirements.save()
            return redirect('home')
    else:
        form = RequirementsForm()

    return render(request, 'requirements_form.html', {'form': form})
```

CRUD operations and API endpoints

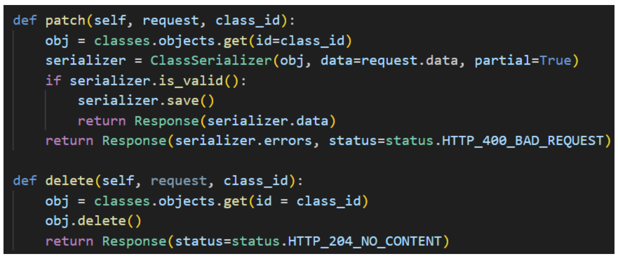
- I have implemented CRUD operations for classes, assignments and questions using Django REST framework. Example -



```
class classeslist(APIView):
    def get(self, request):
        classss = classes.objects.all()
        serializer = ClassSerializer(classss, many= True)
        return Response(serializer.data)

    def post(self, request):
        data = request.data
        serializer = ClassSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
```



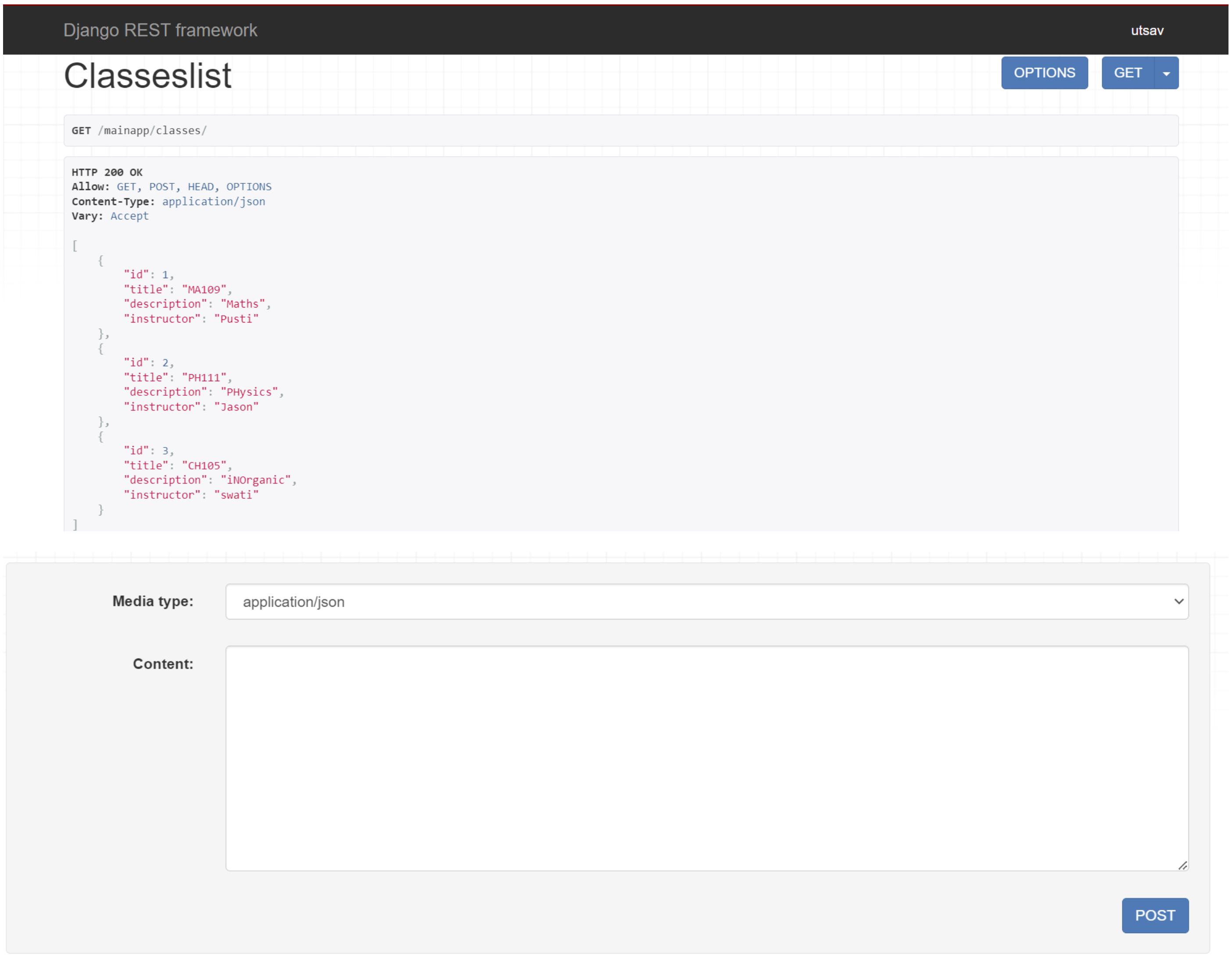
```
class classesdetail(APIView):
    def get(self, request, class_id):
        obj = classes.objects.get(id = class_id)
        serializer = ClassSerializer(obj)
        return Response(serializer.data)

    def put(self, request, class_id):
        obj = classes.objects.get(id = class_id)
        serializer = ClassSerializer(obj, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.error_messages)
```



```
def patch(self, request, class_id):
    obj = classes.objects.get(id=class_id)
    serializer = ClassSerializer(obj, data=request.data, partial=True)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

def delete(self, request, class_id):
    obj = classes.objects.get(id = class_id)
    obj.delete()
    return Response(status=status.HTTP_204_NO_CONTENT)
```

- CREATE – using POST method we can request data and add it to the database
- READ – using GET method we can retrieve the data
- UPDATE – PUT if complete entry corresponding to a id is to be changed and PATCH if some part of a particular entry is to be updated
- DELETE – For deleting data
- **classeslist**, **classesdetail**: These classes define API endpoints for managing classes, **assignmentlist**, **assignmentdetail**: these classes handle requests related to assignments and **questionlist**, **questiondetail**: These classes handle requests related to questions.
- **Request Handling -** Requested data is accessed through request.data. Serializers are used to process incoming data and covert it to required format.
- **Response Handling -** Responses are generated using the Response class from rest framework and serialised data is turned into response body
- Various API links are created which can be used to deliver data (response) in json format

Eg - http://127.0.0.1:8000/mainapp/classes/



Classeslist

```
GET /mainapp/classes/
```

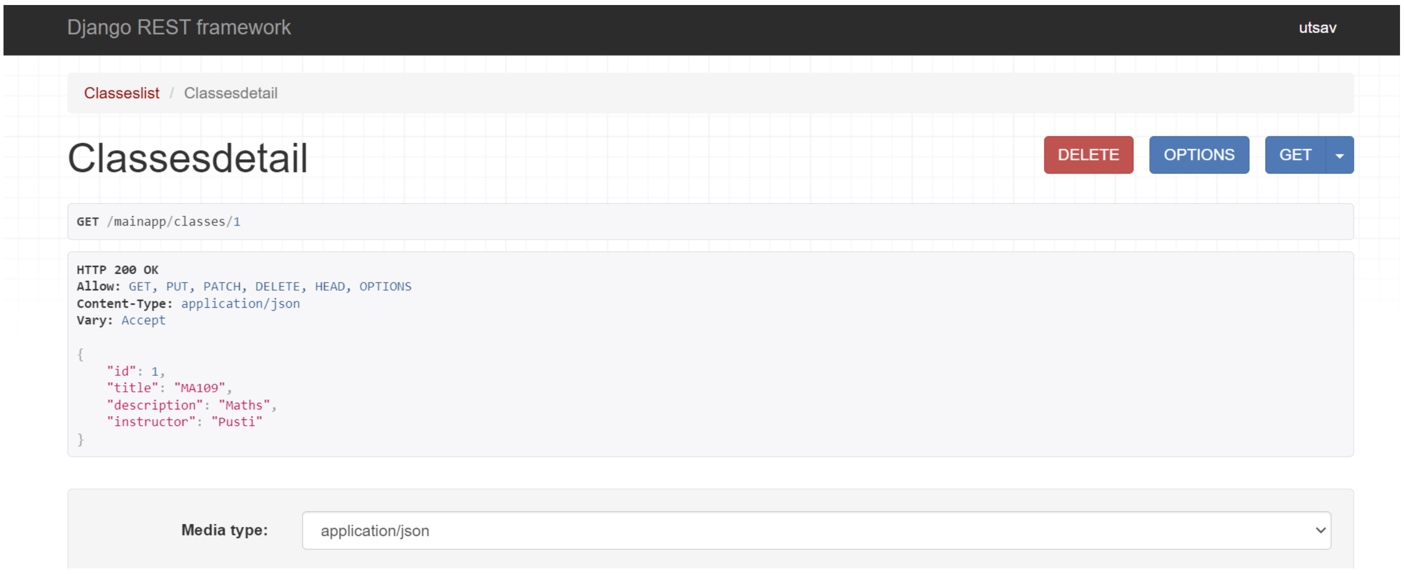
```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "title": "MA109",
        "description": "Maths",
        "instructor": "Pusti"
    },
    {
        "id": 2,
        "title": "PH111",
        "description": "PHysics",
        "instructor": "Jason"
    },
    {
        "id": 3,
        "title": "CH105",
        "description": "iNOrganic",
        "instructor": "swati"
    }
]
```

Media type: application/json

Content:

The APIs can be tested using REST framework default dashboard or 3rd party apps like postman.

For individual course -

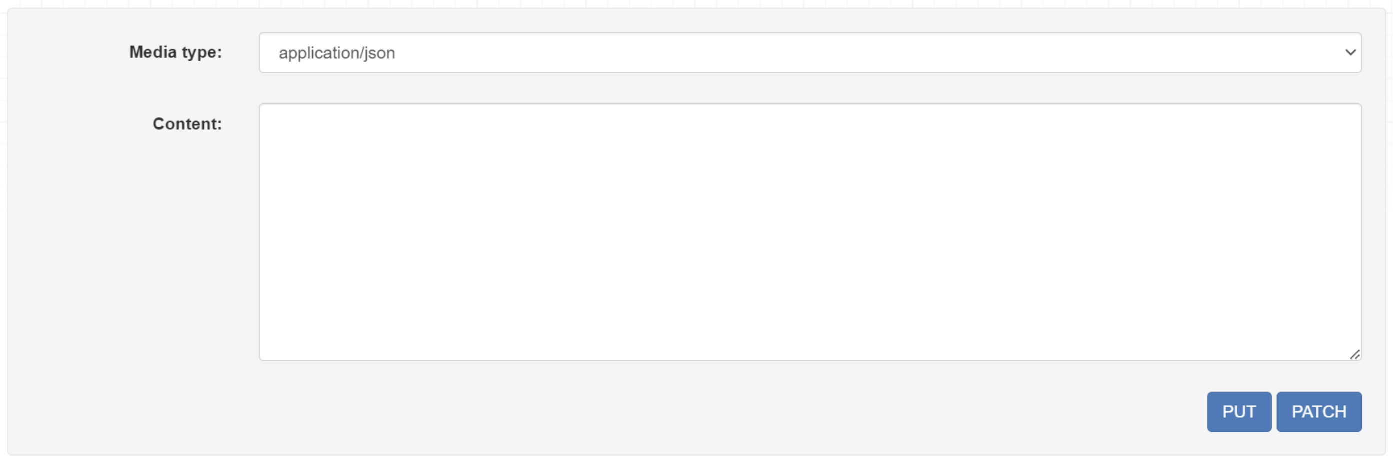


Classesdetail

```
GET /mainapp/classes/1
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "title": "MA109",
    "description": "Maths",
    "instructor": "Pusti"
}
```

Media type: application/json


Media type: application/json

Content:

Similarly all API endpoints can be verified using the links mentioned in the URLs section of codebase.