Abstract

In the modern world, e-learning is widespread in higher education institutions. In this learning format, the resource exchange process takes place online and requires a single and simplified storage system to avoid data loss. Hence, for students, in addition to completing projects, it is important to save them for further use and replenishment of the portfolio. Also, for a storage system, it is important to be an easy-to-use platform.

Inspired by this idea, we have decided to create a web platform that includes a repository that allows us to store finished projects and prevent the loss. The platform also has features to enable students to interact with teachers and companies.

This product is very useful for facilitating the exchange of learning materials that can be accessed through the repository and thus go beyond the educational environment. We are confident this product will allow us to contribute to improving and perfecting the quality of the educational process.

The purpose of our platform to solve an actual problem, so its creation required a lot of analysis and planning. Next, we want to talk in detail about the development and implementation of this idea in this documentation.

Аңдатпа

Қазіргі әлемде жоғары оқу орындарында электронды оқыту кең таралған. Оқытудың осы форматында ресурстармен алмасу процесі онлайн режимге өтеді және деректердің жоғалуын болдырмау үшін бірыңғай сақтау жүйесін қажет етеді. Яғни, студенттерге жобаларды орындаудан бөлек, оларды одан әрі пайдалану және портфолионы толықтыру үшін сақтау маңызды. Сондай-ақ, сақтау платформасы пайдалануға оңай және ыңғайлы болуы керек.

Осы идеядан шабыт алып, біз дайын жобаларды сақтауға және оларды өңдеуге мүмкіндік беретін репозиторийді қамтитын веб-платформа құруды шештік. Платформада студенттердің оқытушылармен және компаниялармен өзара әрекеттесуіне арналған функциялар бар.

Бұл өнім репозиторий арқылы оқу материалдарымен алмасуды қарапайым және қолжетімді ету қарастырылған, осылайша білім беру ортасының шекарасынан шығуға болады. Біз бұл өнім білім беру үдерісінің сапасын жақсартуға өз үлесімізді қосуға мүмкіндік береді деп үміттенеміз.

Біздің платформа өзекті мәселені шешу үшін құрылды, сәйкесінше оны құру ұзақ талдау мен жоспарлауды қажет етті.Әрі қарай, біз осы идеяны әзірлеу және жүзеге асыру процесі туралы осы құжаттамада егжей-тегжейлі баяндағымыз келеді.

Аннотация

В современном мире широко распространено электронное обучение в высших учебных заведениях. При таком формате обучения, процесс обмена ресурсами переходит в онлайн режим и требует единую и упрощенную систему хранения, чтобы избежать потерю данных. То есть, студентам, помимо выполнения проектов, важно их сохранять для дальнейшего использования и пополнения портфолио. Также важно, чтобы платформа для хранения была простой в использовании и удобной.

Вдохновившись данной идеей, мы решили создать веб-платформу, включающую в себя репозиторий, позволяющий хранить готовые проекты и не терять их. Платформа также имеет функции для взаимодействия студентов с преподавателями и компаниями.

Данный продукт очень полезен для упрощения обмена учебными материалами, которые можно сделать доступными через репозиторий, и, таким образом, выйти за рамки образовательной среды. Мы надеемся, что данный продукт позволит нам внести свою лепту в улучшение качества образовательного процесса.

Наша платформа была создана для решения актуальной проблемы, соответственно, его создание требовало длительного анализа и планирования. Дальше мы хотим рассказать в деталях о процессе разработки и осуществления данной идеи в этой документации.

Contents

Chapter 1

Introduction

1.1 Motivation

What is a repository for? Nowadays, there is an active spread of digital specialties, which is associated with the rapid development of IT technologies. We often hear stories of people who worked in a certain field for several years, and after a while, having decided to change their field of activity, they took up the study of programming, eventually achieving success in it.

This entails the creation and appearance of a big amount of data and actualizes the problem of their storage. And at this point, there arises a great need to use the repository.

We faced this problem in the initial courses at the university. Completed projects were deleted from the computer without a trace at the end of the course, and when it became necessary to replenish the portfolio, we came to the understanding that the project could no longer be found.

It can be difficult for students in the early stages to use well-known repositories like Github, for the same reason we wanted to have a repository that would allow us to easily download and store all projects in one place. It was from this idea that the decision to create our platform arose.

Our platform is aimed at improving the quality of e-education and helps to take the first steps in building a future career. To do this, we have the function of creating competitions for students on behalf of the company. Companies can, in coordination with the university and teachers, organize hackathons and other types of competitions in which students can participate in teams.

With the introduction of the distance learning format, we all see that the educational process requires innovation, and several arguments can be made to confirm this. The environment that creates a productive atmosphere disappears, connections with fellow students are lost, and without society, it is difficult for a person who is accustomed to living in it to develop. We hope that our platform will help save the gray days of students at such a time. Every student would like to take steps towards their career as early as possible, and it is available right

now. Our platform allows you to create a team with friends and participate in various competitions offered by companies. Thus, students can develop together, opening up new horizons for themselves.

In this project, we have collected functions that would be very useful during studying, we can say those functions in educational platforms that we lacked during studying. A user-friendly interface, beautiful design, and a set of opportunities will allow students not to lose their finished works, take part in interesting competitions, create their own future right now, and be closer to their dreams.

1.2 Aims and Objectives

The creation of our project is related to our user experience as a student. Initially, we wanted to create a project that would help improve the tools for the educational process. Before starting to work on the project, an analysis of the learning platforms used by us (google classroom, moodle) was carried out. Clear functions have been identified that we want to work on. Before that, we had not been able to create such a global product ourselves, although we had participated in smaller projects.

The goal of this project is to improve the quality of e-learning, which has already become part of everyday life in the field of education.

1.3 Thesis Outline

Chapter 1 is an Introduction chapter, which provides diving into the main idea of the project, with the goals that we pursue by creating it.
Chapter Project details contains a description of the project's functions, instructions for users, and the mechanics of the project.
Chapter Used tools and technologies describes the project development process, the tools used are described in detail.
Chapter Project Development presents the project structure with code examples and links to resources.
Chapter 5 is a Conclusion part, which contains the conclusion of the project.

Chapter 2

Project details

In this chapter, we will delve into the details of the project, take a closer look at what functions and capabilities this project gives us. Contains instructions that will allow you to learn more about how to use this platform.

2.1 Evolution of e-learning

The term e-learning came into use about 20 years ago, but there is evidence to support the existence of early forms of e-learning even in the 19th century. [4]

Isaac Pitman in 1840 introduced the method of correspondence by mail into the educational process, which he taught to his students. This method was later used to send and receive homework assignments. Over time, actively developing technologies were introduced into new teaching methods, which made it possible to create the very first online courses. They were introduced by the creator of Atari in 1984 to people with personal computers. [2]

This progress was followed by several new online courses, platforms that were also useful for students and learners, and some of them are still popular today, such as YouTube.

Now let's take a look at one of the currently popular educational platforms - Moodle. Moodle is a learning platform originally developed by Martin Dougamas (the first version of Moodle was released on August 20, 2002). Moodle, as a reliable open-source e-learning platform, has been used and developed in the following years by global collaborative efforts of the international community. Moodle is designed and continues to evolve to provide educators, administrators, and students with a single secure, secure and integrated system to create a personalized learning environment.

And now, having melted the hope that our work will be the next generation, we created ProjectX.

This is a project for integration into an educational platform that contains many useful features. It is important to note that when creating this project, we were driven by thoughts about the development of e-learning and improving the

quality of distance education, which is still relevant today.

We, as students, who tested the work of existing educational platforms on ourselves, wanted to bring to them even more interesting and useful functions that we would like to have.

Let's try to understand in detail the functionality and mechanics of this project.

2.2 User: Student

Many people know from childhood what they want to be when they grow up, but in adulthood, it is not always easy to fulfill their dreams. The question of employment is one of the most important for students both when they enroll and during their studies, and sometimes even after graduation.

Our platform will help you take the first steps in creating your future career. To that end, we have a function to create competitions for students on behalf of the company. Companies can, in coordination with the university and professors, organize hackathons and other types of competitions in which students can participate in teams.

With the introduction of the distance learning format, we can all see that the learning process requires innovation, and there are several arguments to support this. The environment that creates a productive atmosphere disappears, connection with classmates is lost, and it is difficult for a person who is used to living in the community to develop without it. We hope that our platform will help save the gray everyday life of students in such a time.

Every student would like to take steps towards their career as early as possible, and this is available right now. Our platform allows students to create a team with their friends and participate in various competitions offered by companies. In this way, students can develop together and discover new horizons. In this project we collected features that would be very useful, we can say that these features in educational platforms we lacked during the research.

The main role in the project is played by the student, as the project is aimed at improving the quality of his education.

The student can register using the SDU email and subsequently log in to the site using the same login and password.

When logging in, there are 4 tabs: my courses, available courses, my portfolio, and help. The first tab always displays my courses. This page displays the entire list of courses whose title is highlighted in blue or assignments whose title is highlighted in red that the student is enrolled in. On the left side is a list of courses by a title that you can apply a filter to. By clicking on the block with the course, the student has access to the assignments from this course, the list of students who are registered for this course, if it is a challenge - he can see a list of teams that participate in it, and detailed information about each participant.

With the "Follow" button, the student can join an existing team or, with the "Add" button, create his or her own. When clicking on a block with a team, by clicking on the "details" button, the student can see the files that the team has turned in for the assignment.

In the assignment section, the student can upload, delete, or download folders or files to the computer, and they can always be stored in that location.

So, the files that the student has uploaded for the assignment can remain there until the student wants to delete them. This feature serves as a repository.

When students goes to the available courses tab, the entire list of courses available to the student appears. By clicking on the "Subscribe" button, the student can register for the course. If a course block is marked as "Subscribed", then the student is already on the student list for that course.

The "My portfolio" tab contains detailed information about the student in the form of a CV. At the bottom of the page there is a "Download" button, by clicking on which a student can receive a PDF file with a CV.

In the last "help" tab, you can get the necessary information and instructions on how the site works.

A user-friendly interface, beautiful design, a set of functions will allow students not to lose their finished works, participate in interesting competitions, create their future right now and be closer to their dreams.

2.3 User: Teacher

Although the student is the main user of the platform, the instructor has significant advantages.

The teacher must also log in using the SDU mail. The teacher also has three tabs when logging in: my courses, my course details, my portfolio.

First, when you log in, the My Courses tab opens, displaying all of the courses created by this instructor. Also in this tab there are three tabs: in progress, completed, add a new course. The allocation to the first two tabs is based on the course length set by the teacher when the course was created.

The instructor is the only person who can create new courses and tasks, edit assignments and provide access to company guests.

The My Course Details tab has a View button that allows you to view course details, a list of participants who have signed up for a course, or a list of teams and their participants if that's a problem.

Only the teacher can add a company representative to the task, thereby giving him access to view the work and portfolio of students. In addition, the teacher can only view and download student files to the computer.

The teacher cannot delete the work of the students, everything else is available to him. In the tab "My portfolio", like the students, there is an informational resume of the teacher. The teacher can also download it for himself.

2.4 User: Company

We have added interactions with companies so that students have the opportunity to show themselves and see what opportunities await them.

Company representatives can only enter by gaining access from the instructor who is teaching the course or assignment. Once added to the list, companies can log in as a guest. To do this, there is a special link in the authorization window.

After authorization, 2 links open for companies: "available courses", "help".

Automatically goes to the first tab and in the content, we can see all courses or tasks to which this company representative was added. They can view course details, student information, view files that they have uploaded to complete the assignment. But companies cannot change or add anything to the site.

We think that even such a small integration of interaction between universities and companies can help students find work, fulfill their dreams, or at least get motivated. And companies will not be left without a win either, they will be able to select the best students who can become their loyal colleagues in the future.

Chapter 3

Used tools and technologies

This chapter will provide a detailed description of the tools used for development. The tools must be familiar to the developers so as not to cause complications during the project and not to waste time in vain.

3.1 Front end

3.1.1 Visual Studio Code

For front-end development, we used the program Visual Studio Code. Why did we use this editor?

Visual Studio Code is a powerful code editor, it allows you to connect a git system and has a built-in debugger, as well as consoles and various useful extensions only pleases the developer. Also, a plus is the support of different programming languages, that is, during the layout is not necessary to use another editor for JavaScript code, in one place you can run the site on the local host and test different functionalities.

3.1.2 Extensions

In developing the front-end projects, our front-end developer used different code editor extensions. For the correct operation and compilation of the Sass files, we used the 'Live Sass compiler', which turns the sass files into a minified CSS file. We also used the extension 'All Autocomplete, Auto Close Tag, Beautify' for convenience and the comfort of the coding process.

Also, we have tested our code through 'Live Server', so we ran our site on the local host. To work with JavaScript files we used 'Code Runner', which helped us to run our JavaScript codes in a blink of an eye. With the 'JS Hint' extension, we saw small bugs and reworked them in the process.

3.2 Database

3.2.1 Relational Database Model

One of the main functions of the project is data storage. Therefore, to begin with, our project needed to choose a DBMS where data from the site could be stored. Virtually all developers of modern applications that involve communication with database systems focus on relational DBMSs. According to analysts in 2010, relational DBMSs are used in the vast majority of large information system development projects. This is also why the team chose PostgreSQL for implementation. The database consists of entities and their relationships. These are the main components of the database. Unfortunately there are no general rules for defining what counts as an entity and what counts as a relationship. But we have assumed that by relationships we mean tables with data.

A properly constructed data model is responsible for the structure, manipulation and integrity of the data. Turning to the practical part, the data model is based on the natural representation of the data contained, not on the functionality of the application. Usually, how we represent students and teachers in real life is how we create tables. To create tables, we use Data Definition Language (DDL) commands such as CREATE - Creates objects in the database, ALTER - Alters objects of the database.This way all created tables, fields, primary and foreign keys are responsible for proper data storage.

3.2.2 Models

As we said before, data storage will be done on a database. But the back end, which is written in Python, works directly with the data source. In Python the most popular database adapter for working with PostgreSQL is psycopg2. This library is written in C based on libpq wrapper, resulting in being both efficient and secure.

To work directly with tables in the back end part we use the Models class. Thus each table can be represented as a class that inherits the main functions from the django.db.models.Model. Model attributes are represented as table fields. Moreover, this class has many functions which helps to maintain consistency. First, Django can automatically generate an identity field with DEFAULT AUTO FIELD setting.
Second, Cascading rules. Thanks to it at deletion of concrete record it is possible to delete all rows which are connected with an foreign key from the related tables. Thirdly, for models it is possible to write methods as for usual classes. Thus, one can easily manage specific rows.

To make changes to the data model, Django uses the Migration way. Once the model class has been changed, these changes (e.g. removing columns, changing data types) need to be made in the database. The makemigrations command scans

and reconciles your class with the previous version, then generates a migration file with the new changes. The migration file contains four attributes, of which there are two(dependencies, operations) that are used in many cases.

3.2.3 pgAdmin

To simplify administration on PostgreSQL server, a tool such as pgAdmin is included in the basic installation package. It is a graphical client for working with the server, through which we can manage the databases in a convenient way. You can use pgAdmin for anything from writing basic SQL queries to monitoring your databases and setting up advanced database architectures.

3.3 Back end

3.3.1 Django

For the backend part, we used the python Django framework. [3]

Django is considered the best web framework is written in Python, which was first introduced by two web programmers, Adrian Holovaty and Simon Willison in 2005. For 16 years of its existence, it has changed and improved a lot. This tool is convenient to use for developing websites that work with databases. Django provides a faster and safer build and frees the developer from the routine tasks of web development, eliminating the need to create solutions from scratch.

It works well with all types of websites and supports multiple formats. Since it is written in Python, its structure corresponds to the features of the language. In Django [1], developers work with an MVC (Model-View-Template) architecture that can be used independently of each other. In this architecture, Models contain data information, the View accepts HTTP requests, implements business logic defined by methods and properties, and sends an HTTP response in response to requests. Templates are files with HTML code.

The advantages of this framework: the administrative panel(the Django administrative panel is automatically generated when creating an application), SEO-friendliness (the code written in Python is readable and understandable even to untrained people), libraries (with which it is convenient to solve special tasks), ORM (which ensures the interaction of the application with databases).

3.3.2 PyCharm

For development, we used PyCharm. It is a cross-platform development environment that can be used on Windows, MacOS, and Linux.

The environment includes code analysis tools, a debugger, testing tools, and version control options. It also helps developers create Python plugins using

the various available APIs. The IDE allows us to work with multiple databases directly without integrating them with other tools.

Although it is specifically designed for Python, HTML, CSS, and Javascript files can also be created using this environment.

3.4 UX/UI Design

It took a week to create the site. For the project, we presented how the interface works, how consumers interact with it, and turned the script from start to finish. Our biggest concern was how the final visual would look. The design of the project of our site was created in Figma, and additional sites were used: Coolors, ColorSpace, for the color scheme; pexels for quality paintings.

3.4.1 Figma

Figma has many advantages: it is very convenient to create and maintain design styles for the entire project, you can use components (related elements) on a permanent basis, a large selection of beautiful fonts. Likewise, ease of use. the ability to make responsive whole pages. Wide range of fonts. During the project, basic functions were used, for example: a triangle (for an icon for a quick transition to registration), a rectangle (for highlighting key functions and frames, blocks), an ellipse (for text), as well as in the Fill section, a radiance function was used (darkening, concentration colors) and text, grouping of main pages.

3.4.2 Style

The minimalism style was chosen. Minimalism implies the use of simple images, the absence of unnecessary elements on the layout. Why this particular style? Because in this design, the emphasis is on the content, not the design. Color plays a significant role in this style. We did not want to deviate from the topic of our university, as the site is purposefully made for SDU students.

And the color was chosen blue.[5] The blue color is considered a sign of trust, calmness, openness, security. Also, dark shades of blue, that is, a secondary color, create a sense of reliability and professionalism. The background color is white. So that the student is focused on the main thing, and on certain pages of the site, pictures are added that correspond to the audience, age category, and occupation.

Chapter 4

Project Development

This chapter will describe the main points in development with code examples, which you can see in Appendix A

4.1 Authorization

Code located in Appendix A.1
As mentioned earlier, we have three kinds of user, and for each user there must be three kinds of login. We have

`def user_login(), register(), company_login()`

in views.py.to check the user via mail we used emailbackend.py function

`CustomBackend()`

.
They check the user's mail using if-else, try-catch functions. If user has domain 'stu.sdu.edu.kz' redirects to student-login page, if user has domain 'sdu.edu.kz' redirects to teacher-login page. If there is a company account in the database, then the login is possible at once. There is also a function in decorators.py that enters an error when a user has an account and redirects the user to the login-page.

4.2 Course View

Code located in Appendix A.2
In `in_progress.html` there are courses available for student, he can see the details of the course and all the tasks, he can also create/delete/upload his completed work.
To execute this functionality we have created in `home/views.py` function

`StudentCourseView()`

if the course type is 'course' function

`StudentTaskView(), StudentTaskDetailView()`

is executed. If the course type is 'challenge' function

`ChallengeDetailView()`

is executed.
For the company in `available_course_com.html` there are courses that the company representative has access to. Here he can see the course details, all the tasks and the list of participants.
To implement this functionality, we have created the

`CompanyCourseView()`

function in `home/views.py`.If the course type is 'course' then the

`CompanyTaskView(), CompanyDetailTaskView(), CompanyTaskParticipantsView()`

functions are triggered. If the course type is 'challenge', then functions

`CompanyDetailChallengeView(),CompanyDetailCommandView()`

are triggered. The teacher's functions' mechanics are the same as the company's.

4.3 Filtering and SearchBar

Code located in Appendix A.3
The available courses page has a filter box on the left side and a search bar at the top. All users can filter the courses by category and search for courses by name. For filtering we used loops and conditional statements `for,if,` Django Objects Filter, Python Lambda functions, and for search bar we used conditional statements and Django Objects Filter in `home/views.py` function

`AvailableCourseView()`

.

4.4 Teacher's functionalities (create/edit/delete)

Code located in Appendix A.4
The teacher can create, delete, edit courses/challenges, and tasks.
For that we used the

`TeacherCourseView(),TeacherCreateCourseView(),DeleteTaskView(), UpdateTaskView(),CreateTaskView(),DeleteCourseView()`

functions in `home/views.py`.

4.5 Teacher's functionality (giving access to companies)

Code located in Appendix A.5
A teacher can give access to a company representatives .
For that, the teacher has to create a new user or add a user to the `course.html`.
For this functionality we used in pathhome/views.py function

```
TeacherAddMemberForCourseView(),
TeacherAddNewMemberForCourseView()
```

4.6 Student's work submission

Code located in Appendices A.3, A.4, A.5
In `home/student/task_details.html`, student can create, delete, edit folder and interact with files.
To do this, we created in `home/views.py` functions

```
ChallengeDetailView(), StudentTaskDetailView()
```

in `filesystem/views.py` functions

```
new_project(),delete_project(),
get_project(),new_folder(),
delete_folder(),get_folder(),new_file(),
delete_file(),get_file()
```

We also used js file collaboration for this. We want to note that the created folders and uploaded files are saved in our storage directory.

4.7 Student Upload File

Code located in Appendix A.6
The student can upload completed work in the

```
task_details
```

page.
He can create his own folder in the task and upload his files inside it. Also, the student has the privilege to upload his files, delete them, and return to the folder he created. The downloaded files will be saved in the storage folder.
To implement this functionality, the

```
host,folders,files,path,file,folder,
folder_name,reader,parent_folder
```

variables were created.
Ajax technology was used to send requests without refreshing the page, the

`showData()`

function was created to show the data, the

`request()`

function to send requests, the

```
createFolder(); delete_file();delete_file();
upload_file();changeFolder()
```

functions to interact with the files and also the

`back()`

function to return to the previous file. The front end of this functionality was developed with Javascript, and DOM methods were used.

4.8 Student Subscribe/Unsubscribe

Code located in Appendix A.7
A student can subscribe and unsubscribe the course , join a team or create his own team and download his CV. To do this, we used `home/views.py`

```
CommandView(),DeleteCommandView(),
StudentFollowView(),
StudentUnFollowCommandView(),
SubsribeCourse(),StudentUnFollowCourseView(),
render_pdf_view().
```

4.9 Student/Team Work View

Code located in Appendix A.8
In our system, the teacher and the company representatives will be able to see the uploaded work of the student.
If the course type is 'course' teachers and company representatives will be able to see the full list of participants and their completed work. If the course type is 'challenge' then the teachers and company representatives will be able to see a list of teams and by clicking on the view they can see all the information about the teams work. To do this, we used the

`TeacherWorkView(), CompanyWorkView()`

functions in `home/views.py`.

4.10 Student CV View

Code located in Appendix A.9
In our system, a teacher and a company representative can see the CV of a participating student. In `participant.html`, when you click on the name of the student, the `profile_for_view.html` opens. For this, we used `home/views.py` function

`TeacherCvView(), CompanyCvView()`

Chapter 5

Conclusion

The development of the platform was a great responsibility for us, as it is a very voluminous and complex work. Of course, there were mistakes and problems, but solving and overcoming all the problems, we learned more and more every day.

This project was, first of all, devoted to the purposeful improvement of the educational process for the students of our faculty.

We believe that our project will continue and help improve the quality of e-education in our country.

Appendix A

Code Examples

A.1 Authorization

```
class CustomBackend(ModelBackend):
    def authenticate(self, request, username=
    None, password=None):
        try:
            user = User.objects.get(email=username)
        except User.DoesNotExist:
            return None
        else:
            if user.check_password(password):
                return user
        return None

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        email = request.POST['email']
        if User.objects.filter(email=email).exists():
            messages.error(request,
            'User already exits!')
        elif not (email.endswith('@stu.sdu.edu.kz')
        or email.endswith('@sdu.edu.kz')):
            messages.error(request,
            'Please use SDU email!')

        else:
            if form.is_valid():
                user = form.save()
                newusername = email.split("@")[0]
```

```
                user.username =
                f'{newusername}_{user.id}'
                user.save()
                if email.endswith('@stu.sdu.edu.kz'):
                    group = Group.objects.get(name='Students')
                    user.groups.add(group)
                    Students.objects.create(user=user)
                elif email.endswith('@sdu.edu.kz'):
                    group =
                    Group.objects.get(name='Teachers')
                    user.groups.add(group)
                    Teachers.objects.create(user=user)

                # login(request, user)
                messages.success(request,
                'Registration successfully completed!')
                return redirect('login')
            else:
                messages.error(request,
                'Wrong in registration!')
    else:
        form = UserRegisterForm()
    return render(request,
    'accounts/registration.html', {"form": form})

def user_login(request):
    if request.method == 'POST':
        form = UserLoginForm(data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            if user.groups.filter(name=
            'Students').exists():
                return redirect('my_courses')
            elif user.groups.filter(name=
            'Teachers').exists():
                return redirect('teacher-progress')
            else:
                messages.error(request,
                'Please, use SDU email!')
                return redirect('login')
    else:
        form = UserLoginForm()
```

```
    return render(request,
    'accounts/login.html', {"form": form})


def company_login(request):
    if request.method == 'POST':
        form = UserLoginForm(data=
        request.POST)
        if form.is_valid():
            user = form.get_user()
            try:
                user = User.objects.get
                (username=user.username)
            except user.DoesNotExist:
                return redirect('com_login')
            login(request, user)
            if user.groups.filter(name=
            'Company members').exists():
                return redirect('company_course')
            else:
                messages.error(request,
                'You do not have access,
                please contact the SDU staff!')
                return redirect('com_login')
    else:
        form = UserLoginForm()
    return render(request,
    'accounts/company_login.html', {"form": form})
```

A.2 Course View

```
@auth_check
@allowed_users(allowed_roles=['Students'])
def ChallengeDetailView(request, course_id,
command_id):
    course = Courses.objects.get
    (course_id=course_id)
    task = Task.objects.get(course_id=
    course_id)
    student = Students.objects.get
    (user=request.user)
    command = Command.objects.get
```

```
(command_id=command_id)
members = CommandMember.
objects.filter(command=command)
if request.method=='POST':
    count=0
    list_role=request.POST.getlist('role_name')
    for item in members:
        CommandMember.objects.filter
        (student=item.student).update(role=list_role[count])
        count=count+1
    return redirect('challenge_detail',
    course_id=course_id,
    command_id=command.command_id)
if Command.objects.filter
(command_id=command_id, created_by=student):
    try:
        project = Project.objects.
        get(task=task, student=
        command.created_by, course=course)
        root = Folder.objects.get
        (project=project, parent_id=None)
    except Project.DoesNotExist:
        project = Project(task=task,
        student=command.created_by, course=course)
        project.save()
        root = Folder(project=project,
        name=str(project.id))
        mkdir(f"storage/{root.name}")
        root.save()
    context = {'members': members,
               'course': course,
               'command': command,
               'student': student,
               'task': task,
               'command_id':
               command.command_id,
               'project_id': project.id,
               'root_id': root.id
               }
    return render(request,
    'student/command_detail.html', context=context)
else:
    project = Project.objects.get
```

```
        (task=task, student=command.created_by, course=course)
        root = Folder.objects.get
        (project=project, parent_id=None)
        context = {'members': members,
                   'course': course,
                   'command': command,
                   'student': student,
                   'task': task,
                   'command_id': command.command_id,
                   'project_id': project.id,
                   'root_id': root.id
                   }
        return render(request,
        'student/command_detail.html', context=context)

########################COMPANY#################################

@auth_check
@allowed_users(allowed_roles=['Company members'])
def CompanyCourseView(request):
    user = request.user
    company_member = CompanyMember.objects.get(user=user)
    lis_courses= AccessTable.objects.
    filter(company_member=company_member)
    list_course2 = [course.course.course_id
    for course in lis_courses]
    courses=Courses.objects.filter
    (course_id__in=list_course2)
    categories = CourseCategory.objects.all()
    education_types =
    Education_type.objects.all()
    if request.method =='POST':
        filter_categories =
        [category for category in CourseCategory.objects.all()
               if category.category in request.POST.keys()]
        if not filter_categories:
            filter_categories =
            CourseCategory.objects.all()
        filter_education_types =
        [ed_type for ed_type in Education_type.objects.all()
                                  if
        ed_type.education_type_name in request.POST.keys()]
        if not filter_education_types:
```

```
            filter_education_types =
            Education_type.objects.all()
        courses = list(filter(lambda course:
        course.education_type in
        filter_education_types and
       course.category in filter_categories,
                              courses))
        context = {
                    'courses': courses,
                    'categories': categories,
                    'education_types': education_types
             }
        return render(request,
        'company/available_course_com.html', context=context)

    if 's' in request.GET:
        courses = courses.filter(course_name__icontains=
        request.GET.get('s'))
    context = {
        'courses': courses,
        'categories': categories,
        'education_types': education_types
    }
    return render(request,
    'company/available_course_com.html', context=context)


@auth_check
@allowed_users(allowed_roles=['Company members'])
def CompanyTaskDetailView(request, course_id):
    course = Courses.objects.get(course_id=course_id)
    tasks = Task.objects.filter(course=course)
    if course.education_type.education_type_id == 2:
        return redirect('company_detail_challenge_view'
        , course_id=course_id)
    else:
        teacher = course.teacher
        task = tasks.first()
        context = {'course': course, 'tasks': tasks,
        'course_id': course_id, 'task': task, 'teacher':teacher}
        return render(request,
        'company/task_list_main.html', context=context)
```

```
@auth_check
@allowed_users(allowed_roles=['Company members'])
def CompanyDetailTaskView(request, course_id, task_id):
    course =Courses.objects.get(course_id=course_id)
    tasks = Task.objects.filter(course=course)
    teacher =course.teacher
    task= Task.objects.get(task_id=task_id)
    context={'course':course, 'tasks': tasks,
    'course_id':course_id, 'task': task, 'teacher':teacher}
    return render(request, 'company/task_list.html',
    context=context)


@auth_check
@allowed_users(allowed_roles=['Company members'])
def CompanyDetailChallengeView(request, course_id):
    course = Courses.objects.get(course_id=course_id)
    commands = Command.objects.filter(challenge=course)
    return render(request,
    'company/command_list.html', {'commands': commands,
    'course': course})


@auth_check
@allowed_users(allowed_roles=['Company members'])
def CompanyDetailCommandView(request, command_id):
    command = Command.objects.get(command_id=command_id)
    course = command.challenge
    teacher = course.teacher
    task = Task.objects.filter(course=course).first()
    members = CommandMember.objects.
    filter(command=command)
    context = {
        'command': command,
        'created_by': command.created_by,
        'members': members,
        'teacher': teacher,
        'course': course,
        'task': task,
        'command_id': command_id
    }
    return render(request, 'company/
```

```
    command_detail.html', context=context)

@auth_check
@allowed_users(allowed_roles=['Company members'])
def CompanyTaskParticipantsView
(request, course_id, task_id):
    course = Courses.objects.get(course_id=course_id)
    task = Task.objects.get(task_id=task_id)
    teacher=course.teacher
    participant = Registration.objects.
    filter(course=course)
    context= {'course':course, 'task': task,
    'teacher':teacher, 'participant':participant}
    return render (request, 'company/
    participant.html', context=context)

######################TEACHER###################################
@auth_check
@allowed_users(allowed_roles=['Teachers'])
def DetailCourseView(request, course_id):
    course =Courses.objects.get(course_id
    =course_id)
    tasks = Task.objects.filter(course=course)
    if course.education_type.education_type_id == 2:
        return redirect('detail_challenge_view',
        course_id=course_id)
    else:
        teacher = course.teacher
        task = tasks.first()
        context={'course':course, 'tasks':
        tasks, 'course_id': course_id, 'task':
        task, 'teacher':teacher}
        return render(request, 'teacher/
        task_list_main.html', context=context)

@auth_check
@allowed_users(allowed_roles=['Teachers'])
def DetailChallengeView(request, course_id):
    course = Courses.objects.get(course_id=
    course_id)
    commands = Command.objects.
    filter(challenge=course)
    return render(request, 'teacher/
```

```
    command_list.html', {'commands':
    commands, 'course': course})
@auth_check
@allowed_users(allowed_roles=['Teachers'])
def DetailTaskView(request, course_id, task_id):
    course =Courses.objects.get(course_id=
    course_id)
    tasks = Task.objects.filter(course=course)
    teacher =course.teacher
    task= Task.objects.get(task_id=task_id)
    context={'course':course, 'tasks': tasks,
    'course_id':course_id, 'task': task, 'teacher':teacher}
    return render(request, 'teacher/
    task_list.html', context=context)

@auth_check
@allowed_users(allowed_roles=['Teachers'])
def DetailCommandView(request, command_id):
    command = Command.objects.
    get(command_id=command_id)
    teacher = Teachers.objects.get(user=request.user)
    course = command.challenge
    task = Task.objects.filter(course=course).first()
    members = CommandMember.objects.
    filter(command=command)
    context = {
        'command':command,
        'created_by':command.created_by,
        'members':members,
        'teacher':teacher,
        'course':course,
        'task':task,
        'command_id':command_id
    }
    return render(request, 'teacher/
    command_detail.html', context=context)
```

A.3 Filtering and Search Bar

```
####################Student's Filter##################
@auth_check
@allowed_users(allowed_roles=['Students'])
```

```
def AvailableCourseView(request):
    if request.method =='POST':
        courses= Courses.objects.filter(end_time__gte
        =date.today())
        user = request.user
        student = Students.objects.get(user=user)
        reg_courses = Registration.objects.filter
        (student=student)
        list_id = []
        for item in reg_courses:
            list_id.append(item.course.course_id)
        categories = CourseCategory.objects.all()
        education_types =
        Education_type.objects.all()
        filter_categories = [category for
        category in CourseCategory.objects.all()
                             if category.category
                             in request.POST.keys()]
        if not filter_categories:
            filter_categories =
            CourseCategory.objects.all()
        filter_education_types = [ed_type
        for ed_type in Education_type.objects.all()
                                  if ed_type.education_type_name
                                  in request.POST.keys()]
        if not filter_education_types:
            filter_education_types =
            Education_type.objects.all()
        courses = list(filter(lambda course:
        course.education_type in filter_education_types and
                         course.category
                        in filter_categories, courses))
        context = {
            'reg_courses_id': list_id,
            'courses': courses,
            'categories': categories,
            'education_types': education_types,
            'selected_categories': filter_categories,
            'selected_education_types':
            filter_education_types
        }
        return render(request, 'student/
        availablecouse.html', context=context)
```

```
    if 's' in request.GET:
        courses = Courses.objects.
        filter(course_name__icontains=
        request.GET.get('s'), end_time__gte=date.today())
    else:
        courses =  Courses.objects.filter
        (end_time__gte=date.today())
        courses = courses.order_by('-start_time')
    user = request.user
    student = Students.objects.get(user=user)
    reg_courses = Registration.objects.filter
    (student=student)
    list_id = []
    for item in reg_courses:
        list_id.append(item.course.course_id)
    categories = CourseCategory.objects.all()
    education_types = Education_type.objects.all()
    context = {
           'reg_courses_id': list_id,
           'courses': courses,
           'categories': categories,
           'education_types': education_types,
            'selected_categories': [],
            'selected_education_types': []
    }
    return render(request, 'student/
    availablecouse.html', context=context)

######################Teacher's Filter#############

@auth_check
@allowed_users(allowed_roles=['Teachers'])
def TeacherAvCourseView(request):
    teacher = Teachers.objects.get(user=request.user)
    courses = Courses.objects.filter(teacher=teacher)
    categories = CourseCategory.objects.all()
    education_types = Education_type.objects.all()
    if request.method == 'POST':
        filter_categories = [category for category
        in CourseCategory.objects.all()
                             if category.category in
                             request.POST.keys()]
```

```
        if not filter_categories:
            filter_categories = CourseCategory.
            objects.all()
        filter_education_types = [ed_type for
        ed_type in Education_type.objects.all()
                                  if ed_type.education_
                                  type_name in request
                                  .POST.keys()]
        if not filter_education_types:
            filter_education_types =
            Education_type.objects.all()
        courses = list(filter(lambda course:
        course.education_type in filter_
        education_types and
          course.category
          in filter_categories, courses))
        context = {
            'courses': courses,
            'categories': categories,
            'education_types': education_types
        }
        return render(request, 'teacher/
        availablecourse3.html', context=context)

    if 's' in request.GET:
        courses = courses.filter
        (course_name__icontains=request.GET.get('s'))
    context = {
        'courses': courses,
        'categories': categories,
        'education_types': education_types
    }
    return render(request, 'teacher/
    availablecourse3.html', context=context)

################Company's Filter#################
@auth_check
@allowed_users(allowed_roles=
['Company members'])
def CompanyCourseView(request):
    user = request.user
    company_member = CompanyMember.
    objects.get(user=user)
```

```
lis_courses= AccessTable.objects.
filter(company_member=company_member)
list_course2 = [course.course.course_id
for course in lis_courses]
courses=Courses.objects.filter(course_id__in
=list_course2)
categories = CourseCategory.objects.all()
education_types = Education_type.objects.all()
if request.method =='POST':
    filter_categories = [category for category
    in CourseCategory.objects.all()
                         if category.category in
                         request.POST.keys()]
    if not filter_categories:
        filter_categories = CourseCategory.
        objects.all()
    filter_education_types = [ed_type
    for ed_type in Education_type.objects.all()
                              if ed_type.education_
                              type_name in request.
                              POST.keys()]
    if not filter_education_types:
        filter_education_types =
        Education_type.objects.all()
    courses = list(filter(lambda course:
    course.education_type in filter_education_types and
          course.category in filter_categories,
            courses))
    context = {
                'courses': courses,
                'categories': categories,
                'education_types': education_types
         }
    return render(request, 'company/
    available_course_com.html', context=context)

if 's' in request.GET:
    courses = courses.filter(course_name
    __icontains=request.GET.get('s'))
context = {
    'courses': courses,
    'categories': categories,
    'education_types': education_types
```

```
    }
    return render(request, 'company/
    available_course_com.html', context=context)
```

A.4 Teacher's functionalities (create/edit/delete)

```
@auth_check
@allowed_users(allowed_roles=['Teachers'])
def TeacherCreateCourseView(request):
    teacher = Teachers.objects.get
    (user=request.user)
    if request.method == 'POST':
        form = CourseCreationForm
        (request.POST)
        if form.is_valid():
            course=Courses.objects.
            create(teacher=teacher,
    course_name=request.POST['course_name'],
    course_code=request.POST['course_code'],
    course_descrip =request.POST['course_descrip'],
    start_time=request.POST['start_time'],
    end_time= request.POST['end_time'],
    course_key= request.POST['course_key'],
   category=CourseCategory.objects.get
                                    (category_id=request.
                                    POST['category']),
    education_type=Education_type.objects.
    get(education_type_id =request.
    POST['education_type'])
                                    )
            number =request.
            POST['education_type']
            if number == '2':
                Task.objects.create
                (course=course, task_name
                ='You task name')
            return redirect('teacher-progress')
    else:
        form = CourseCreationForm()
    return render(request, 'teacher/
    teacheradd.html', {'form': form})
```

```
@auth_check
@allowed_users(allowed_roles=
['Teachers'])
def DeleteCourseView(request, course_id):
    Courses.objects.get(course_id=
    course_id).delete()
    return redirect('teacher-progress')

@auth_check
@allowed_users(allowed_roles=
['Teachers'])
def CraeteTaskView(request, course_id):
    course = Courses.objects.get
    (course_id=course_id)
    tasks = Task.objects.filter(course_id
    =course)
    if request.method == 'POST':
        course = Courses.objects.get
        (course_id=course_id)
        form = TasksForm(request.POST)
        if form.is_valid():
            # print(form.cleaned_data)
            Task.objects.create(course
            =course,
                                task_name=
                                request.POST['task_name'],
                                end_time=
                                request.POST['end_time'],
                                task_descrip
                                =request.POST
                                ['task_descrip'])
            return redirect('task-view',
            course_id=course_id)
    else:
        form = TasksForm()
    return render(request, 'teacher/
    task_creation.html', {'form': form,
    'course': course})


@auth_check
@allowed_users(allowed_roles=
['Teachers'])
```

```
def UpdateTaskView(request,
course_id, task_id):
    course = Courses.objects.get
    (course_id=course_id)
    task = Task.objects.get(task_id
    =task_id)
    if request.method == 'POST':
        form = TasksForm(request.
        POST)
        if form.is_valid():
            # print(form.cleaned_data)
            Task.objects.filter(task_id
            =task_id).update(
                                task_name=
                                request.POST
                                ['task_name'],
                                end_time=request.
                                POST['end_time'],
                                task_descrip=
                                request.POST['task_descrip'])
            return redirect('task-view',
            course_id=course_id)
    else:
        form = TasksForm()
    return render(request, 'teacher/
    task_update.html', {'form': form,
    'course': course, 'task': task})


@auth_check
@allowed_users(allowed_roles=['Teachers'])
def DeleteTaskView(request, course_id, task_id):
    task = Task.objects.get(task_id=task_id, course=
    Courses.objects.get(course_id=course_id))
    task.delete()
    return redirect('task-view', course_id=course_id)

@auth_check
@allowed_users(allowed_roles=['Teachers'])
def TeacherCourseView(request, course_id):
    course = Courses.objects.get(course_id
    =course_id)
    if request.method == 'POST':
```

```
        update_form =CourseUpdateForm
        (request.POST, instance=course)
        if update_form.is_valid():
            update_form.save()
            return redirect('course-view',
            course_id=course_id)
    member_form =
    SelectCompanyMembersForm()
    update_form =
    CourseUpdateForm(instance=course)
    members = AccessTable.
    objects.filter(course=course)
    form = GiveAccessForm()
    user_form = UserRegisterForm()
    context = {'course': course,
               'access_members': members,
               'form' : form,
               'user_form' : user_form,
               'update_form' : update_form,
               'member_form': member_form}
    return render(request,
    'teacher/course.html', context=context)
```

A.5 Teacher's functionality (giving access to companies)

```
@auth_check
@allowed_users(allowed_roles=
['Teachers'])
def TeacherAddMemberForCourseView
(request, course_id):
    course = Courses.objects.
    get(course_id=course_id)
    if request.method == 'POST':
        member_form =
        SelectCompanyMembersForm
        (request.POST)
        if member_form.is_valid():
            access = AccessTable.objects.
            filter(course=course, company_member
            =CompanyMember.
            objects.get(member_id=
```

```
            request.POST['members']))
            if access:
                pass
            else:
                AccessTable.objects.create
                (course=course, company_member
                =CompanyMember.
                objects.get(member_id=request.
                POST['members']))
                return redirect('course-view',
                course_id=course_id)
    member_form = SelectCompany
    MembersForm()
    update_form = CourseUpdateForm
    (instance=course)
    members = AccessTable.objects.
    filter(course=course)
    form = GiveAccessForm()
    user_form = UserRegisterForm()
    context = {'course': course,
               'access_members': members,
               'form': form,
               'user_form': user_form,
               'update_form': update_form,
               'member_form': member_form}
    return render(request, 'teacher/
    course.html', context=context)


@auth_check
@allowed_users(allowed_roles=
['Teachers'])
def TeacherAddNewMemberFor
CourseView(request, course_id):
    course = Courses.objects.get
    (course_id=course_id)
    if request.method == 'POST':
        user_form= UserRegisterForm
        (request.POST)
        form = GiveAccessForm
        (request.POST)
        if user_form.is_valid() and
        form.is_valid():
```

```
            email=request.POST['email']
            if User.objects.filter(email=
            email).exists():
                messages.error(request,
                'User already exits!')
                return redirect('course-view',
                course_id=course_id)
            else:
                newusername = email.split("@")[0]
                user = user_form.save()
                user.username = f'
                {newusername}_{user.id}'
                user.save()
                group = Group.objects.get
                (name='Company members')
                user.groups.add(group)
                company_member=
                CompanyMember.objects.
                create(user=user,
                company_name
                =request.POST['
                company_name'])
                AccessTable.objects.create
                (course=course, company_
                member=company_member)
                return redirect('course-view',
                course_id=course_id)
    else:
        member_form =
        SelectCompanyMembersForm()
        update_form = CourseUpdateForm
        (instance=course)
        members = AccessTable.objects.
        filter(course=course)
        form = GiveAccessForm()
        user_form = UserRegisterForm()
        context = {'course': course,
                   'access_members': members,
                   'form' : form,
                   'user_form' : user_form,
                   'update_form' : update_form,
                   'member_form':
                   member_form}
```

```
        return render(request, 'teacher/
        course.html', context=context)
```

A.6 Student Upload File

```
<script>
        let host = window.location.protocol + "//"
        + window.location.host;
        let folders = document.querySelector("#folders");
        let files = document.querySelector("#files");
        let folder = {{ root_id }};
        let parent_folder;
        let path = document.querySelector("#path");
        let file = document.querySelector("#fileupload");
        let folder_name = document.querySelector("#folder_name");
        let reader  = new FileReader();

        function showData(data){
            if (folder !== {{ root_id }}){
                folders.innerHTML = "<i id='back' onclick='back()'
                class=\"bi bi-arrow-left\"></i>";
            } else {
                folders.innerHTML = "";
            }
            files.innerHTML = "";
            for (let i=0; i<data.files.folders.length;i++){
                folders.innerHTML+=`<div class='folder'>
                <div style="display: inline-block">
                ${data.files.folders[i].name}
                </div>
<div class="btn" onclick=
"delete_folder(${data.files.folders[i].id})">
<i class="bi bi-trash"></i></div>
<div class="btn" onclick=
"change_folder(${data.files.folders[i].id})">
<i class="bi bi-folder-symlink"></i></div>
</div>`
            }
            for (let i=0; i<data.files.files.length;i++){
                files.innerHTML+=`<div class='file'>
                <div style="display: inline-block">
                ${data.files.files[i].name}</div>
```

```
<div class="btn" onclick="delete_file(${data.files.files[i].id})">
<i class="bi bi-trash"></i></div>
<div class="btn" onclick="window.location =
'${data.files.files[i].download}'">
<i class="bi bi-file-earmark-arrow-down"></i></div>
</div>`
            }
            path.innerHTML = data.files.path;

        }

        function request(url, data, handler){
            data["csrfmiddlewaretoken"] = "{{ csrf_token }}"
            $.ajax({
                url:url,
                type:'POST',
                data:data,
                success:function(data){
                    handler(data);
                },
                error:function(error){
                    console.log(error);
                }
            });
        }

        function createFolder(){
            if (folder_name.value !== ""){
                request(`${host}/filesystem/new_folder`,
                {"parent": folder, "name": folder_name.value},
                function (){
                    request(`${host}/filesystem/get_folder`,
                    {"folder": folder}, showData)
                })
            }
        }

        function upload_file(){
            let data = file.files[0];
            reader.onloadend = function () {
                request(`${host}/filesystem/new_file`,
                {"folder": folder, "name": data.name,
                        "data": reader.result}, function (){
```

```
                request(`${host}/filesystem/get_folder`,
                {"folder": folder}, showData)
            })
        };
        reader.readAsDataURL(data);
    }

    function change_folder(folder_id){
        folder = folder_id;
        request(`${host}/filesystem/get_folder`,
        {"folder": folder_id}, function (data){
            parent_folder = data.files.parent;
            showData(data);
        })
    }

    function back(){
        change_folder(parent_folder)
    }

    function delete_folder(folder_id){
        request(`${host}/filesystem/delete_folder`,
        {"folder": folder_id}, function (){
                request(`${host}/filesystem/get_folder`,
                {"folder": folder}, showData)
            })
    }

    function delete_file(file_id){
        console.log(file_id)
        request(`${host}/filesystem/delete_file`,
        {"file": file_id}, function (){
                request(`${host}/filesystem/get_folder`,
                {"folder": folder}, showData)
            })
    }

    request(`${host}/filesystem/get_project`,
    {"project": {{project_id}}}, showData)
</script>
```

A.7 Student Subscribe/Unsubscribe

```
@auth_check
@allowed_users(allowed_roles=['Students'])
def render_pdf_view(request):
    template_path = 'student/pdf_portfolio.html'
    user = request.user
    student = Students.objects.get(user=user)
    context = {'user': user,
               'student': student}
    response = HttpResponse(content_type='application/pdf')
    response['Content-Disposition'] = 'filename="my_cv.pdf"'
    template = get_template(template_path)
    html = template.render(context)
    pisa_status = pisa.CreatePDF(html, dest=response)
    if pisa_status.err:
       return HttpResponse('We had some errors <pre>'
       + html + '</pre>')
    return response

@auth_check
@allowed_users(allowed_roles=['Students'])
def SubcribeCourse(request, course_id):
        user = request.user
        course = Courses.objects.get(course_id=course_id)
        student = Students.objects.get(user=user)
        Registration.objects.create(student=student,
        course=course)
        return AvailableCourseView(request)

@auth_check
@allowed_users(allowed_roles=['Students'])
def StudentUnFollowCourseView(request, course_id):
    student = Students.objects.get(user=request.user)
    course = Courses.objects.get(course_id=course_id)
    Registration.objects.get(student=student,
    course=course).delete()
    return redirect('my_courses')

@auth_check
@allowed_users(allowed_roles=['Students'])
def CreateCommandView(request, course_id):
    course = Courses.objects.get(course_id=course_id)
```

```
    student = Students.objects.get(user=request.user)
    form = CommandCreationForm()
    commands = Command.objects.filter(challenge=course)
    if request.method=='POST':
        form = CommandCreationForm(request.POST)
        if form.is_valid():
            command_name = request.POST['command_name']
            command=Command.objects.create
            (challenge=course, command_name
            =command_name, created_by=student)
            CommandMember.objects.create
            (student=student, command=command)
            return redirect('challenge_detail',
            course_id=course_id, command_id=
            command.command_id)
    return render(request,
    'student/command_list.html', {'commands':
    commands, 'course': course, 'form': form})

@auth_check
@allowed_users(allowed_roles=['Students'])
def StudentFollowView(request, command_id):
    command = Command.objects.get
    (command_id=command_id)
    course=command.challenge
    student =Students.objects.
    get(user=request.user)
    CommandMember.objects.create
    (student=student, command=command)
    return StudentCourseView
    (request, course.course_id)

@auth_check
@allowed_users(allowed_roles=['Students'])
def StudentUnFollowCommandView(request, command_id):
    command = Command.objects.get(command_id=command_id)
    course=command.challenge
    student =Students.objects.get(user=request.user)
    CommandMember.objects.get
    (student=student, command=command).delete()
    return StudentCourseView(request, course.course_id)

@auth_check
```

```
@allowed_users(allowed_roles=['Students'])
def DeleteCommandView(request, course_id, command_id):
    command = Command.objects.get
    (command_id=command_id)
    project = Project.objects.get
    (student=command.created_by, course=command.challenge)
    root = Folder.objects.get(project=project, parent_id=None)
    project.delete()
    root.delete()
    Command.objects.get(command_id=command_id).delete()
    return redirect('create_command', course_id=course_id)
```

A.8 Student/Team Work View

```
@auth_check
@allowed_users(allowed_roles=['Teachers'])
def TeacherWorkView(request,student_id,course_id, task_id ):
    course = Courses.objects.get(course_id=course_id)
    task = Task.objects.get(task_id=task_id)
    try:
        project = Project.objects.
        get(task=task, student=Students.objects.
        get(student_id=student_id), course=course)
        root = Folder.objects.get(project=project,
        parent_id=None)
    except Project.DoesNotExist:
        context = {'course': course, 'task': task,
        'project_id': 'none'}
        return render(request, 'teacher/
        work_view.html', context=context)
    context = {'course': course, 'task': task,
    'project_id': project.id, "root_id": root.id}
    return render(request, 'teacher/work_view.html',
    context=context)

@auth_check
@allowed_users(allowed_roles=['Company members'])
def CompanyWorkView(request,student_id,
course_id, task_id ):
    course = Courses.objects.get(course_id=
    course_id)
    task = Task.objects.get(task_id=task_id)
```

```
    try:
        project = Project.objects.get(task=task, student=
        Students.objects.get(student_id=
        student_id), course=course)
        root = Folder.objects.get(project=project,
        parent_id=None)
    except Project.DoesNotExist:
        context={'project_id': 'none'}
        return render(request, 'company/
        work_view.html', context=context)
    context = {'course': course, 'task': task,
    'project_id': project.id, "root_id": root.id}
    return render(request, 'company/
    work_view.html', context=context)
```

A.9 Student CV View

```
@auth_check
@allowed_users(allowed_roles=['Teachers'])
def TeacherCvView(request,student_id):
    student = Students.objects.get(student_id
    =student_id)
    context = {'student': student}
    return render(request, 'teacher/
    profile_for_view.html', context=context)

@auth_check
@allowed_users(allowed_roles=
['Company members'])
def CompanyCvView(request, student_id):
    student = Students.objects.get
    (student_id=student_id)
    return render(request,
    'company/profile_for_view.html',
    {'student': student})
```

References

[1] *Django documentation*. URL: https://docs.djangoproject.com/en/3.2/.

[2] Imarkey. *Evolution of online learning*. URL: https://www.timetoast.com/timelines/evolution-of-online-learning-47bbe2be-e7f6-4f2a-bc20-164311118628.

[3] Dawid Karczewski. *Django*. URL: https://www.ideamotive.co/blog/amazing-django-website-development-examples-you-should-see-right-now.

[4] Igor M.S. *1 янв 1840 г. - Исаак Питман начал обучать студентов стенографии*. URL: https://time.graphics/ru/event/1888146.

[5] David W. Smith. *THE COLOR OF SAFETY*. URL: http://agrilife.org/agsafety/files/2011/06/THE-COLOR-OF-SAFETY1.pdf.