views.py

```
from student.analytics import getCompletionPercentages, getTutorialProgress
from django.http.response import HttpResponse, HttpResponseRedirect
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse
from django.views.decorators.http import require_POST, require_GET

from .forms import TutorialForm, LessonForm, LoginForm, RegistrationForm,
StudentProfileForm, InstructorProfileForm, ExerciseForm
from .models import Assignment, ExerciseFeedback, ExerciseSubmissionEvent,
LessonFeedback, LessonSubmissionEvent, Tutorial, Lesson, Exercise, StudentProfile,
InstructorProfile, Class, LessonProgress, ExerciseProgress, ExerciseSolution,
LoginEvent
from .analytics import get_overall_completion_rate, groupStudents
from .utils import run_testcases, is_testcases_pass

import json

def home(request):
    if request.user.is_authenticated:
        return redirect('classes', permanent=True)
    else:
        return redirect('login', permanent=True)


@login_required
def tutorial_statistics(request, class_id, tutorial_id):
    profile = InstructorProfile.objects.get(user=request.user)
    completionPercentages = getCompletionPercentages(
                                getTutorialProgress(tutorial_id)
                            )
    overallCompletionPercent = get_overall_completion_rate(completionPercentages)
    tutorial = Tutorial.objects.get(id=tutorial_id)

    context = {
        'tutorial': tutorial,
        'completionPercentages': completionPercentages,
        'overallCompletionRate': overallCompletionPercent,
        'profile': profile,
    }
    return render(request, 'tutorial_statistics.html', context)


@login_required
def logout_view(request):
    logout(request)
    return redirect('login', permanent=True)
```

```
@login_required
def account(request):
    try:
        profile = InstructorProfile.objects.get(user=request.user)
    except:
        profile = StudentProfile.objects.get(user=request.user)

    context = {
        'profile': profile
    }
    return render(request, 'account.html', context)


@login_required
def my_classes(request):
    try:
        profile = InstructorProfile.objects.get(user=request.user)
        classes = Class.objects.filter(instructor=profile)
        profile.role = 'I'
    except:
        profile = StudentProfile.objects.get(user=request.user)
        classes = profile.class_enrolled.all()
        profile.role = 'S'

    context = {
        'profile': profile,
        'classes': classes,
    }
    return render(request, 'my_classes.html', context)


# Create your views here.
@login_required
def class_tutorials(request, id):
    '''
    - assignment
        - student
        - tutorial

    We want to get the progress
        * we need all progress in the tutorial of the assignment
        * we need to retrieve the progress of all the lessons in the tutorial
        * 1 tutorial = [] <- array of progress for every lessons
        * we find if there is any incomplete progress in the lesson
        * if yes, we set the progress as incomplete for this tutorial
    TODO: Add the completed status to each Tutorial, in the tutorial cards?
    '''
    student = get_object_or_404(StudentProfile, user=request.user)
    assignments = Assignment.objects.filter(student=student,
tutorial__tutorial_class__id=id)
    progresses = []
    # this implicitly assumes that the progresses and tutorials are parallel
arrays
    # this assumption is valid given we loop through assignment array to assign
```

```
    tutorials and progresses
        for assignment in assignments:
            progress = False
            if assignment.tutorial.tutorial_type == Tutorial.LESSON:
                lessons = Lesson.objects.filter(tutorial=assignment.tutorial)
                for lesson in lessons:
                    lesson_completed =
    LessonProgress.objects.get(assignment=assignment, lesson=lesson).completed
                    if not lesson_completed:
                        break
            else:
                exercises = Exercise.objects.filter(tutorial=assignment.tutorial)
                for exercise in exercises:
                    exercise_completed =
    ExerciseProgress.objects.get(assignment=assignment, exercise=exercise)
                    if not exercise_completed:
                        break
            progresses.append(progress)

        context = {
            'assignments': assignments,
            'progresses': progresses,
            'profile': student
        }
        return render(request, 'class.html', context)


    @login_required
    def tutorial(request, id):
        profile = StudentProfile.objects.get(user=request.user)
        tutorial = get_object_or_404(Assignment, id=id).tutorial
        if tutorial.tutorial_type == Tutorial.LESSON:
            units = Lesson.objects.filter(tutorial__id=tutorial.id).order_by('order')
            progresses = [LessonProgress.objects.get(lesson=unit,
    assignment__id=id).completed for unit in units]
        else:
            units =
    Exercise.objects.filter(tutorial__id=tutorial.id).order_by('order')
            progresses = [ExerciseProgress.objects.get(exercise=unit,
    assignment__id=id).completed for unit in units]

        for i in range(len(units)):
            units[i].progress = progresses[i]


        context = {
            'tutorial': tutorial,
            'units': units,
            'profile': profile
        }
        return render(request, 'tutorial.html', context)


    @login_required
```

```
def lesson(request, id, assignment_id):
    profile = StudentProfile.objects.get(user=request.user)
    tutorial = get_object_or_404(Assignment, id=assignment_id).tutorial
    lesson = Lesson.objects.get(id=id)
    next_lesson = tutorial.get_next_lesson(id=lesson.id)
    prev_lesson = tutorial.get_prev_lesson(id=lesson.id)

    context = {
        'lesson': lesson,
        'next': next_lesson,
        'prev': prev_lesson,
        'profile': profile,
    }
    return render(request, 'lesson.html', context)

@login_required
def exercise(request, id, assignment_id):
    profile = StudentProfile.objects.get(user=request.user)
    assignment = get_object_or_404(Assignment, id=assignment_id)
    tutorial = assignment.tutorial
    exercise = Exercise.objects.get(id=id)
    testcases = json.loads(exercise.testcases)
    completed = ExerciseProgress.objects.get(exercise=exercise,
assignment=assignment)
    context = {
        'exercise': exercise,
        'testcases': testcases,
        'completed': completed,
        'profile': profile,
    }
    return render(request, 'exercise.html', context)


@require_POST
def submit_exercise(request, id, assignment_id):
    data = json.loads(request.body.decode("utf-8"))
    code = data['code']
    testcases = Exercise.objects.get(id=id).testcases
    results = run_testcases(code, testcases)
    if is_testcases_pass(results):
        # update progress
        progress = ExerciseProgress.objects.get(exercise__id=id,
assignment__id=assignment_id)
        progress.completed = True
        progress.save()

        # publish the solution
        exercise = Exercise.objects.get(id=id)
        assignment = Assignment.objects.get(id=assignment_id)
        solution = ExerciseSolution(exercise=exercise, assignment=assignment,
solution=code)
        solution.save()
    return JsonResponse(results, safe=False)
```

```
@require_POST
def record_exercise_attempts(request, id, assignment_id):
    data = json.loads(request.body.decode("utf-8"))
    num_of_attempts = data['attempts']
    is_pass = data['pass']
    duration = data['duration']
    assignment = Assignment.objects.get(id=assignment_id)
    exercise = Exercise.objects.get(id=id)
    event = ExerciseSubmissionEvent(assignment=assignment, exercise=exercise,
frequency=num_of_attempts, duration=duration, result=is_pass)
    event.save()
    return JsonResponse({'success': True})


@require_POST
def record_lesson_attempts(request, id, assignment_id):
    data = json.loads(request.body.decode("utf-8"))
    num_of_attempts = data['attempts']
    is_pass = data['pass']
    duration = data['duration']
    assignment = Assignment.objects.get(id=assignment_id)
    lesson = Lesson.objects.get(id=id)
    event = LessonSubmissionEvent(assignment=assignment, lesson=lesson,
frequency=num_of_attempts, duration=duration, result=is_pass)
    event.save()
    return JsonResponse({'success': True})


@require_POST
def submit_lesson(request, id, assignment_id):
    progress = LessonProgress.objects.get(lesson__id=id,
assignment_id=assignment_id)
    progress.completed = True
    progress.save()
    return JsonResponse({'successful': True})

'''
INSTRUCTOR VIEWS
'''
@login_required
def editor(request, class_id, tutorial_id):
    profile = InstructorProfile.objects.get(user=request.user)
    tutorial = Tutorial.objects.get(id=tutorial_id)
    if request.method == 'POST':
        if request.POST['tutorial_type'] == Tutorial.LESSON:
            # New lesson
            lesson_form = LessonForm(request.POST)
            if request.POST.get('content_id', None) == None:
                new_lesson = lesson_form.save(commit=False)
                new_lesson.tutorial = tutorial
                new_lesson.save()
            # Update old lesson
            else:
```

```
                if lesson_form.is_valid():
                    lesson = Lesson.objects.get(id=request.POST.get('content_id'))
                    cd = lesson_form.cleaned_data
                    lesson.title = cd['title']
                    lesson.order = cd['order']
                    lesson.markdown = cd['markdown']
                    lesson.exercise_question = cd['exercise_question']
                    lesson.exercise_ans = cd['exercise_ans']
                    lesson.save()

        else:
            # New exercise
            exercise_form = ExerciseForm(request.POST)
            if request.POST.get('content_id', None) == None:
                new_exercise = exercise_form.save(commit=False)
                new_exercise.tutorial = tutorial
                new_exercise.save()
            else: # update old exercise
                if exercise_form.is_valid():
                    exercise =
Exercise.objects.get(id=request.POST.get('content_id'))
                    cd = exercise_form.cleaned_data
                    exercise.title = cd['title']
                    exercise.order = cd['order']
                    exercise.question = cd['question']
                    exercise.testcases = cd['testcases']
                    exercise.save()

    if tutorial.tutorial_type == Tutorial.LESSON:
        form = LessonForm()
        materials = Lesson.objects.filter(tutorial=tutorial)
        if request.GET.get('content_id', None) != None:
            lesson = get_object_or_404(Lesson, id=request.GET['content_id'])
            data = {
                'title': lesson.title,
                'order': lesson.order,
                'markdown': lesson.markdown,
                'exercise_question': lesson.exercise_question,
                'exercise_ans': lesson.exercise_ans
            }
            form = LessonForm(initial=data)
    else:
        form = ExerciseForm()
        materials = Exercise.objects.filter(tutorial=tutorial)
        if request.GET.get('content_id', None) != None:
            exercise = get_object_or_404(Exercise, id=request.GET['content_id'])
            data = {
                'title': exercise.title,
                'order': exercise.order,
                'question': exercise.question,
                'testcases': exercise.testcases,
            }
            form = ExerciseForm(initial=data)
```

```
    context = {
        'form': form,
        'materials': materials,
        'profile': profile,
    }
    return render(request, 'editor.html', context)


@login_required
def tutorial_dir(request, class_id):
    '''
    * retrieve all tutorial objects from database
    * display as cards
    * allows instructor to assign tutorials to students by clicking \
    on cards.
        1. instructor click on assign
        2. popup a list of students
            * CONTENT:
            * students already assigned with this tutorial
                * how? filter all assignments with this tutorial
                * retrieve their student ids (using array comprehension?)
                * match with all students
                * matched students appear as ady assigneed
            * students not assigned appear as checkboxes
            * a shortcut button to assign to all students not assigned
    * Show button to Editor view as 'Create tutorial'
    '''
    profile = InstructorProfile.objects.get(user=request.user)
    if request.method == 'POST':
        tutorial_form = TutorialForm(request.POST)
        new_tutorial = tutorial_form.save(commit=False)
        new_tutorial.created_by = InstructorProfile.objects.get(user=request.user)
        new_tutorial.tutorial_class = Class.objects.get(id=class_id)
        new_tutorial.save()
        return redirect(f'/class/{class_id}/editor/{new_tutorial.id}',
permanent=True)

    the_class = get_object_or_404(Class, id=class_id)
    tutorials = Tutorial.objects.filter(tutorial_class__id=class_id)
    tutorial_form = TutorialForm()

    context = {
        'tutorials': tutorials,
        'tutorial_form': tutorial_form,
        'profile': profile,
    }
    return render(request, 'tutorial_dir.html', context)

@require_POST
def assign(request, class_id, tutorial_id):
    assigned_students = request.POST.getlist('assigned_student')
    for student_id in assigned_students:
        student = StudentProfile.objects.get(id=student_id)
        tutorial = Tutorial.objects.get(id=tutorial_id)
```

```
            assignment = Assignment.create(student=student, tutorial=tutorial)
        return redirect(f'/class/{class_id}/tutorial_dir', permanent=True)


    def assigned_students(request, class_id, tutorial_id):
        '''
        1. retrieve all students in this class
        2. for each student in the class, find an Assignment which has both the
    Student and Tutorial
        3. if not found: student not assigned (vice versa)
        '''
        students = StudentProfile.objects.filter(class_enrolled__id=class_id)
        response = []
        for student in students:
            is_assigned = False
            try:
                Assignment.objects.get(student=student, tutorial__id=tutorial_id)
                is_assigned = True
            except:
                pass
            response.append({
                'id': student.id,
                'name': student.user.username,
                'status': is_assigned
            })
        return JsonResponse(response, safe=False)


    def login_view(request):
        if request.method == 'POST':
            form = LoginForm(request.POST)
            if form.is_valid():
                cd = form.cleaned_data
                user = authenticate(request,
                                    username=cd['username'],
                                    password=cd['password'])
                if user is not None:
                    if user.is_active:
                        # get user profile
                        role_matching = True
                        try:
                            role = cd['role']
                            if role == 'S':
                                StudentProfile.objects.get(user=user)
                            else:
                                InstructorProfile.objects.get(user=user)
                        except:
                            role_matching = False

                        if role_matching:
                            login(request, user)
                            if cd['role'] == 'S':
                                student = StudentProfile.objects.get(user=user)
                                event = LoginEvent(student=student)
```

```
                            event.save()
                        return redirect('classes', permanent=True)
                    else:
                        context = {
                            'form': form,
                            'error': f"We found no matching account with username
{cd['username']}",
                        }
                        return render(request, 'login.html', context)

            else:
                return HttpResponse('Invalid Login')
    else:
        form = LoginForm()

    context = {
        'form': form
    }
    return render(request, 'login.html', context)


def register(request):
    if request.method == 'POST':
        user_form = RegistrationForm(request.POST)
        instructor_form = InstructorProfileForm()
        student_form = StudentProfileForm()
        if request.POST['role'] == 'student':
            profile_form = StudentProfileForm(request.POST, request.FILES)
        else:
            profile_form = InstructorProfileForm(request.POST, request.FILES)

        if user_form.is_valid() and profile_form.is_valid():
            new_user = user_form.save(commit=False)
            new_user.set_password(
                user_form.cleaned_data['password']
            )
            new_user.save()
            new_profile = profile_form.save(commit=False)
            new_profile.user = new_user
            new_profile.save()
            user = authenticate(request,
                                username=user_form.cleaned_data['username'],
                                password=user_form.cleaned_data['password'])
            login(request, user)
            return redirect('classes', permanent=True)
        else:
            context = {
                'user_form': user_form,
                'instructor_form': instructor_form,
                'student_form': student_form,
                'profile_form': profile_form
            }
            return render(request, 'register.html', context)
```

```
    else:
        user_form = RegistrationForm()
        instructor_form = InstructorProfileForm()
        student_form = StudentProfileForm()

    context = {
        'user_form': user_form,
        'instructor_form': instructor_form,
        'student_form': student_form,
    }
    return render(request, 'register.html', context)


@require_POST
def feedback(request, id, assignment_id):
    data = json.loads(request.body.decode("utf-8"))
    feedback = data['feedback']
    rating = data['rating']
    content_type = data['content_type']
    assignment = Assignment.objects.get(id=assignment_id)
    if content_type == Tutorial.LESSON:
        lesson = Lesson.objects.get(id=id)
        lesson_feedback = LessonFeedback(lesson=lesson, assignment=assignment,
                                         feedback=feedback, rating=rating)
        lesson_feedback.save()
    else:
        exercise = Exercise.objects.get(id=id)
        exercise_feedback = ExerciseFeedback(exercise=exercise,
assignment=assignment,
                                         feedback=feedback, rating=rating)
        exercise_feedback.save()
    return JsonResponse({'successful': True})


@require_GET
def get_solutions(request, id, assignment_id):
    solutions = ExerciseSolution.objects.filter(exercise__id=id)
    response_arr = []
    for solution in solutions:
        username = solution.assignment.student.user.username
        solution = solution.solution
        response = {
            'username': username,
            'solution': solution
        }
        response_arr.append(response)

    return JsonResponse(response_arr, safe=False)

'''
Pass an array of Student Profiles which is a 2-dimensional array.
Example Output:
[
```

```
    [Student Profile 2, Student Profile 8, Student Profile 5],
    [Student Profile 3, Student Profile 1, Student Profile 7],
    ...
]
'''
@login_required
def grouping(request, class_id):
    students = StudentProfile.objects.filter(class_enrolled__id=class_id)
    if request.method == "POST":
        group_size = int(request.POST['size'])
        criteria = request.POST['criteria']
        students = groupStudents(students, group_size, criteria)
    context = {
        'students': students
    }
    return render(request, 'grouping.html', context)
```

models.py

```
from django.db import models
from django.conf import settings

class InstructorProfile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL,
                                on_delete=models.CASCADE)
    phone = models.CharField(max_length=50)
    # TODO: specify a default photo
    photo = models.ImageField(upload_to='profile')


class Class(models.Model):
    '''
    Class is a collection of students, instructors, and tutorials
    Represents a class in real-life.
    '''
    title = models.CharField(max_length=400)

    # Introduction in Markdown
    introduction = models.TextField()
    instructor = models.ForeignKey(InstructorProfile, on_delete=models.CASCADE)


class StudentProfile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL,
                                on_delete=models.CASCADE)
    phone = models.CharField(max_length=50)
    # TODO: specify a default photo
    photo = models.ImageField(upload_to='profile')
    class_enrolled = models.ManyToManyField(Class,
                                            related_name='enrolled_in',
                                            blank=True)
```

```
class Tutorial(models.Model):
    '''
    Tutorial is the main content on the platform.
    Created by the instructors, assigned to the students.
    '''
    LESSON = 'LS'
    EXERCISE = 'EX'
    TUTORIAL_TYPE = [
        (LESSON, 'Lesson'),
        (EXERCISE, 'Exercise')
    ]

    title = models.CharField(max_length=400)
    created_by = models.ForeignKey(InstructorProfile,
                                   on_delete=models.DO_NOTHING)
    tutorial_class = models.ForeignKey(Class, on_delete=models.CASCADE)
    date_created = models.DateField(auto_now_add=True)
    tutorial_type = models.CharField(max_length=2, choices=TUTORIAL_TYPE)

    # Assigned to which student
    assignment = models.ManyToManyField(StudentProfile,
                                        through='Assignment',
                                        blank=True)
    def get_next_lesson(self, id):
        lessons = Lesson.objects.filter(tutorial=self)
        lessons_size = len(lessons)
        for i in range(lessons_size):
            if lessons[i].id == id:
                # last lesson
                if i == lessons_size - 1:
                    return None
                return lessons[i + 1]
        return None

    def get_prev_lesson(self, id):
        lessons = Exercise.objects.filter(tutorial=self)
        lessons_size = len(lessons)
        for i in range(lessons_size):
            if lessons[i].id == id:
                # last lesson
                if i == 0:
                    return None
                return lessons[i - 1]
        return None

    def get_next_exercise(self, id):
        exercises = Exercise.objects.filter(tutorial=self)
        exercises_size = len(exercises)
        for i in range(exercises_size):
            if exercises[i].id == id:
                # last lesson
                if i == exercises_size - 1:
```

```
                    return None
                return exercises[i + 1]
        return None

    def get_prev_exercise(self, id):
        exercises = Exercise.objects.filter(tutorial=self)
        exercises_size = len(exercises)
        for i in range(exercises_size):
            if exercises[i].id == id:
                # last exercise
                if i == 0:
                    return None
                return exercises[i - 1]
        return None


class Lesson(models.Model):
    '''
    Lesson is a sub-unit of Tutorial. It stores a read-only content,
    with an optional quick exercise.
    '''

    # Parent tutorial
    tutorial = models.ForeignKey(Tutorial, on_delete=models.CASCADE)

    title = models.CharField(max_length=400)

    # The order of this exercise in the tutorial
    order = models.IntegerField()

    # The content in Markdown
    markdown = models.TextField()

    # Optional question and answer for quick exercise
    exercise_question = models.TextField(blank=True)
    exercise_ans = models.TextField(blank=True)
    date_created = models.DateField(auto_now_add=True)

    def __str__(self):
        return f'ID:{self.id}. Title: {self.title}'


class Exercise(models.Model):
    '''
    Exercise is a sub-unit of Tutorial. It stores the programming exercise.
    '''

    # Parent Tutorial
    tutorial = models.ForeignKey(Tutorial, on_delete=models.CASCADE)

    # Title of this exercise
    title = models.CharField(max_length=400)

    # The order of this exercise in the tutorial
```

```
    order = models.IntegerField()

    # The question in Markdown
    question = models.TextField()

    # Testcases in JSON
    testcases = models.TextField()

    # Date Exercise is created
    date_created = models.DateField(auto_now_add=True)


class Assignment(models.Model):
    student = models.ForeignKey(StudentProfile, on_delete=models.CASCADE)
    tutorial = models.ForeignKey(Tutorial, on_delete=models.CASCADE,
                                 related_name="tutorial")
    progress = models.IntegerField(default=0)
    is_optional = models.BooleanField(default=False)

    # keep in mind this has a side effect
    # another possible solution might be to use a SAVE signal receiver
    @classmethod
    def create(cls, student, tutorial):
        assignment = cls(student=student, tutorial=tutorial)
        assignment.save()
        if tutorial.tutorial_type == Tutorial.LESSON:
            lessons = Lesson.objects.filter(tutorial=tutorial)
            for lesson in lessons:
                progress = LessonProgress(lesson=lesson, assignment=assignment)
                progress.save()
        else:
            exercises = Exercise.objects.filter(tutorial=tutorial)
            for exercise in exercises:
                progress = ExerciseProgress(exercise=exercise,
assignment=assignment)
                progress.save()
        return assignment

    def create_exercises_progress(self):
        exercises = Exercise.objects.filter(tutorial=self.tutorial)
        progresses = [ExerciseProgress(exercise=exercise, assignment=self)
                      for exercise in exercises]
        return progresses

    def create_lessons_progress(self):
        lessons = Lesson.objects.filter(tutorial=self.tutorial)
        progresses = [LessonProgress(lesson=lesson, assignment=self)
                      for lesson in lessons]
        return progresses

    def __str__(self):
        return f'{self.tutorial.title}: {self.student.user.username}'
```

```
class LessonProgress(models.Model):
    lesson = models.ForeignKey(Lesson, on_delete=models.CASCADE)
    assignment = models.ForeignKey(Assignment, on_delete=models.CASCADE)
    completed = models.BooleanField(default=False)


class ExerciseProgress(models.Model):
    exercise = models.ForeignKey(Exercise, on_delete=models.CASCADE)
    assignment = models.ForeignKey(Assignment, on_delete=models.CASCADE)
    completed = models.BooleanField(default=False)


class ExerciseSolution(models.Model):
    exercise = models.ForeignKey(Exercise, on_delete=models.CASCADE)
    assignment = models.ForeignKey(Assignment, on_delete=models.CASCADE)
    solution = models.TextField()


class LessonFeedback(models.Model):
    lesson = models.ForeignKey(Lesson, on_delete=models.CASCADE)
    assignment = models.ForeignKey(Assignment, on_delete=models.CASCADE)
    feedback = models.TextField()
    rating = models.IntegerField()


class ExerciseFeedback(models.Model):
    exercise = models.ForeignKey(Exercise, on_delete=models.CASCADE)
    assignment = models.ForeignKey(Assignment, on_delete=models.CASCADE)
    feedback = models.TextField()
    rating = models.IntegerField()


class LoginEvent(models.Model):
    student = models.ForeignKey(StudentProfile, on_delete=models.CASCADE)
    time = models.DateField(auto_now_add=True)


class ExerciseSubmissionEvent(models.Model):
    assignment = models.ForeignKey(Assignment, on_delete=models.CASCADE)
    exercise = models.ForeignKey(Exercise, on_delete=models.CASCADE)
    frequency = models.IntegerField
    duration = models.FloatField()
    result = models.BooleanField()


class LessonSubmissionEvent(models.Model):
    assignment = models.ForeignKey(Assignment, on_delete=models.CASCADE)
    lesson = models.ForeignKey(Lesson, on_delete=models.CASCADE)
    frequency = models.IntegerField
    duration = models.FloatField()
    result = models.BooleanField()
```

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    # path for logins
    path('login', views.login_view, name='login'),
    path('logout', views.logout_view, name='logout'),
    path('register', views.register, name='register'),
    path('account', views.account, name='account'),
    path('', views.home, name='home'),
    path('classes', views.my_classes, name='classes'),
    path('class/<int:id>', views.class_tutorials, name='class_tutorials'),
    path('tutorial/<int:id>', views.tutorial, name='tutorial'),
    path('tutorial/<int:assignment_id>/lesson/<int:id>', views.lesson,
name='lesson'),
    path('tutorial/<int:assignment_id>/exercise/<int:id>', views.exercise,
name='exercise'),
    # POST path for submitting exercises / lessons
    path('tutorial/<int:assignment_id>/submit_exercise/<int:id>',
views.submit_exercise, name='submit_exercise'),
    path('tutorial/<int:assignment_id>/submit_lesson/<int:id>',
views.submit_lesson, name='submit_lesson'),
    path('tutorial/<int:assignment_id>/record_exercise_attempts/<int:id>',
views.record_exercise_attempts, name='record_exercise_attempts'),
    path('tutorial/<int:assignment_id>/record_lesson_attempts/<int:id>',
views.record_lesson_attempts, name='record_tutorial_attempts'),
    path('feedback/<int:assignment_id>/<int:id>', views.feedback,
name='feedback'),
    # GET path for getting solutions
    path('get_solutions/<int:assignment_id>/<int:id>', views.get_solutions,
name='get_solutions'),
    # path for instructors
    path('class/<int:class_id>/editor/<int:tutorial_id>', views.editor,
name='editor'),
    path('class/<int:class_id>/tutorial_dir', views.tutorial_dir,
name='tutorial_dir'),
    path('class/<int:class_id>/grouping', views.grouping, name='grouping'),
    path('class/<int:class_id>/tutorial/<int:tutorial_id>/assignment_status',
views.assigned_students, name='assignment_status'),
    # path for instructors: ANALYTICS
    path('class/<int:class_id>/tutorial_statistics/<int:tutorial_id>',
views.tutorial_statistics, name='tutorial_statistics'),
    # path for instructors that require POST
    path('assign/<int:class_id>/<int:tutorial_id>', views.assign, name='assign'),
]
```