



Project title: Web platform for live lectures

Final report

CM3203 One Semester Individual Project

Andon Gribachev: C1824840

Supervised by: Dr. Daniela Tsaneva

Moderated by: Dr. Paul L Rosin

Abstract:

Ever since the pandemic outbreak, CO-VID 19 impacted the lives of every person around the globe. Having said that, universities had to adapt to remote teaching and respectively the students had to adapt to the remote learning process. Many universities already had student web platforms where it allowed for a mixed way of teaching. However, some universities struggled a lot as they had not had an implemented system, which caused severe problems in how the teaching process would be delivered during these pandemic times. The project I chose aims to provide an interactive platform for both the students and the lecturers, where lectures could be streamed and recorded for later view with a chatbot that would be able to answer different queries the users might have.

The project is detailed with background research, design, implementation, testing and evaluation of the platform built into a Django REST framework. Also, user test cases were evaluated and conducted, whilst the platform itself was tested with unit tests. Having said that, both the strengths and the weaknesses of the platform were evaluated using these tests.

Acknowledgements:

I would personally like to thank my supervisor Dr Daniela Tsaneva. The given guidance and support to the overall project, as well as me, was critical for the progress made over the projects' duration.

Also, I would like to thank my friend Georgi Vasilev for providing me with resources (lectures) on how the front-end should be designed interactively. That helped me significantly during the development of the front-end of the platform as I previously had zero to none of experience designing interfaces.

Table of Contents

Introduction	6
Background	7
Existing solutions	7
OBS paired with YouTube	7
User Research – Personas	10
Personas justification:	11
Project justification	11
Project Constraints	11
Approach	13
Agile development methodology	13
Learning	13
Django framework:	13
Data storage:	14
Chatbot:	14
Specification	15
Functional requirements	15
Non-functional requirements	17
Use cases:	19
Use case diagram – Django platform	19
Use cases – Django platform	20
Use case diagram – Chatbot	32
Use cases – Chatbot	33
Design	36
UML Class Diagram	36
Overview of the UML Class Diagram:	37
User interface prototypes	38
Initial prototypes:	38
Implementation	51
Database design	51
ER (Entity Relationship) Diagram - Pages	51
Description of the ER Diagram – Pages	52
ER (Entity Relationship) Diagram – Sessions	58

Description of the ER Diagram – Sessions	58
ER (Entity Relationship) Diagram - Authentication	59
Description of the ER Diagram – Authentication	60
ER (Entity Relationship) Diagram – Content types.....	60
Description of the ER Diagram – Content types	60
Activity diagrams.....	61
User logging in (Both Student and Lecturer interface)	61
Student interface	62
Lecturer interface.....	62
Overview of the design and implementation practices.....	64
Detailed implementation	65
The Django platform	65
The chatbot	86
Test cases	96
Function requirements test cases.....	96
Non-functional requirements test cases	100
Comparing my solution to already existing ones:.....	101
Conclusion and Future work	102
Innovation	103
Reflection on the learning process	103
References	105

Introduction

Whilst I stayed in Bulgaria, I have identified certain issues with the platforms that Bulgarian universities use for their lectures and live sessions. Some universities do use Teams in collaboration with their platforms, however, some universities struggle in the teaching process and are finding it hard to deliver a proper learning process for the students. That motivated me to create a student web portal for live streamable lectures with featured past recordings that would be suitable both for the lecturers and the students.

The main objectives for developing the platform are the following:

- Having two separate user interfaces (UI) for lecturers and students
- The lecturers should be able to live-stream lectures
- The module-specific live-streamed lectures should be visible to, only the enrolled to the course the module is in, students
- The lecturer should be able to both automatically and manually upload past recordings
- The past recordings should be available to the students that are enrolled to a course that the module is in
- There should be discussion board specific to every module
- There should be chatrooms that could be booked by students to meet with the specific lecturer if needed. These chatrooms should only be accessible by the student if the specific lecturer has approved of such meeting. If approved, unique chatroom should be initiated
- There should be chatbot that would interactively answer questions of the students (limited to 5 use-cases)
- There should be dynamically generated module-specific self-assessments for the students

The web platform intends to be user-friendly and effective user design principles would be used to create a design base for the implementation process.

The report would include all information regarding processes such as designing, testing, evaluation, and implementation.

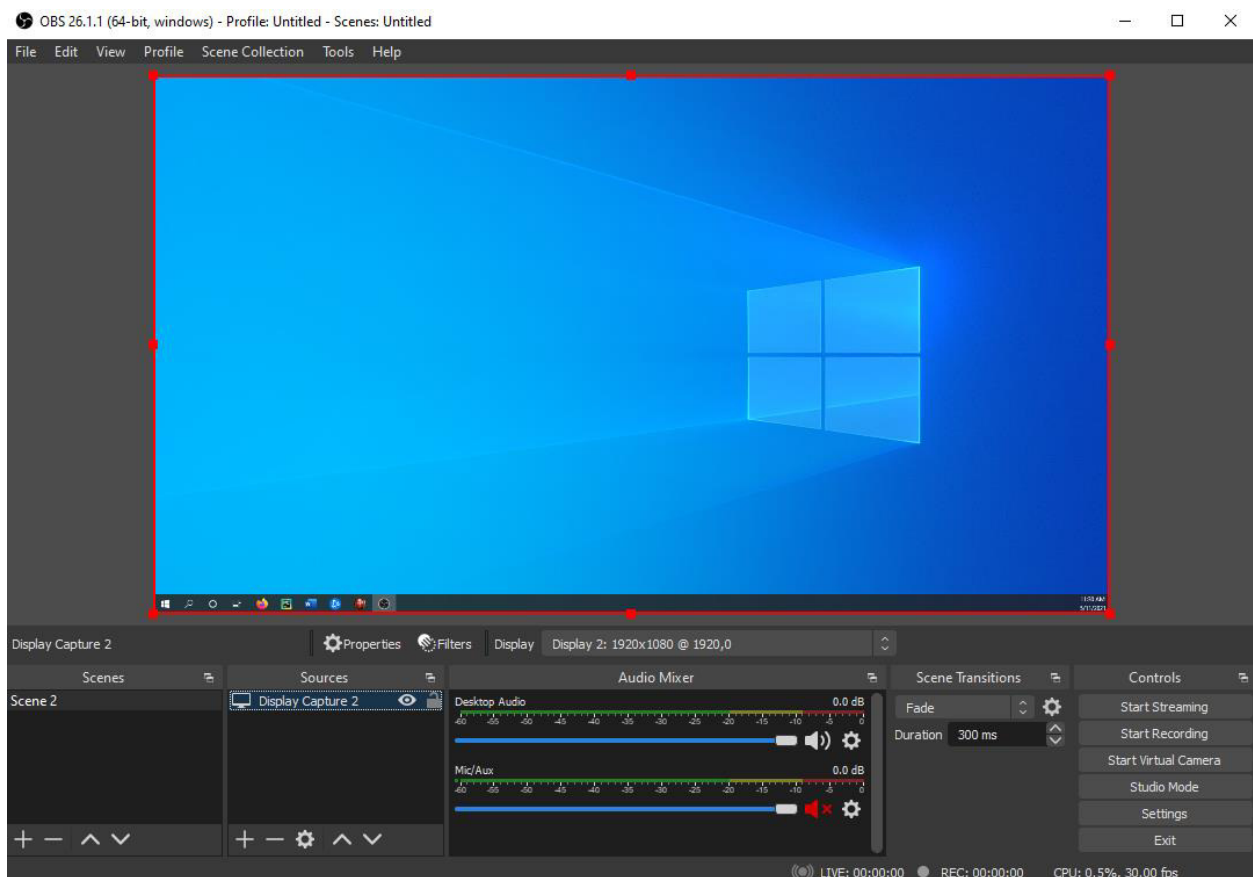
Background

Existing solutions

OBS paired with YouTube

Before I started working towards developing the platform, I had to research how the live-streaming platforms work.

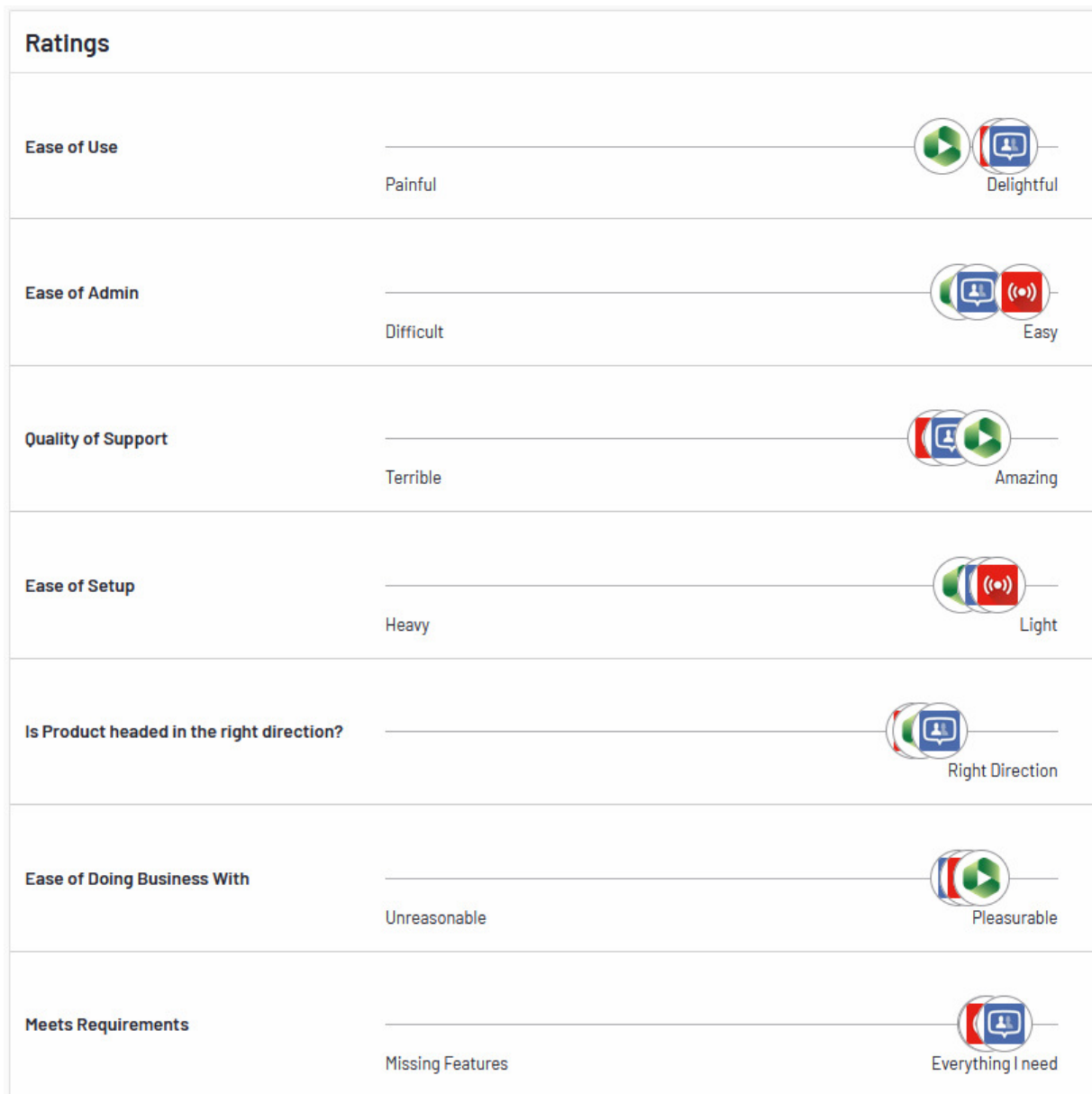
Initially, my research focused on the software that could broadcast live streamable video onto these platforms. Having said that, I found that the easiest in terms of usability and most used in that specific field was the OBS Studio. OBS Studio is a free and open-source recording and cross-platform streaming program.^[1]



Its interface allows the user to both record and stream video from different sources as captures and screens.

In order to stream the video, I had to research further onto different servers and live streaming platforms. In the following evaluation, I would go through the three main live streaming platforms (Facebook Live, YouTube Live, Panopto) and explain my decision in more detail ^[2]:

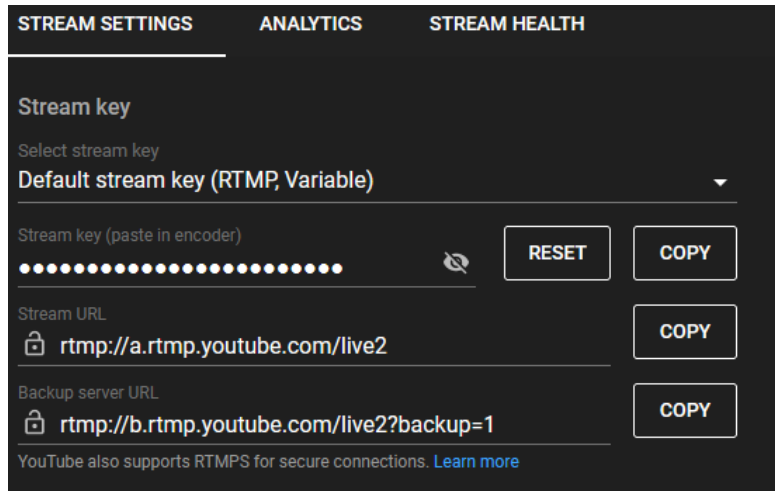
//Note: The statistics of user ratings could be created as on the website of reference [2] it has an option to compare different Livestream platforms.



One of the crucial features of the live streaming platform is that it should be easy to admin as I have no previous experience in live streaming video. Even though Facebook Live is rated easier to use and Panopto has a better quality of support by the rating, YouTube Live is rated very easy to set up.

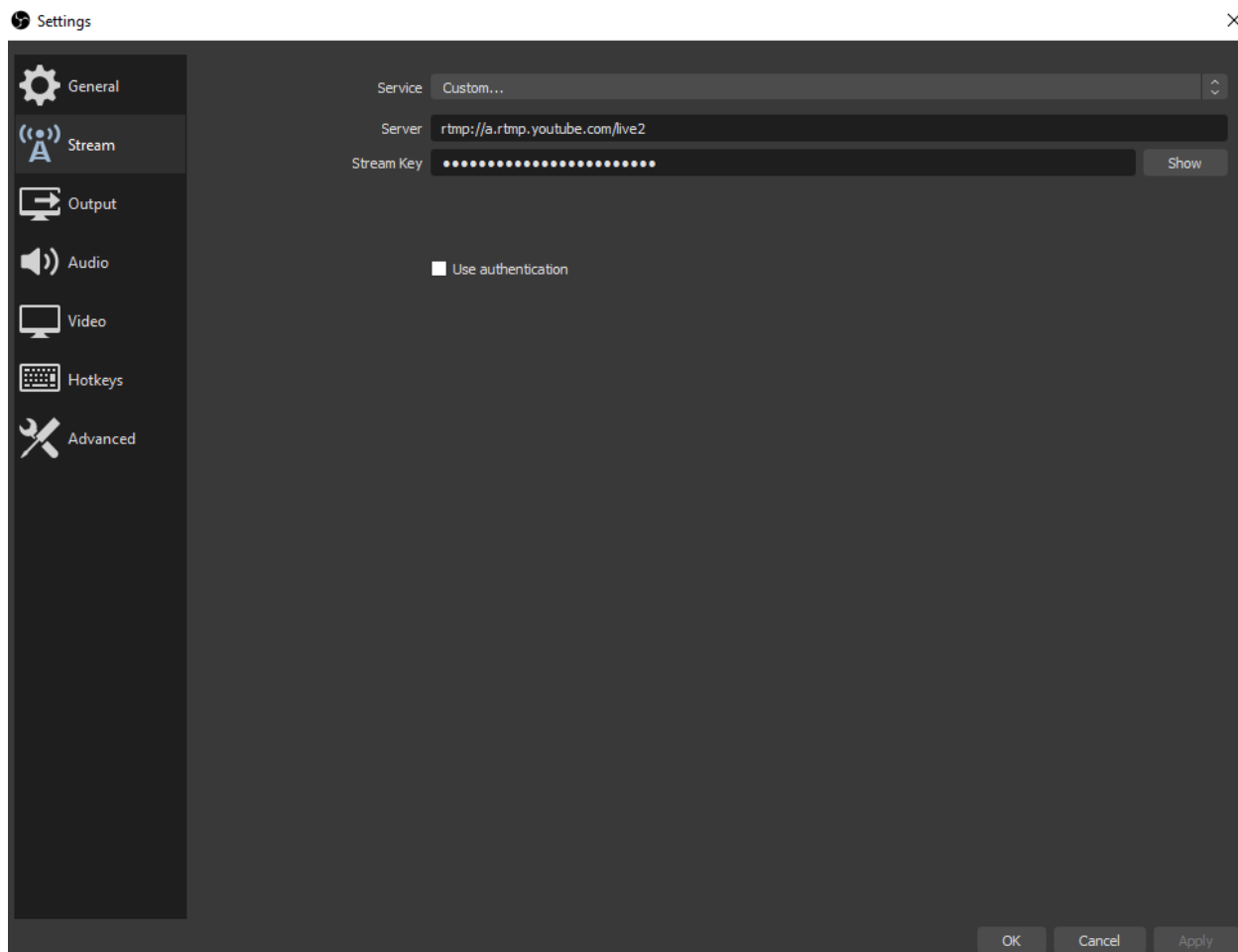
These two factors made YouTube Live the best choice for my student portal, as not only that, but it makes embedding the video much easier.

Furthermore, YouTube live allows the viewers to chat with the host of the stream and even though the chat itself is not embeddable it could still be seen as the tab of the Livestream should remain open whilst streaming video.





When initialising the live session at YouTube it gives the user a unique to the user stream key and a stream URL of the server.

Then the stream key and the Stream URL of the server are then to be inserted into the OBS Studio settings.



User Research – Personas

<p>Name: George Johnson</p> <p>Age: 21</p> <p>Technical competency:</p> <div><div></div></div>	
<p>Quote:</p> <p>"I would like my university to use some kind of portal where I would be able to find more resources and lectures that would help my learning process during these pandemic times."</p>	
<p>Description:</p> <p>George is in his final year studying Civil Engineering. He finds himself in a hard situation, since the covid outbreak, because resources are not as easily accessible for him. Lectures are presented in Zoom with no option to be seen afterwards, which makes it very hard for him since he is working a part-time job and is not always online for his lectures.</p>	
<p>Goals:</p> <ul style="list-style-type: none">• Would like to have more accessible resources on a web portal of the university• Would like to be able to watch lectures afterwards they are presented	

<p>Name: Jessica Morgan</p> <p>Age: 19</p> <p>Technical competency:</p> <div><div></div></div>	
<p>Quote:</p> <p>"Sometimes I do not understand the terminology set in my course, since I am a foreign student. It would be very helpful if there could be a chatroom meeting with lecturers one on one so I could ask questions privately to the professor."</p>	
<p>Description:</p> <p>Jessica is a foreign student studying biomedical science. She is the first year but has already found herself struggling with the learning process as due to the pandemic the resources for her modules are not as accessible and she is finding it hard to get through the learning curve.</p>	
<p>Goals:</p> <ul style="list-style-type: none">• Would like to have more accessible resources on a web portal of the university• Would like to speak one on one with a lecturer in a chatroom meeting	

Personas justification:

I decided to create personas to get a better understanding of the platform should perform. Visualising potential end-user is crucial for the further development of the web portal. By imagining the potential goals of the end-users, I got a good grasp of what the platform's functionality should be and what features could there be implemented. The interface should be quite user-friendly as the technical competency of the end-users might vary. Accessible resources are the priority of the project as they would allow the users (both students and lecturer) to have an easier time studying and teaching in those pandemic times.

Project justification

Whilst performing the background research I did investigated and understand that such platforms barely exist in Bulgaria and rarely integrated into Bulgarian universities.

In specific, Bulgarian Universities do use different resources such as Zoom, Discord and Teams for their regular lectures/meetings, but they do not have a platform that would be in favour of the students. For such platform to succeed and provide the students more pleasurable experience it would need to satisfy certain requirements:

- Clean minimalistic and intuitive design of the platform, so the application's interface is allowing easy usage for both the lecturers and the students.
- Flexibility in terms of that users might and should have different permissions (Students and Lecturers) from adding streamable lectures to past recordings

Project Constraints

During the final-year project undergoing I faced a couple of project constraints that might have affected the outcome of the project itself. I worked hard to minimise the impact of the following constraints.

Firstly, the impact the covid-19 outbreak had over the project. It made the developing and designing process a bit more diligent. It mainly impacted the usability testing and testing of the platform. Furthermore, I was not on university premises, which limited my resources that I could have obtained.

Secondly, I had never used python's framework Django. It meant that I had to research and implement things along with learning. I was using mainly the documentation of the framework as guidance of how certain things should be implemented on the platform. That made the implementation of the platform a slightly more complicated task as I was experimenting, whilst implementing and designing.

Last, but not least, the scale of the project is enormous and with the time given I could develop the platform as a proof-of-concept project, not as a finished product. Personally, I had not taken any

project of this scale and I had not developed a working web portal from scratch. To keep on track with all the mini-goals, I set myself to implement on the platform I had regular meetings with my supervisor where I was explaining my ideas in further details and showed the progress weekly made.

Approach

Agile development methodology

Due to my lack of experience developing web portals/platforms the project is to follow an Agile development methodology. In order to deliver the project in time, I will follow an iterative approach to set milestones. This will allow me to learn, develop and expand my skills in the Django framework's usage. Having said that, it will leave an opportunity for research if needed and issues occur.

The main idea is that the project will be split into specific milestones with an expected finish date. Doing so will make the project more manageable since I do not have current experience in delivering projects of such scale to a finish. The initial idea is that I deliver a minimum viable product, after which I would work mainly on the chatbot or any other additional features as all the main functionalities would be working. Ensuring that would help me deliver the project in time and all the requirements and the acceptance criteria are met.

Regular reviews of the milestones and the progress made will be discussed with the supervisor of mine. That way any issues could be addressed on time.

Personally, I choose agile over other methodologies because it suited the project the most. Agile allows easy and quick adaptivity. If problems do occur during the process of the development phase I would be able to react quite fast and adapt to them. As I want to continue working on the platform even after the project finishes I want as much higher quality product as I can produce. Furthermore, it would help me planning and predict estimated delivery dates.

Learning

Django framework:

Django is a REST framework that is a powerful and flexible toolkit for building web APIs. I decided personally to use Django as I have no previous experience in developing web APIs. It has quite extensive documentation and great community support, which is quite beneficial as I can learn and build a reasonable skillset during the developing part of the project. Furthermore, Django is very customisable as I can use function-based views returning different requests and responses. Also, it has serialization that is supported for both ORM and non-ORM sources. Last, but not least – it is used and trusted by many international companies such as Mozilla, Red Hat, Eventbrite and Heroku. Django is considered a very secure framework, which is very helpful for me as well because security is a serious concern if private data (lectures, student and lecturer data, university data) is stored. Even though, I have taken the project as a proof-of-concept I still consider issues that might occur if security gets exploited.

Researching and getting to know the MVC framework was crucial for the development of the web API. MVC(Model-View-Controller) is an architectural pattern that separates the application into three main logical components – Model, View, Controller. Each of them is built to handle different specific development aspect. The model component stores data and its related logic, the views are concerned with how the data would be represented back to the user and the controller is responsible

for the part of the application that requires user interaction. The controller, for example, will send a command to the model part to update its state, resulting in saving a specific document. Many advantages could be said about the MVC architecture pattern, but I believe the most beneficial for the project was the fact that the code was easy to extend and grow, as well as being easy to maintain.

Django follows the MVC pattern quite closely, but it uses different terminology. It is essentially MTV(Model-Template-View) framework. The framework uses the terms:

- Templates for the Views
- Views for the Controllers.

Having said that, Django is a very appropriate framework for me to use as it covers and solves potential problems I might have with the web API in further phases.

Data storage:

Initially, my idea was to use the university Oracle service, since I did have experience with the DBMS from the modules studied in the second and the third year (CM2102 Database Systems, CM2305 Group Project and CM3104 Large-Scale Databases). However, Django primarily uses SQLite and whilst researching and reviewing different key attributes of different DBMS, I found that SQL lite would be appropriate as the project even though of a large-scale - it is a proof-of-concept. I found SQLite very easy to use as Django allows the migrations to the database to be made dynamically. As the project is a proof-of-concept, problems such as concurrency (no more than one user can be writing to the database at the same time) are not as hard to deal with. Furthermore, it makes the project independent as SQLite is a single-file database and thus serverless. Resulting in making the process of designing the database and its' properties undemanding as I could experiment over the design and change it along the way to fit the requirements.

Chatbot:

In order to find the appropriate library to implement the chatbot into the platform, I had to research existing libraries that could help with the task. Rasa API is an open-source machine learning library for automated text and voice-based conversations. It understands messages, holds conversation and could be connected to external API. Having said that, I decided to use Rasa API for the chatbot service on the web platform. To recognise what the user is saying the Rasa API uses NLU(Natural language understanding) and different intents with different examples are needed to train the bot. Also, the bot has different responses now that it “knows” what the users intent is. Formulating the sequence of responses allows the user to make custom-based actions(methods) that would retrieve data from the given database and return it to the user. I found that feature very useful as I am able to retrieve data such as timetable within the chatbot scope.

Specification

Functional requirements

1. FR1 – The system would provide the user its own secure space, which could only be personally accessed. (Student interface) – Type: Must have

Acceptance Criteria:

- User can log in with a username and a password to the system.
- Method of signing up would be provided to the users.

2. FR2 – The system would allow the user to watch past recordings (Student interface) – Type: Must have

Acceptance Criteria:

- After entering the past-recordings interface the user should be able to see all past recordings from the module
- Video player should be generated for every lecture/past-recording
- Only lectures from the course in the university of the specific user(student) should be allowed to be seen

3. FR3 – The system would allow the user to watch live-streamed lecture (Student interface) – Type: Must have

Acceptance Criteria:

- After entering the live-streaming interface the user should be able to choose in between live-streamed lectures in the current moment
- Video player should allow the user to see the content
- Only Livestream lectures specific to the user's course should be seen

4. FR4 – The system would allow the user to interact with the chatbot (Student interface) – Type: Must have

Acceptance Criteria:

- The chatbot, as well as the platform, is considered as a proof-of-concept and the chatbot would be limited to five use-cases.
- The chatbot should be able to retrieve the timetable that is student-specific
- The chatbot should be able to give more information on different terms that are discussed during the lectures

5. FR5 - The system should allow the user to book a chatroom with the intent of meeting a specific tutor (Student interface) – Type: Must have

Acceptance Criteria:

- Booking a chatroom to meet with a specific lecturer would require a timeframe of initialisation and date on the initialisation of the chatroom. Also, it would require the specific lecturer that the student would like to meet. If the meeting

is approved by the qualified personnel to do so (Lecturers), a chatroom would be created.

6. FR6 – The system should allow the user to join in a chatroom (Student interface) – Type: Must have

Acceptance Criteria:

- The student would be able to join in the already approved chatroom.
- The student would be able to discuss and chat with the lecturer.
- The chatroom itself should be unique and discussion between lecturer and student should remain confidential as that might cause privacy issues if violated.

7. FR7 – The system should allow the user to participate in discussion board discussions (Both Lecturer and Student interface) – Type: Must have

Acceptance Criteria:

- The student would be able to discuss freely on the discussion board that would be specific to the module.

8. FR8 – The system should allow the user to take a self-assessment specific to the module taken (Student interface) – Type: Should have

Acceptance Criteria:

- The student would be able to have a module-specific self-assessment that would generate random questions in the context of the module.

9. FR9 – The system would provide the user its own secure space, which could only be personally accessed. (Lecturer interface) – Type: Must have

Acceptance Criteria:

- User can log in with a username and a password to the system.
- Lecturers can only log in with admin-regulated credentials. They should only be given an account to work with, instead of registering as lecturers as that might cause potential authentication issues.

10. FR10 - The system should allow the user to upload past recordings both automatically and manually (Lecturer interface) – Type: Must have

Acceptance Criteria:

- The user/lecturer would be able to upload past recordings both manually – where the lecturer would attach the lecture and upload it and automatically – where the lecture would be taken straight from the folder and uploaded into the database.

11. FR11 – The system should allow the user to upload a live streamable lecture for specific module (Lecturer interface) – Type: Must have
Acceptance Criteria:
 - The user would be able to upload live streamable lecture with fields being required such as caption of the lecture, URL of the Livestream, starting time and ending time
12. FR12 – The system should allow the user to approve a chatroom meeting with a student (Lecturer interface) – Type: Must have
Acceptance Criteria:
 - The user would be able to approve a chatroom meeting with a student. After approval, the meeting room will be created.
13. FR13 – The system should allow the user to participate in discussion board discussions (Lecturer interface) – Type: Must have
Acceptance Criteria:
 - The user would be able to participate in discussion board discussions freely.
14. FR14 – The system should allow the user to join in a chatroom meeting (Lecturer interface) – Type: Must have
Acceptance Criteria:
 - The user would be able to join in a chatroom meeting with a student and write freely messages that are chatroom specific.

Non-functional requirements

1. NON-FR1 – The system would store entries such as:
 - Data needed to store the live streamable lecture and portray it back to the students.
 - Data needed to store the past recordings
 - Data needed to store the self-assessments specific to the modules
 - Data needed to store the chatroom meetings
 - Data to store the discussion board specific to the module– Type: Must have
Acceptance Criteria:
 - Once a new entry is submitted it should be stored instantaneous and be portrayed back to the user.
2. NON-FR2 – The system will be intuitive and easy to use. (Student and Lecturer interface) – Type: Must have

Acceptance Criteria:

- The User interface should be appropriate and consistent to use.

3. NON-FR3 –The user will have accessibility over the past lectures (Student interface) – Type: Must have

Acceptance Criteria:

- The user will be able to see all the past recordings

4. NON-FR4 – The user will have accessibility over the live-streamed lectures (Student interface) – Type: Must have

Acceptance Criteria:

- The user should be able to see the live-streamed lectures that are part of the modules in his/her course

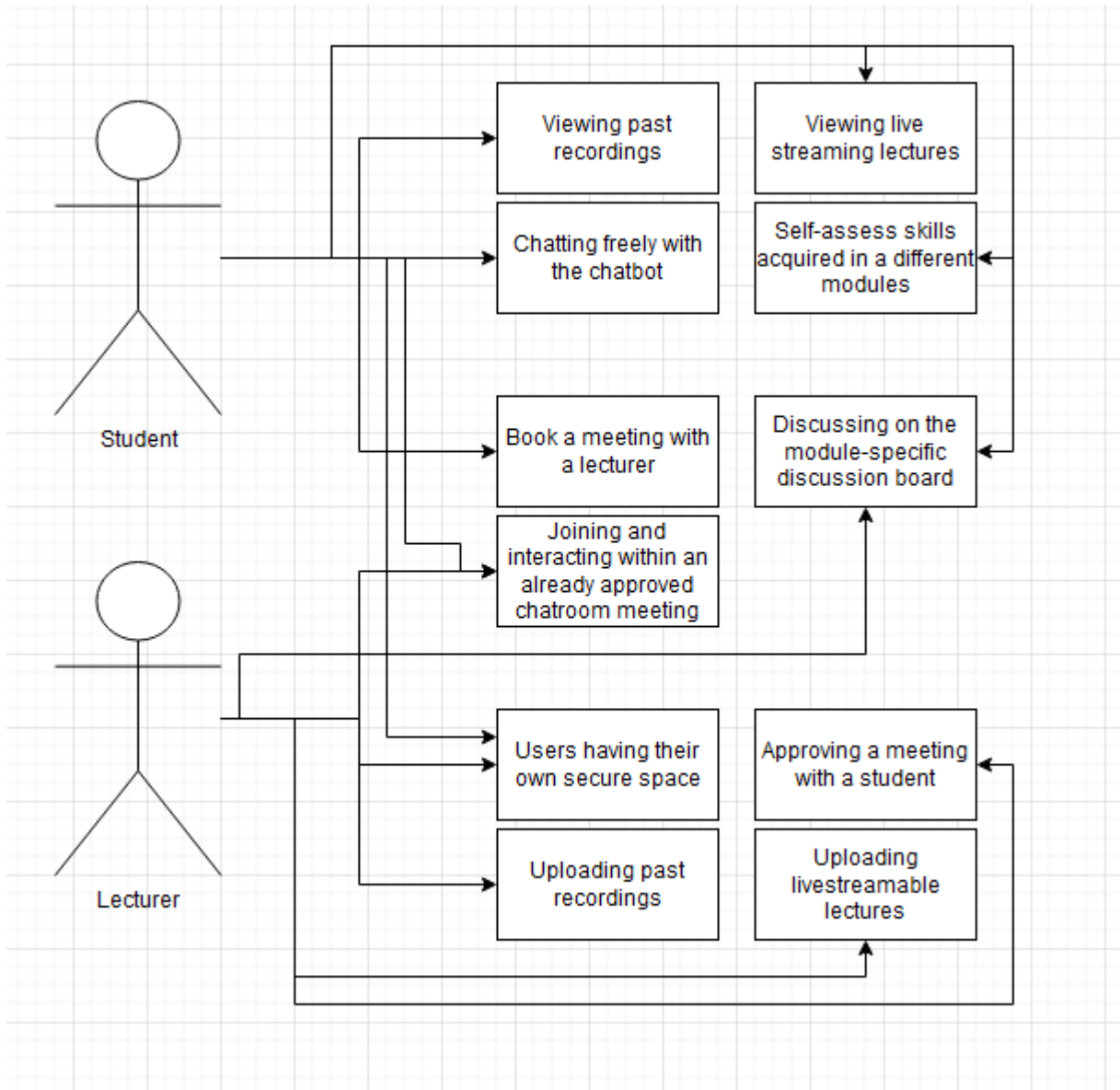
5. NON-FR5 – The user will have the given permissions to upload past recordings and live-streamable lectures. (Lecturer interface) – Type: Must have

Acceptance Criteria:

- The user should be able to upload both past recordings and live-streamable lectures to the platform.

Use cases:

Use case diagram – Django platform

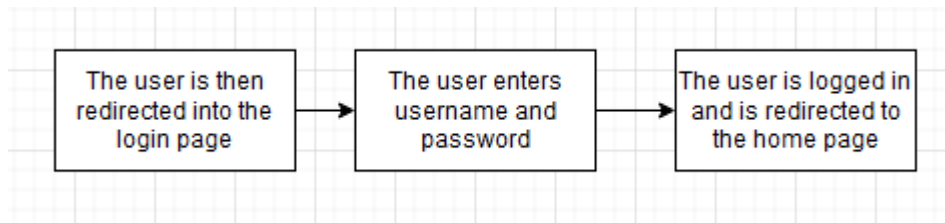


Use cases – Django platform

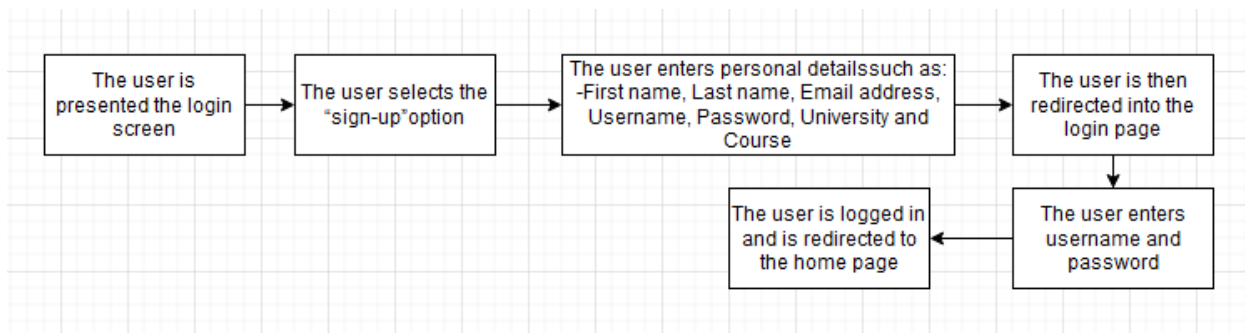
In the use cases section, I will discuss in more detail the use cases including both base and alternative flow with STNs (State Transition Network) visualising how the system would be performing.

Use case No: 1 Users having their own secure space (Both Student and Lecturer interface)		Type: Must have
Goal:	The goal is that a user can log in to an account. If the user is not registered, the system would provide a way for the user to log in on the platform.	
Preconditions:	The user might have no profile on the system. Otherwise, none.	
Basic Flow:	Student: <ol style="list-style-type: none"> 1) The user is then redirected into the login page 2) The user enters username and password 3) The user is logged in and is redirected to the home page 	
Alternative Flow:	Student: <ol style="list-style-type: none"> 1) The user is presented the login screen 2) The user selects the “sign-up” option 3) The user enters personal details such as: - Email address, Username, Password, University and Course 4) The user is then redirected into the login page 5) The user enters username and password 6) The user is logged in and is redirected to the home page Lecturer: <p>The lecturer is provided with an account as otherwise, it might cause potential identity issues.</p>	
Related use cases	All the use-cases require the user to be logged in an account	

Basic flow diagram of Use case No: 1 –

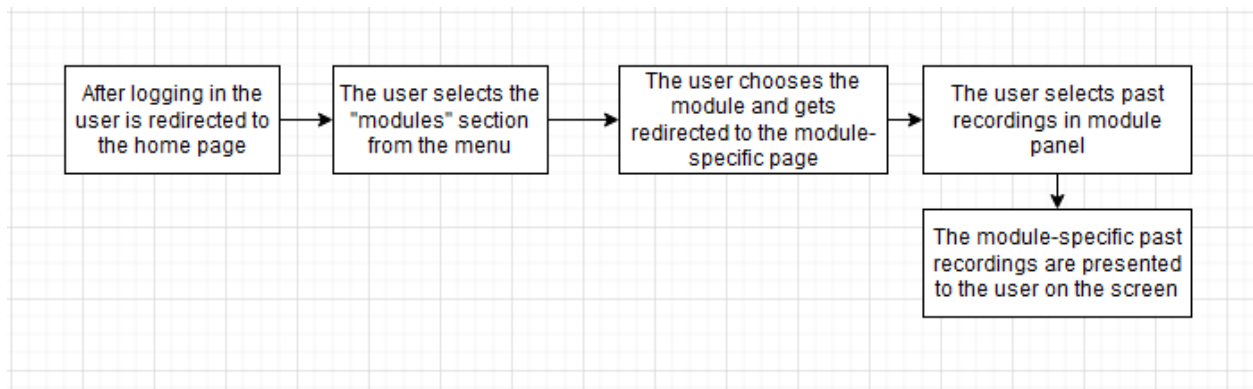


Alternative flow diagram of Use case No: 1 –



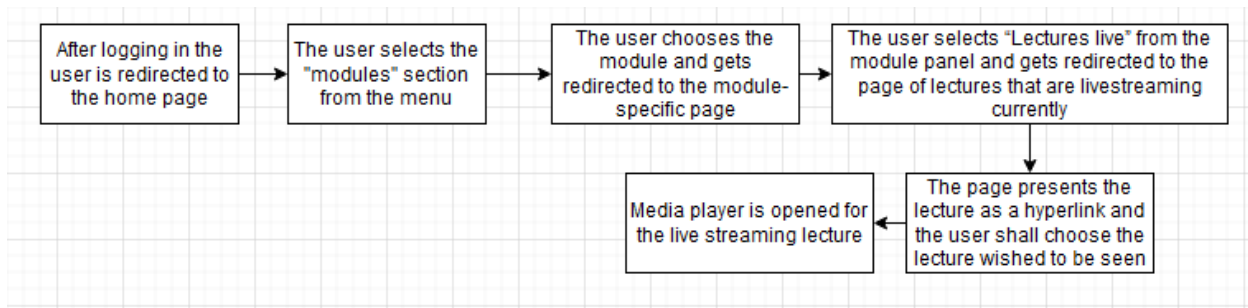
Use case No: 2 Users viewing past recordings (Student interface)		Type: Must have
Goal:	The goal is that a user can view past recordings. All the past recordings would be displayed on a single screen.	
Preconditions:	Firstly, the user would need to be logged as a student to watch the past recordings. Furthermore, the past recordings are module-specific, so the student should be enrolled for the specific module.	
Basic Flow:	1) After logging in the user is redirected to the home page 2) The user selects the "modules" section from the menu 3) The user chooses the module and gets redirected to the module-specific page 4) The user selects past recordings in module panel 5) The module-specific past recordings are presented to the user on the screen	
Alternative Flow:	N/A	
Related use cases	Use case No:10, Use case No: 1	

Basic flow diagram of Use case No: 2



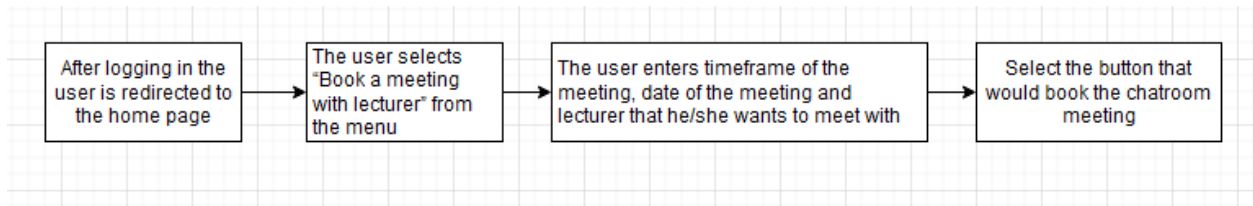
Use case No: 3 Users viewing live streamable lectures (Student interface)		Type: Must have
Goal:	The goal is that a user can view live-stream lectures. The lectures that are currently live streaming would be portrayed as hyperlinks to a page with a media player that would portray the live streaming content	
Preconditions:	The user should be enrolled and enrolled on the module the live streaming lecture is a part of.	
Basic Flow:	<ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects the “modules” section from the menu 3) The user selects the module and gets redirected to the module-specific page 4) The user selects “Lectures live” from the module panel and gets redirected to the page of lectures that are live streaming currently 5) The page presents the lecture as a hyperlink and the user shall choose the lecture wished to be seen 6) Media player is opened for the live streaming lecture 	
Alternative Flow:	N/A	
Related use cases	Use case No: 9, Use case No: 1	

Basic flow diagram of Use case No: 3



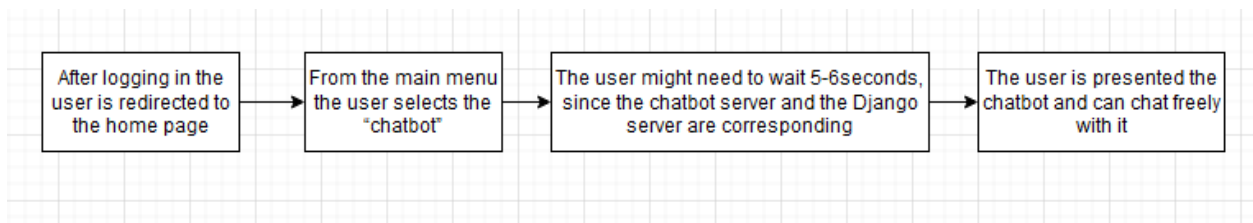
Use case No: 4 Users booking a chatroom for a meeting with a lecturer (Student interface)		Type: Must have
Goal:	The goal is that a user can book a chatroom meeting with a lecturer	
Preconditions:	The user is logged onto the system as a Student.	
Basic Flow:	1) After logging in the user is redirected to the home page 2) The user selects "Book a meeting with lecturer" from the menu 3) The user enters timeframe of the meeting, date of the meeting and lecturer that he/she wants to meet with 4) Select the button that would book the chatroom meeting	
Alternative Flow:	N/A	
Related use cases	Use case No:8, Use case No:6, Use case No: 1	

Basic flow diagram of Use case No: 4



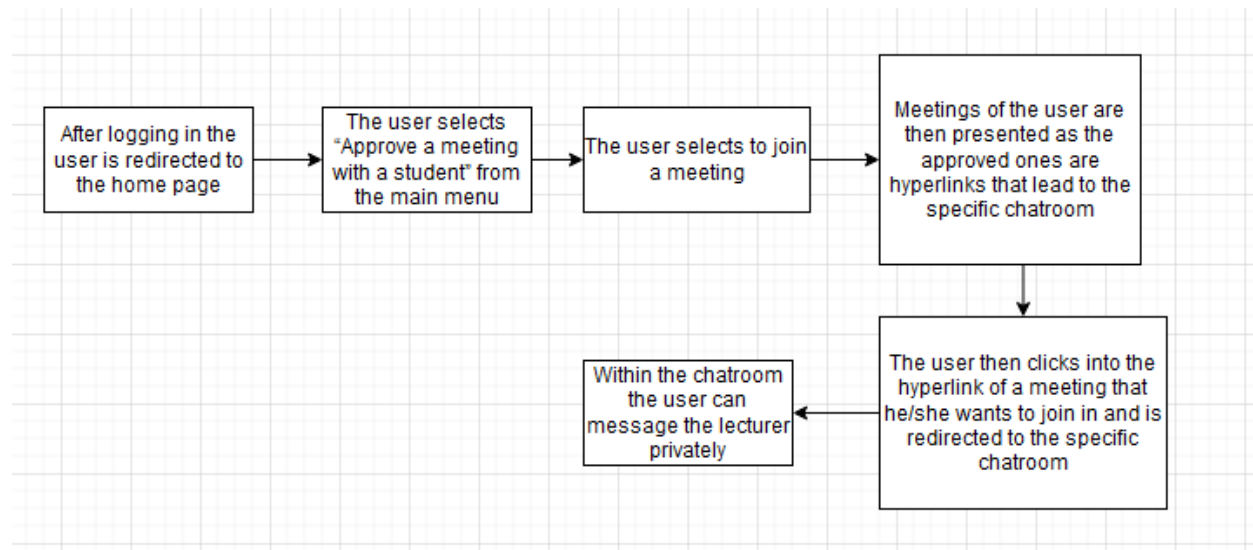
Use case No: 5 Chatting freely with the chatbot (Student interface)		Type: Must have
Goal:	The goal is that user should be able to chat freely with the chatbot	
Preconditions:	The user needs to be logon to the system as a student	
Basic Flow:	<ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) From the main menu the user selects the "chatbot" 3) The user might need to wait 5-6 seconds, since the chatbot server and the Django server are corresponding 4) The user is presented the chatbot and can chat freely with it 	
Alternative Flow:	N/A	
Related use cases	Use case No: 1	

Basic flow diagram of Use case No: 5

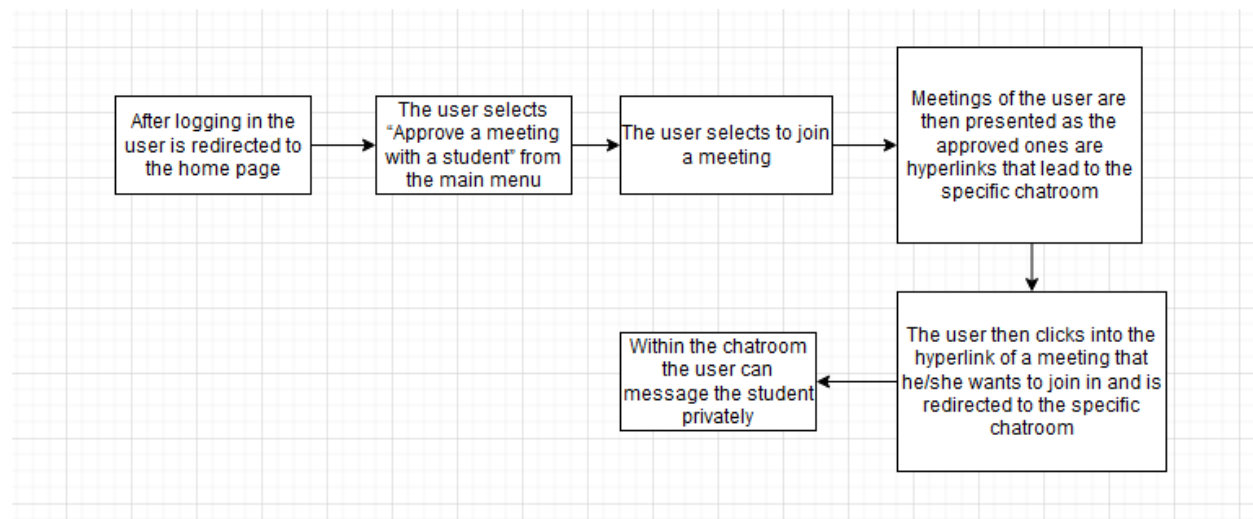


Use case No: 6 Joining and interacting within the chatroom meeting (Both Lecturer and Student interface).		Type: Must have
Goal:	The goal is that a user can join and interact within the chatroom meeting.	
Preconditions:	The user needs to be logged onto the system as a student. Furthermore, the user needs to have the chatroom meeting booked by the student and approved by the lecturer before both users joining.	
Basic Flow:	<p>Student:</p> <ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects “Book a meeting with lecturer” from the main menu 3) The user selects join a meeting 4) Meetings of the user are then presented as the approved ones are hyperlinks that lead to the specific chatroom 5) The user then clicks into the hyperlink of a meeting that he/she wants to join in and is redirected to the specific chatroom 6) Within the chatroom the user can message the lecturer privately <p>Lecturer:</p> <ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects “Approve a meeting with a student” from the main menu 3) The user selects to join a meeting 4) Meetings of the user are then presented as the approved ones are hyperlinks that lead to the specific chatroom 5) The user then clicks into the hyperlink of a meeting that he/she wants to join in and is redirected to the specific chatroom 6) Within the chatroom the user can message the student privately 	
Alternative Flow:	N/A	
Related use cases	Use case No:8, Use case No:4, Use case No: 1	

Basic flow diagram of Use case No: 6 for the student

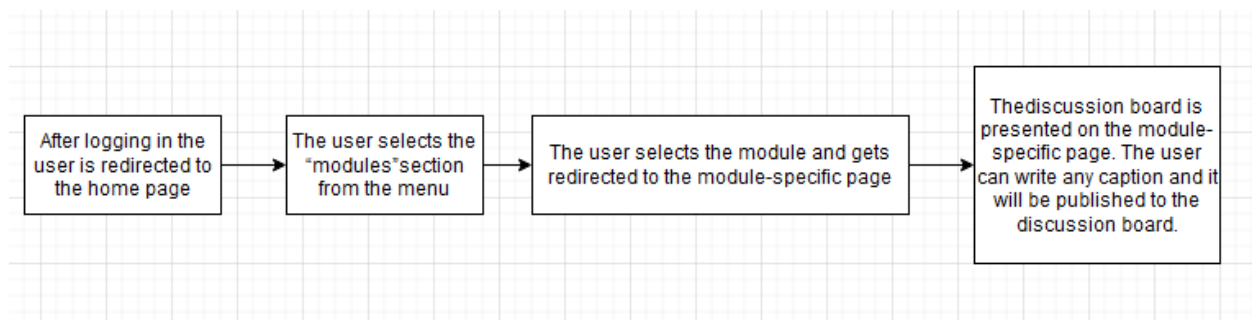


Basic flow diagram of Use case No: 6 for the lecturer



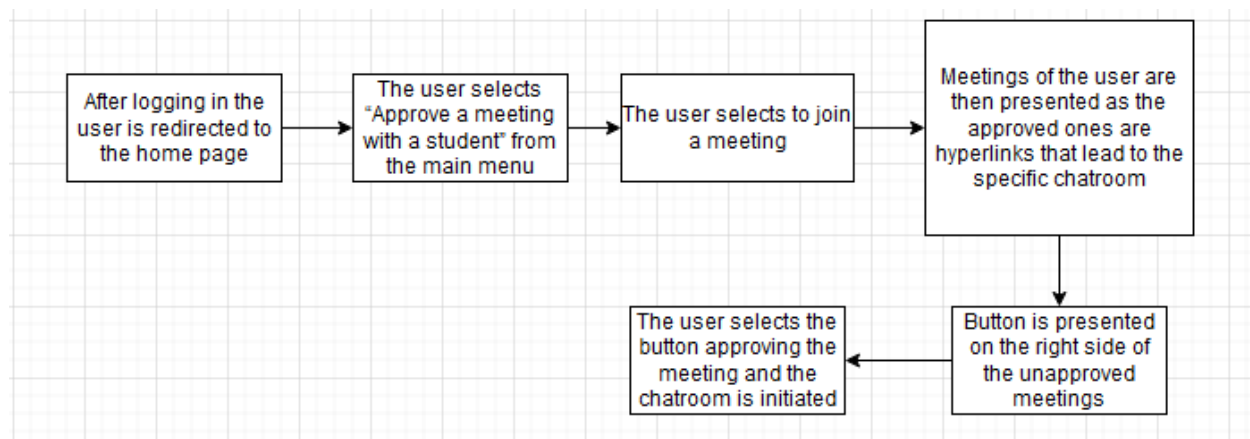
Use case No: 7 Users discussing on the discussion board (Both Student and Lecturer interface)		Type: Must have
Goal:	The goal is that both types of users can discuss in a module they are part of.	
Preconditions:	The users should have logged onto the system.	
Basic Flow:	Both the lecturer and the student follow the same flow: <ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects the “modules” section from the menu 3) The user selects the module and gets redirected to the module-specific page 4) The discussion board is presented on the module-specific page. The user can write any caption and it will be published to the discussion board. 	
Alternative Flow:	N/A	
Related use cases	Use case No: 1	

Basic flow diagram of Use case No: 7



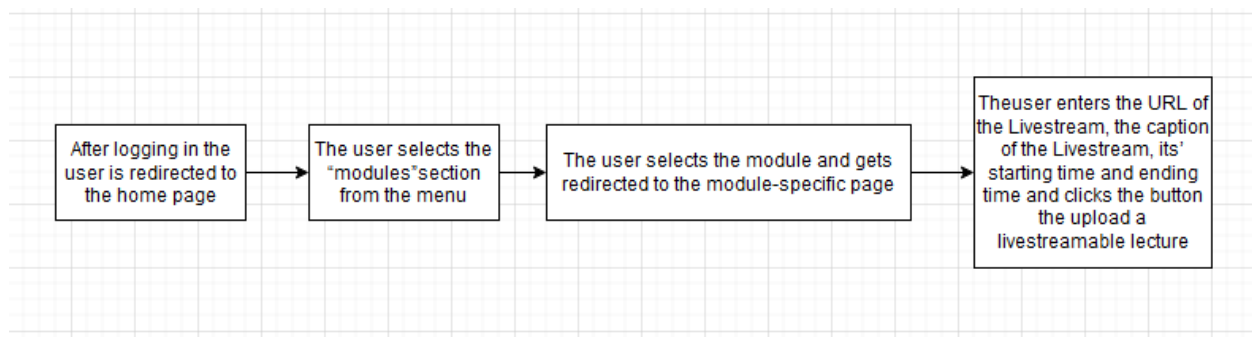
Use case No: 8 Approving the chatroom meeting (Lecturer interface)		Type: Must have
Goal:	The goal is that a user (lecturer) can approve a chatroom meeting with a student	
Preconditions:	The user should have logged onto the system as a lecturer.	
Basic Flow:	<ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects “Approve a meeting with a student” from the main menu 3) The user selects to join a meeting 4) Meetings of the user are then presented as the approved ones are hyperlinks that lead to the specific chatroom 5) Button is presented on the right side of the unapproved meetings 6) The user selects the button approving the meeting and the chatroom is initiated 	
Alternative Flow:	N/A	
Related use cases	Use case No:4, Use case No:6, Use case No: 1	

Basic flow diagram of Use case No: 8



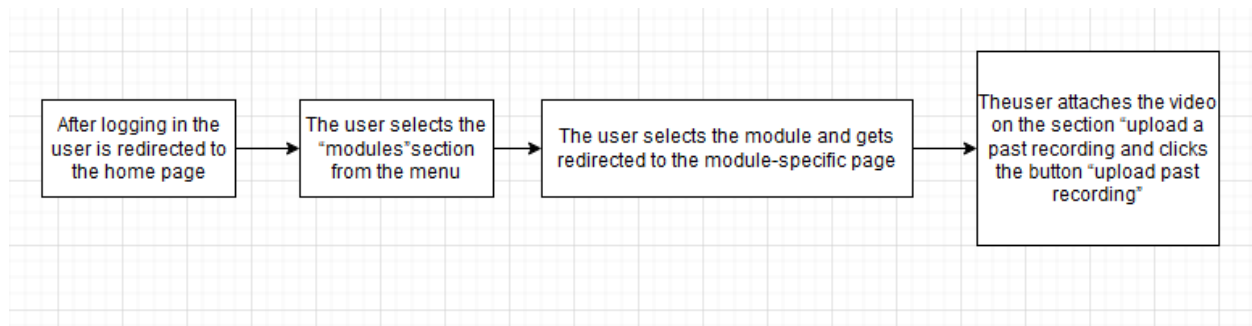
Use case No: 9 Uploading live streamable lectures (Lecturer interface)		Type: Must have
Goal:	The goal is that a user can upload live-streaming lectures.	
Preconditions:	The user should be logged onto the system as a lecturer.	
Basic Flow:	<ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects the “modules” section from the menu 3) The user selects the module and gets redirected to the module-specific page 4) The user enters the URL of the Livestream, the caption of the Livestream, its’ starting time and ending time and clicks the button the upload a live streamable lecture 	
Alternative Flow:	N/A	
Related use cases	Use case No: 3, Use case No: 1	

Basic flow diagram of Use case No: 9



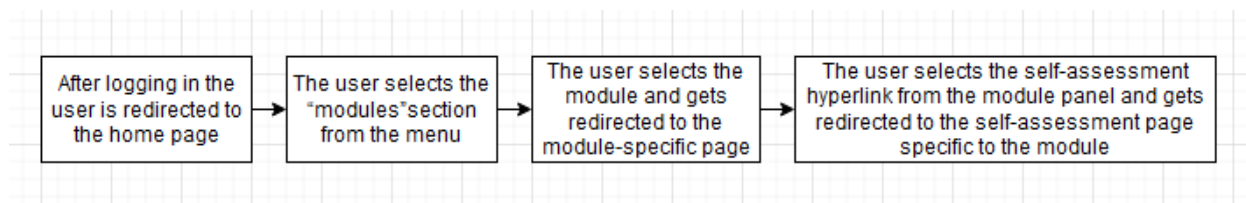
Use case No: 10 Uploading past recordings (Lecturer interface)		Type: Must have
Goal:	The goal is that a user can upload past recordings.	
Preconditions:	The user should be logged onto the system as a lecturer.	
Basic Flow:	<ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects the “modules” section from the menu 3) The user selects the module and gets redirected to the module-specific page 4) The user attaches the video on the section “upload a past recording and clicks the button “upload past recording” 	
Alternative Flow:	N/A	
Related use cases	Use case No: 2, Use case No: 1	

Basic flow diagram of Use case No: 10

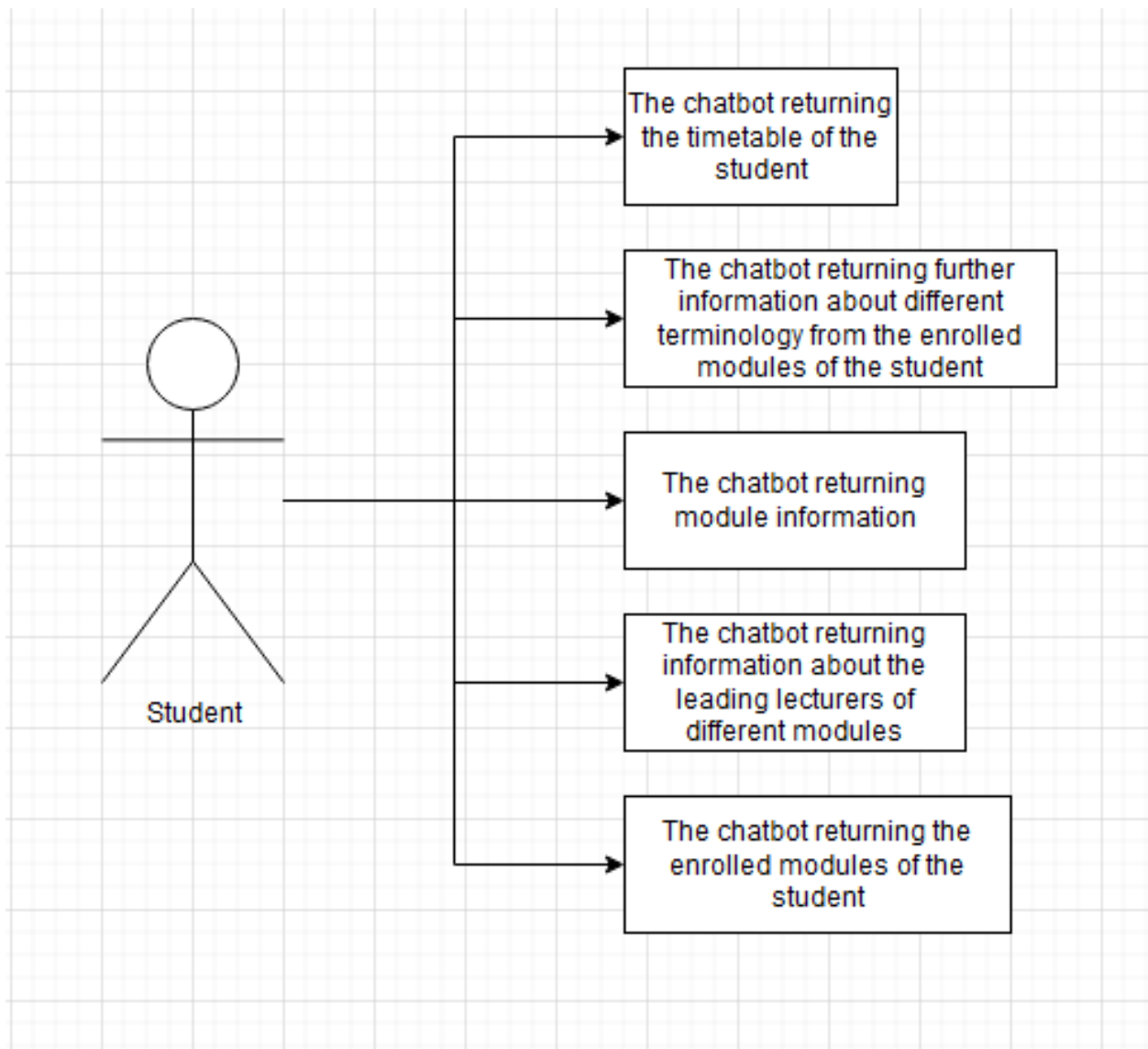


Use case No: 11 The user could self-assess the skills acquired from a specific module (Student interface)		Type: Must have
Goal:	The goal is that a user could self-assess the skills acquired from a specific module	
Preconditions:	The user should be logged onto the system as a student.	
Basic Flow:	<ol style="list-style-type: none"> 1) After logging in the user is redirected to the home page 2) The user selects the “modules” section from the menu 3) The user selects the module and gets redirected to the module-specific page 4) The user selects the self-assessment hyperlink from the module panel and gets redirected to the self-assessment page specific to the module 	
Alternative Flow:	N/A	
Related use cases	N/A	

Basic flow diagram of Use case No: 11



Use case diagram – Chatbot



Use cases – Chatbot

Use case No: 12 The chatbot returning the timetable of the student		Type: Must have
Goal:	The chatbot returns information about the timetable of the student	
Preconditions:	The user needs to be logged onto the system as a student and be on the chatbot page.	
Basic Flow:	1) The machine learning algorithm behind the RASA API will detect if such action is to be executed from the users' input 2) Timetable being returned	
Alternative Flow:	N/A	
Related use cases	Use case No: 1, Use case No: 5	

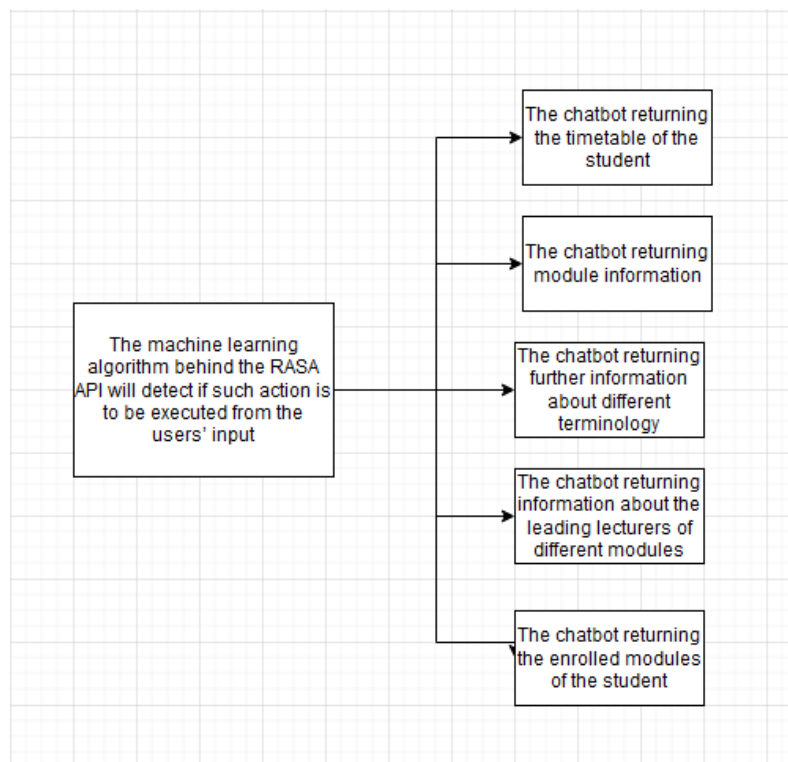
Use case No: 13 The chatbot returning further information about different terminology		Type: Must have
Goal:		
Preconditions:	The user needs to be logged onto the system as a student and be on the chatbot page.	
Basic Flow:	1) The machine learning algorithm behind the RASA API will detect if such action is to be executed from the users' input 2) Information about different terminology being returned	
Alternative Flow:	N/A	
Related use cases	Use case No: 1, Use case No: 5	

Use case No:14 The chatbot returning module information		Type: Must have
Goal:		
Preconditions:	The user needs to be logged onto the system as a student and be on the chatbot page.	
Basic Flow:	1) The machine learning algorithm behind the RASA API will detect if such action is to be executed from the users' input 2) Module information being returned	
Alternative Flow:	N/A	
Related use cases	Use case No: 1, Use case No: 5	

Use case No: 15 The chatbot returning information about the leading lecturers of different modules		Type: Must have
Goal:		
Preconditions:	The user needs to be logged onto the system as a student and be on the chatbot page.	
Basic Flow:	1) The machine learning algorithm behind the RASA API will detect if such action is to be executed from the users' input 2) Information regarding leading lecturers of different modules is being returned	
Alternative Flow:	N/A	
Related use cases	Use case No: 1, Use case No: 5	

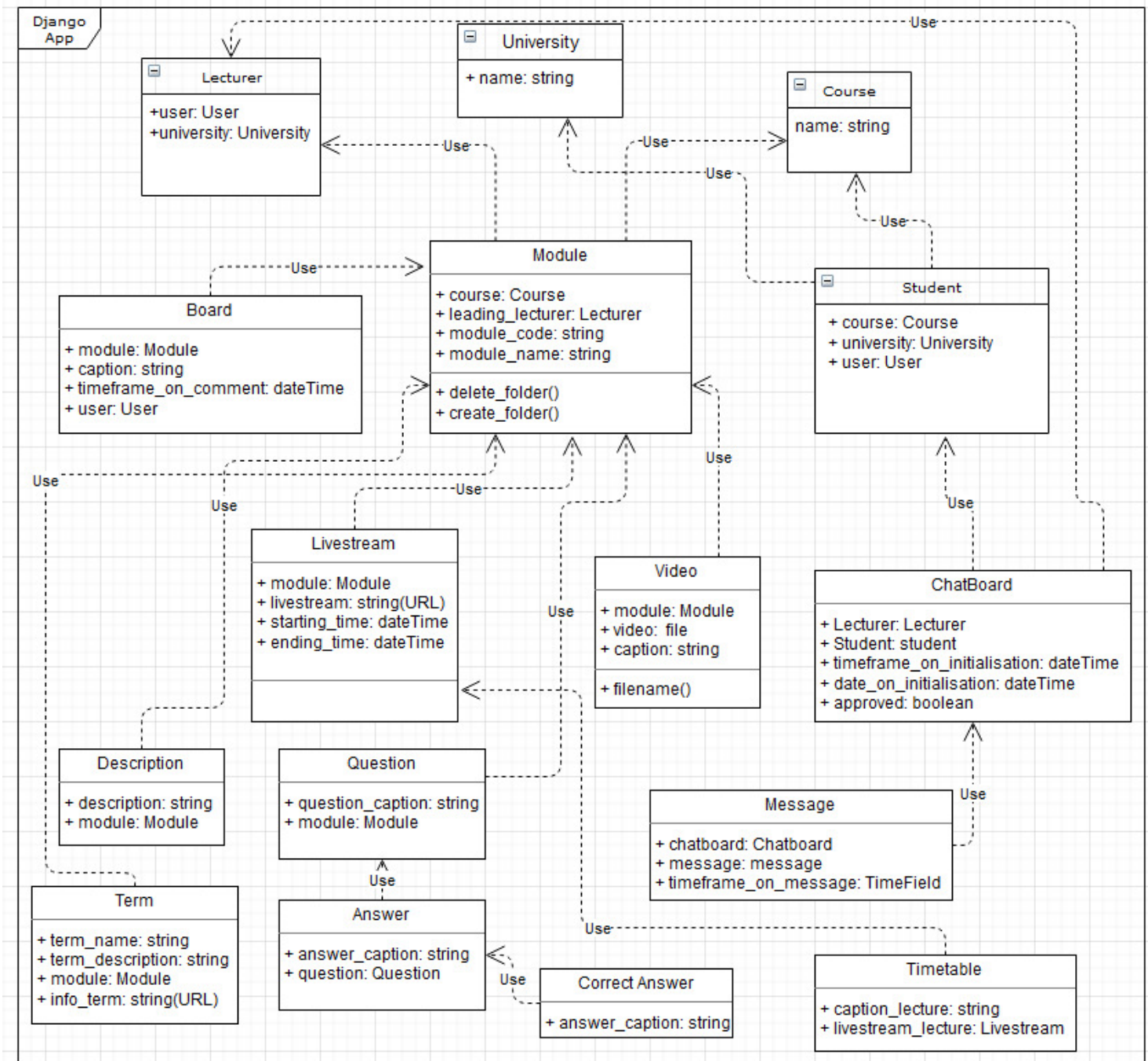
Use case No: 16 The chatbot returning the enrolled modules of the student		Type: Must have
Goal:		
Preconditions:	The user needs to be logged onto the system as a student and be on the chatbot page.	
Basic Flow:	1) The machine learning algorithm behind the RASA API will detect if such action is to be executed from the users' input 2) The enrolled modules of the student being returned	
Alternative Flow:	N/A	
Related use cases	Use case No: 1, Use case No: 5	

General basic flow diagram of Use Case No: 12-16



Design

UML Class Diagram



Overview of the UML Class Diagram:

The UML Class Diagram of a Django application is different in a way from most of the UML class diagrams of other applications. The reason for this to happen is that one of the most powerful Django's features is the Django ORM(Object-relational mapper)^[7]. There are no ordinary classes and interfaces as Django's MVT divides the main sections into the discussed earlier models, views and templates. Having said that, the Class Diagram of a Django application is very close to what the entity-relationship diagram might look like, where the ER diagram describes the way Django's models work (database). In the following example, I have shown how a normal class is defined within the ORM of Django.

```
class Module(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    leading_lecturer = models.ForeignKey(Lecturer, on_delete=models.CASCADE)
    module_code = models.CharField(max_length=7)
    module_name = models.CharField(max_length=30)
    def __str__(self):
        return self.module_code + " " + self.module_name

@receiver(signals.post_save, sender=Module)
def create_folder(sender, instance, **kwargs):
    x = Path(os.getcwd())
    pathing = join(x, "media\\pastrecordings\\" + instance.module_name)
    os.makedirs(pathing)

@receiver(signals.post_delete, sender=Module)
def delete_folder(sender, instance, **kwargs):
    x = Path(os.getcwd())
    pathing = join(x, "media\\pastrecordings\\" + instance.module_name)
    os.rmdir(pathing)
```

User interface prototypes

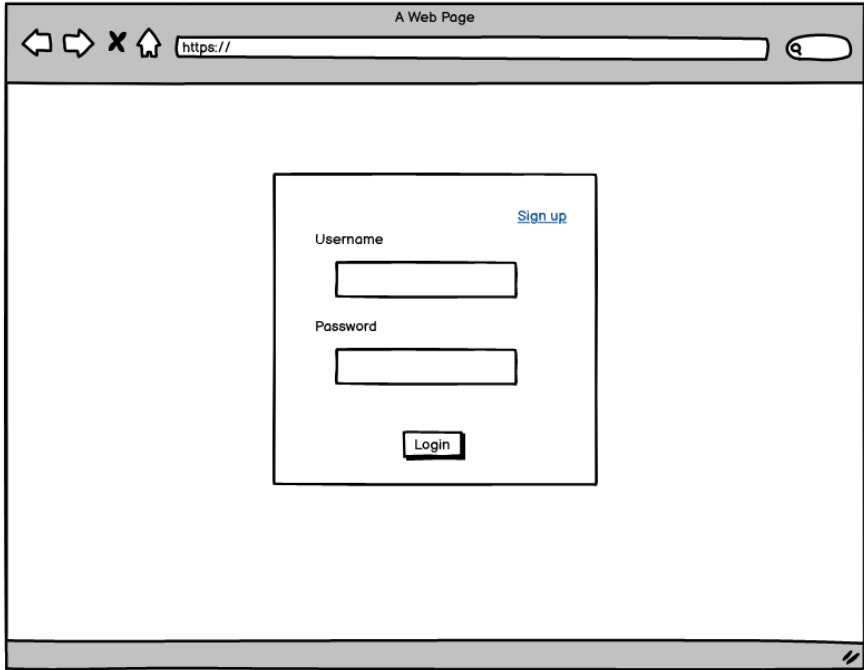
One of the main priorities of the project was the usability of the web platform. It was essential to create mock-ups and prototypes of the system before I started developing the whole web platform. For the mock-ups to be precise, I kept in mind the personas, the functional requirements of the system and the use cases.

The main principles I looked upon applying were the CARP principles – contrast, alignment, proximity, and repetition. The heuristics of Nielsen were also applied when it came down to evaluate the prototypes. ^[8]

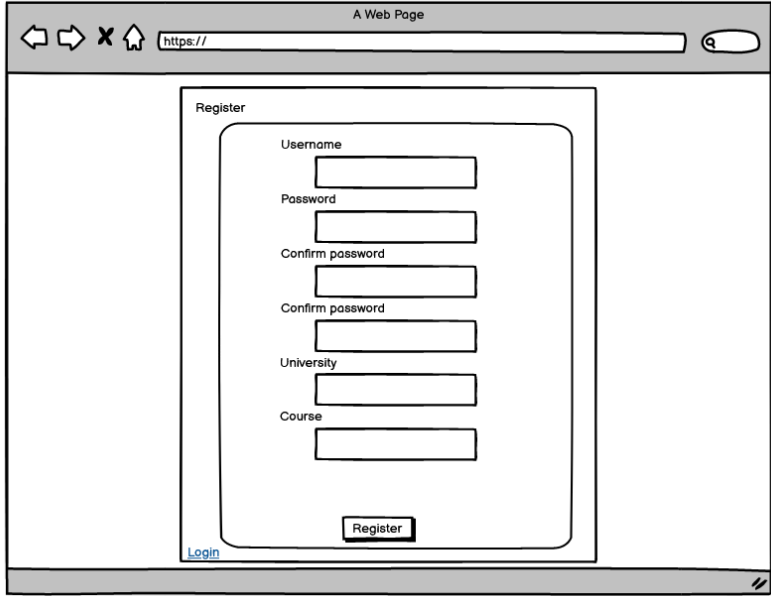
Since in the HCI coursework (Human-Computer interaction) we used the Balsamiq cloud, I decided to use it again, because I was familiar with how it works, for my prototypes and mock-ups as it is an efficient tool for its' purpose.

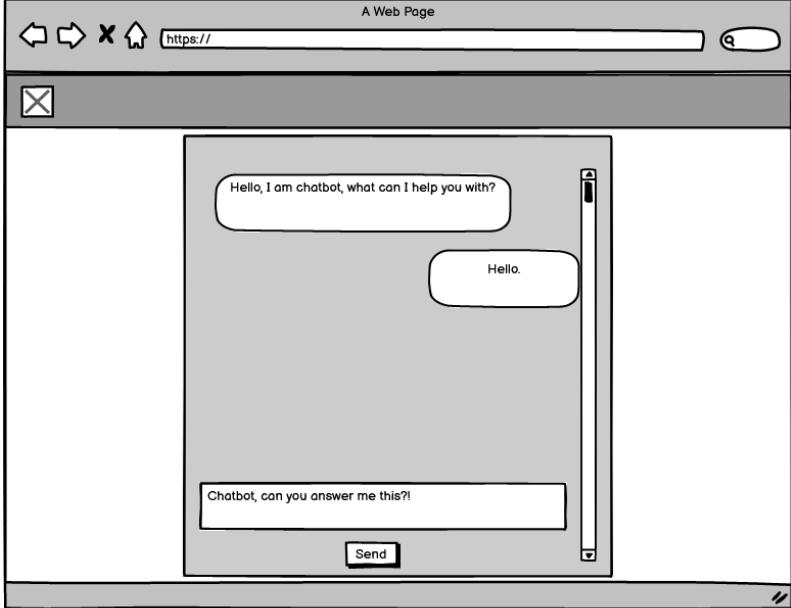
Initial prototypes:

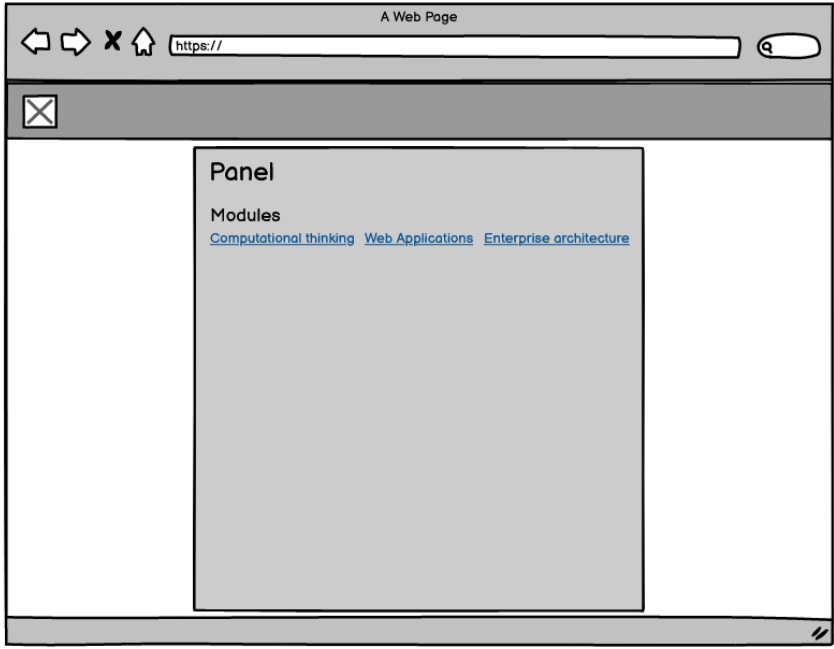
With all the requirements in mind, the initial prototype was a MVP design of the platform. The following prototypes do not follow the STNs precisely, but the final prototypes do. After evaluating the initial prototype I derived a final prototype, which would be precisely based on the STNs and all the requirements.

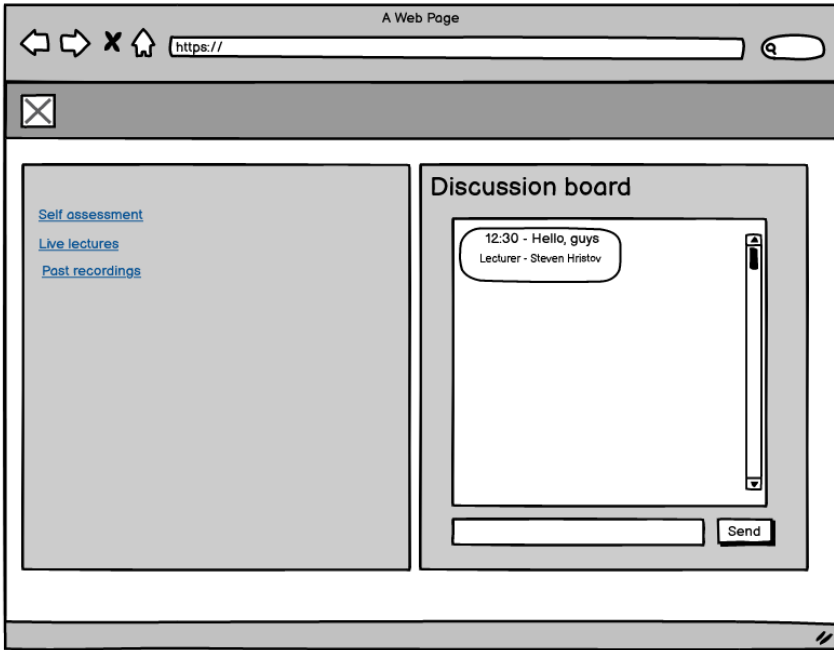
Prototype No: 1	
Description	The user logging onto the system(Both Student and Lecturer interface) This screen is where the user would log onto the system. To do so, the user would have to fill in his/her username and password and press the login button.

	<p>The option of signing up would redirect the user to the registration page/panel. Colour differentiation is one of the concepts of the CARP principles. The blue, white and grey gamma would be used all around the web platform. Having said that, all the elements are also aligned, which is the CARP principle as well.</p> <p>Related use cases: Use case No. 1</p>
--	---

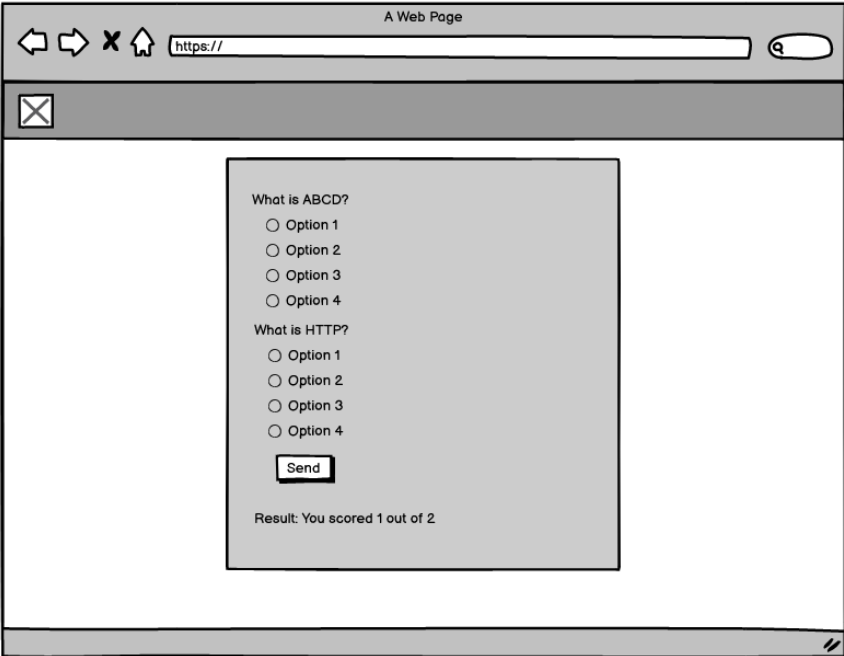
<p>Prototype No: 2</p>	
<p>Description</p>	<p>The user registering on the platform</p> <p>On the following screen is presented the registration page, where the user can register by filling the fields of username, password, university and course. The colour scheme, as well as the alignment, are present here also.</p> <p>Related use cases: Use case No. 1</p>

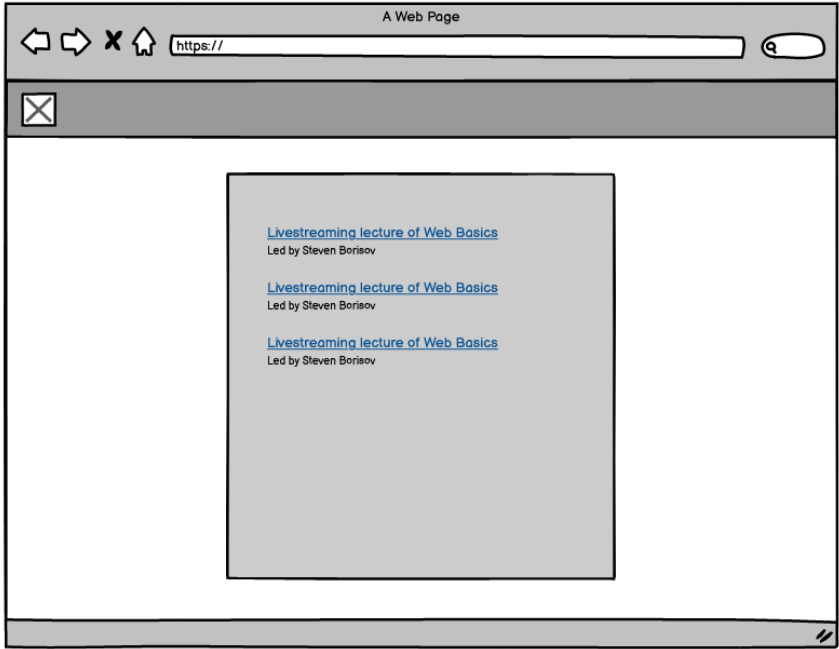
Prototype No: 3	
Description	<p>The user interacting with the chatbot(Student interface)</p> <p>The following screen presents the chatbot page. The user will be able to correspond with the chatbot by typing a message in the text field and pressing the button send.</p> <p>Related use cases: Use case No.5 and Use cases No. 12-16</p>

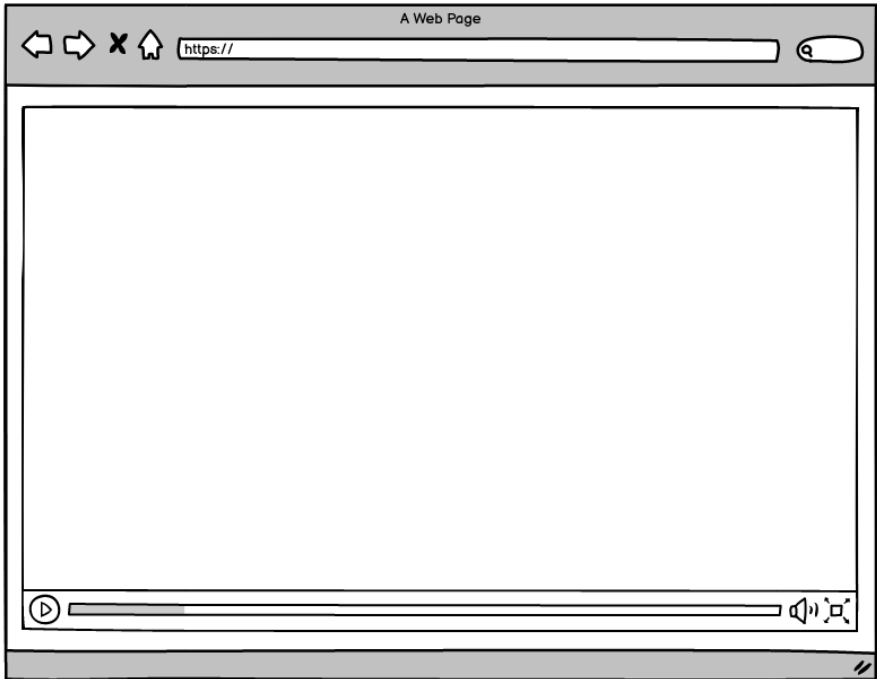
Prototype No: 4	
Description	<p>The main panel including the modules of the user(Both Student and Lecturer interface)</p> <p>The following screen portrays the modules panel. All the enrolled modules of the student are portrayed as hyperlinks that would redirect to the module-specific page. The colour gamma and the aligning factor are present.</p> <p>Related use cases: N/A</p>

Prototype No: 5	
Description	<p>The module-specific page(Student interface)</p>

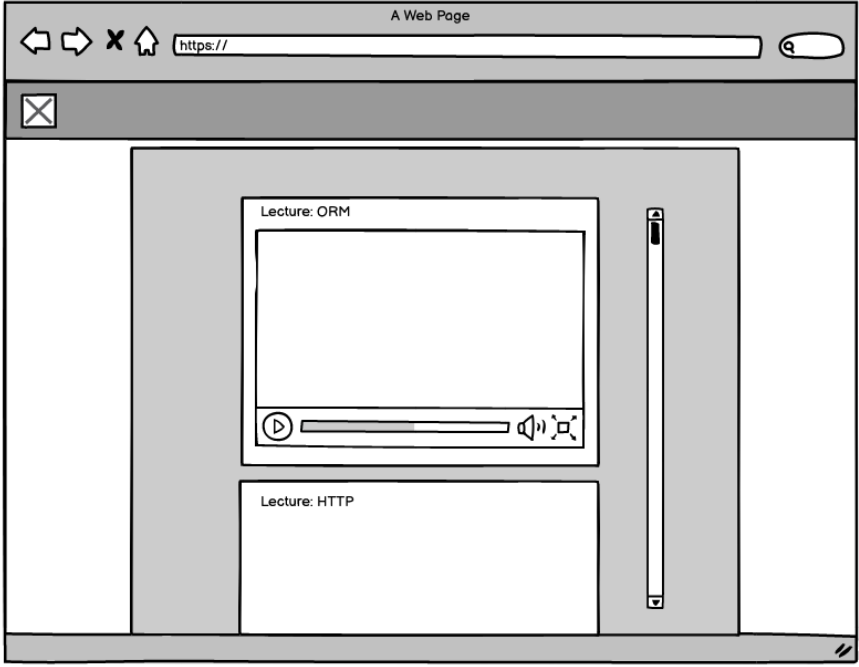
	<p>The following screen portrays the modules-specific page. The user can access the self-assessment, the live lectures and the past recordings by clicking on the hyperlinks. A discussion board is present as well, where the user can interact with other users that are part of the module. For the user to write anything on the discussion board, the caption should be written in the text field and the button send should be clicked. The message will then be portrayed on the discussion board.</p> <p>Related use cases: Use case No. 7</p>
--	---

Prototype No: 6	
Description	<p>The user having access over the module-specific self-assessment(Student interface)</p> <p>The following screen portrays the self-assessment page, where the user can self-assess the skills acquired from a specific module. The questions are shown above the answers. Different answers could be selected as they are presented as radio buttons. When the user decides on submitting by pressing the button send, the result would pop under the button indicating the result he/she got.</p> <p>Related use cases: Use case No. 11</p>

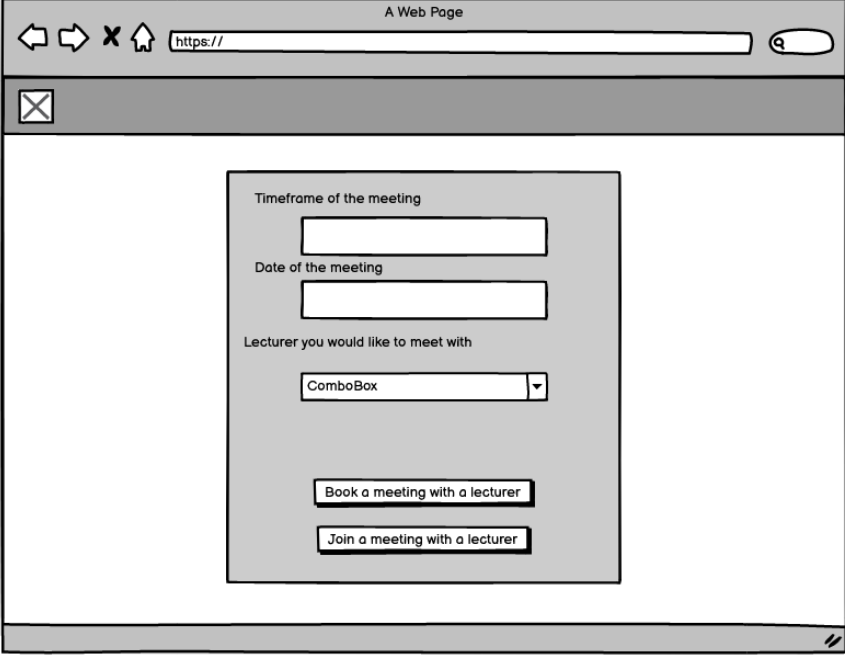
Prototype No: 7	 <p>A web browser window titled "A Web Page" with a search bar and navigation icons. The main content area displays a list of three identical hyperlinks: "Livestreaming lecture of Web Basics" followed by "Led by Steven Borisov".</p>
Description	<p>The user having access over the streaming lectures(Student interface)</p> <p>The following screen portrays the page that represents to the student the live lectures. The live lectures are presented as hyperlinks that redirect to the specific live streaming lecture. Below the hyperlink, a text is written of who is the lecturer that leads the lecture.</p> <p>Related use cases: Use case No.3</p>

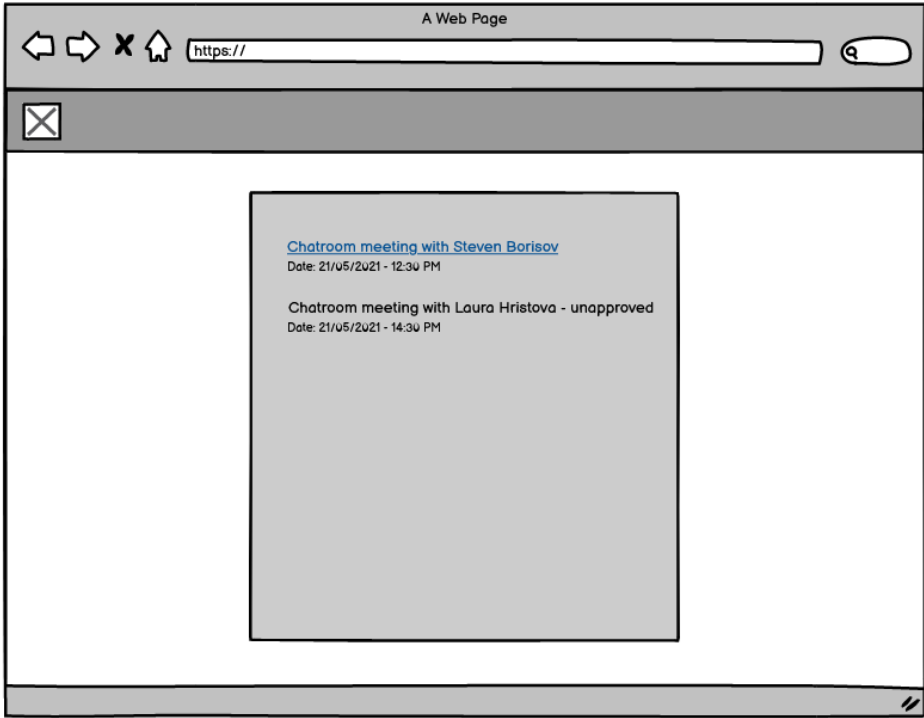
Prototype No: 8	 <p>A web browser window titled "A Web Page" with a search bar and navigation icons. The main content area is a large empty rectangular box, likely a video player. At the bottom of the box, there is a progress bar and control icons (play, pause, volume, etc.).</p>
-----------------	--

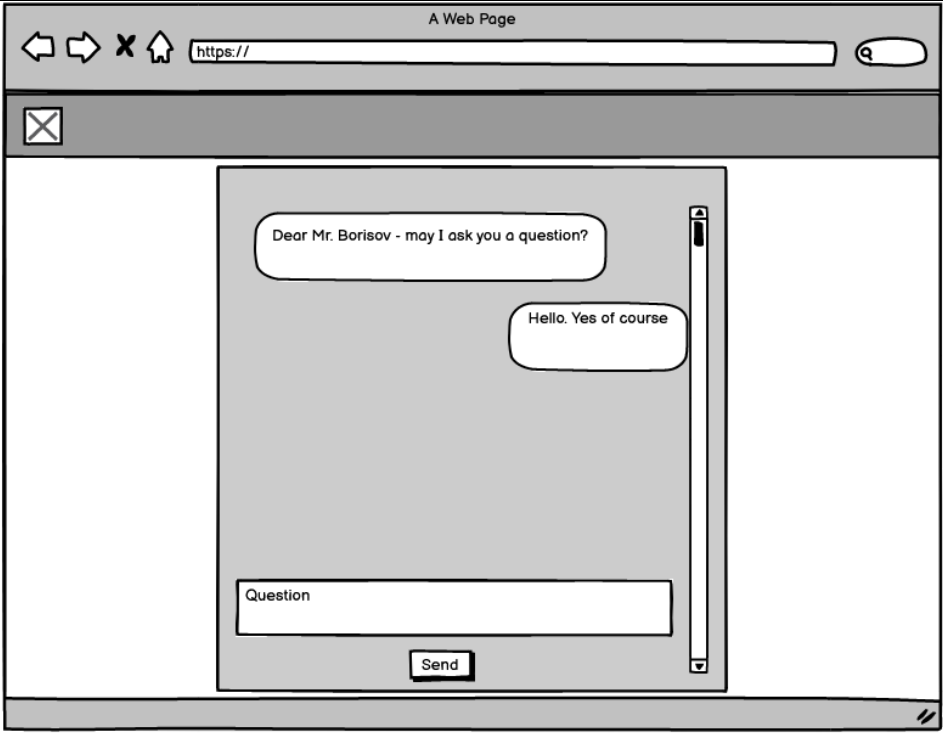
Description	<p>The user having access over the streaming lecture (Student interface)</p> <p>On the following screen, the media player of the live streaming lecture is being portrayed.</p> <p>Related use cases: Use case No.3</p>
-------------	---

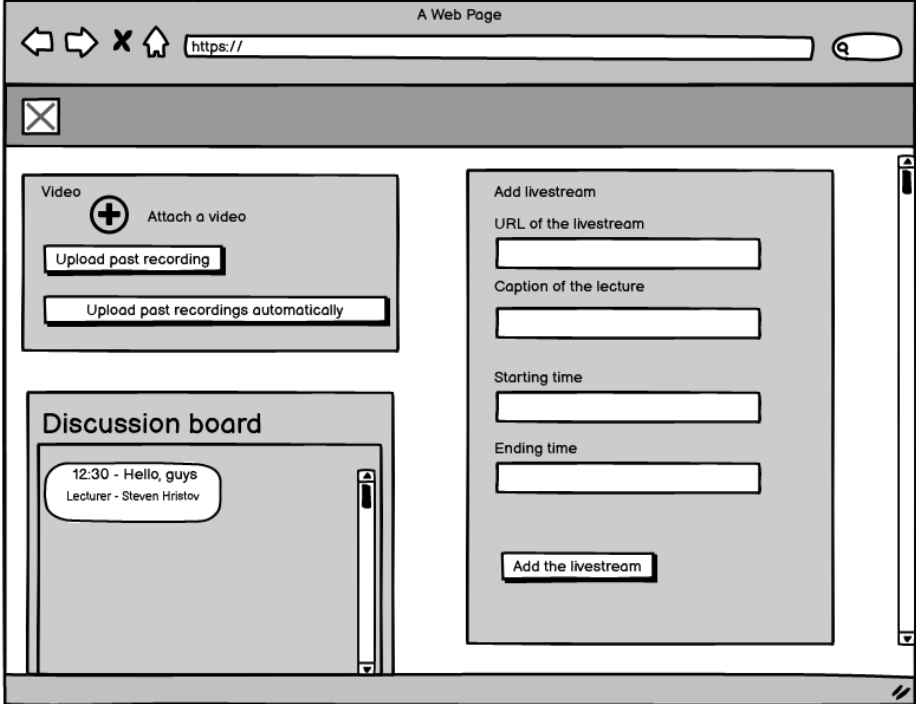
Prototype No: 9	
Description	<p>The user having access over the past recordings of a module(Student interface)</p> <p>The past recordings screen is portraying media players inside different “cards” with the title of the lecture being presented on the top of the “card”.</p> <p>Related use cases: Use case No.2</p>

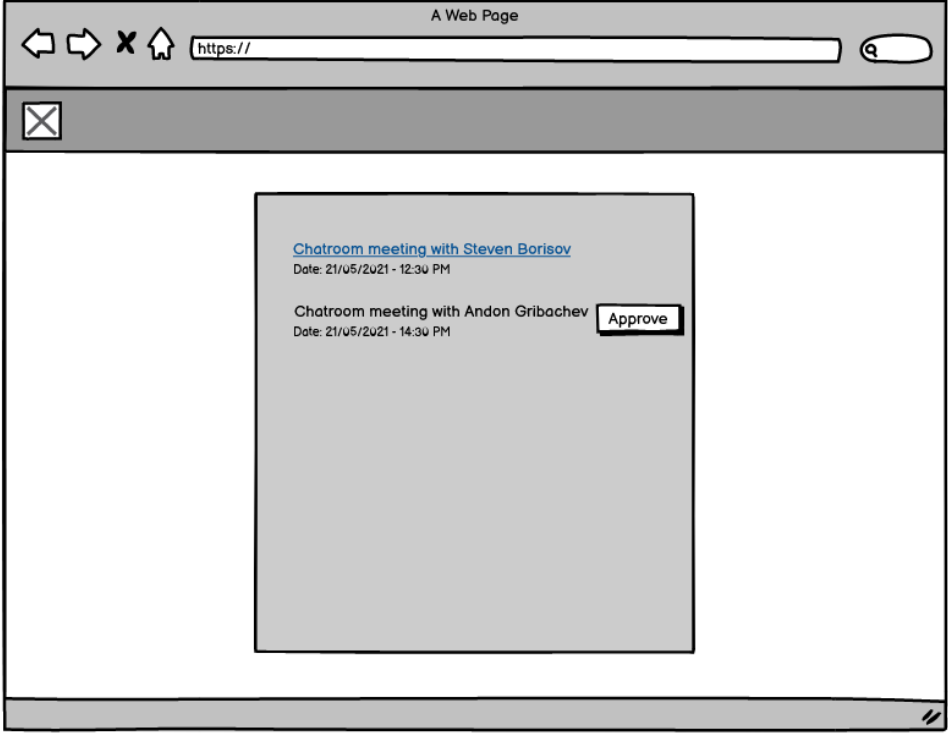
S

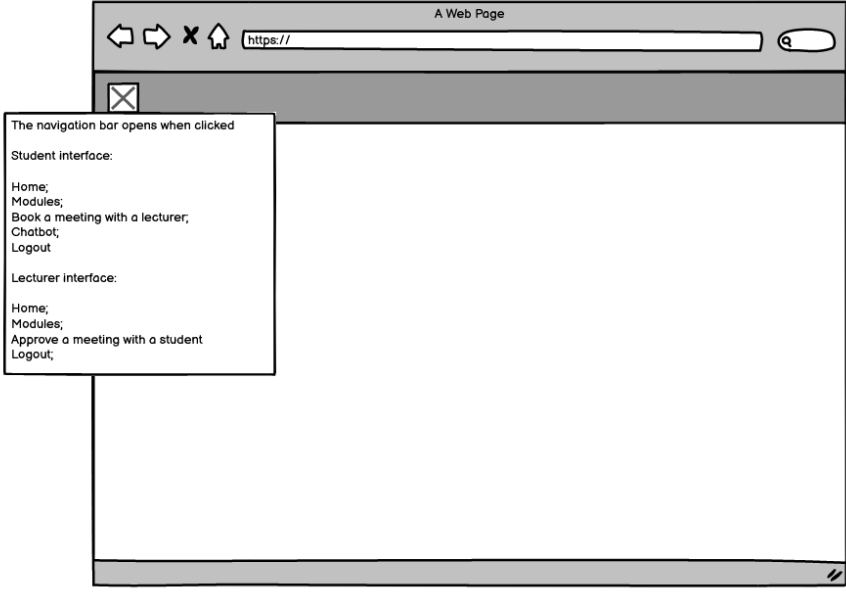
Prototype No: 10	
Description	<p>The user being able to book a meeting with a lecturer(Student interface)</p> <p>The following screen represents the book a meeting section in the student interface. By entering the timeframe of the meeting, date of the meeting and choosing the lecturer the student wants to meet with, a chatroom meeting is initiated by pressing the button “book a meeting with a lecturer”. On the other hand, if the user would like to join a specific meeting, the button that would join a meeting with a lecturer would redirect the student into the screen of the meeting.</p> <p>Related use cases: Use case No. 4; Use case No. 6; Use case No. 8;</p>

Prototype No: 11	
Description	<p>The user having access over the chatroom meetings (Student interface)</p> <p>The meetings screen of the student interface portrays the chatroom meetings that the user can join. The joinable ones are presented as hyperlinks and the unapproved ones are presented as the text below.</p> <p>Related use cases: Use case No. 4; Use case No. 6; Use case No. 8;</p>

Prototype No: 12	
Description	<p>The user having access over a chatroom meeting (Both Lecturer and Student interface)</p> <p>The chatroom meeting screen is portraying the chatroom itself. Messages are presented in a typical manner. By entering the text field and pressing the button send, the user (both lecturer and student) can send a message privately.</p> <p>Related use cases: Use case No. 4; Use case No. 6; Use case No. 8;</p>

Prototype No: 13	
Description	<p>The user having access over the module-specific page (Lecturer interface)</p> <p>The module-specific page screen portrays the interface of the lecturer. The lecturer can add a live-streaming lecture by writing the URL of the live stream, the caption of the lecture, its starting time and ending time. With filled fields, by pressing the button add a live stream the live-streaming lecture would be added. Furthermore, the lecturer can attach a video and upload it as a past recording by pressing the button upload past recording. On the other hand, the lecturer can also automatically add past recordings to the database. Just like the student, the lecturer can also type in the discussion board and see its content. Even though, portrayed below, the discussion board has the same functionality and interface as in the students' module-specific page.</p> <p>Related use cases: Use case No.7; Use case No.9, Use case No.10</p>

Prototype No: 14	
Description	<p>The user having access over the chatroom meetings (Lecturer interface)</p> <p>Just as in the student interface, in the lecturer interface the accessible chatroom meetings are portrayed as hyperlinks. However, the lecturer has the permission of approving a specific meeting by pressing the button on the right-side of a booked by the student meeting. That way the chatroom goes from unapproved to approved and from text to hyperlink.</p> <p>Related use cases: Use case No. 4; Use case No. 6; Use case No. 8;</p>

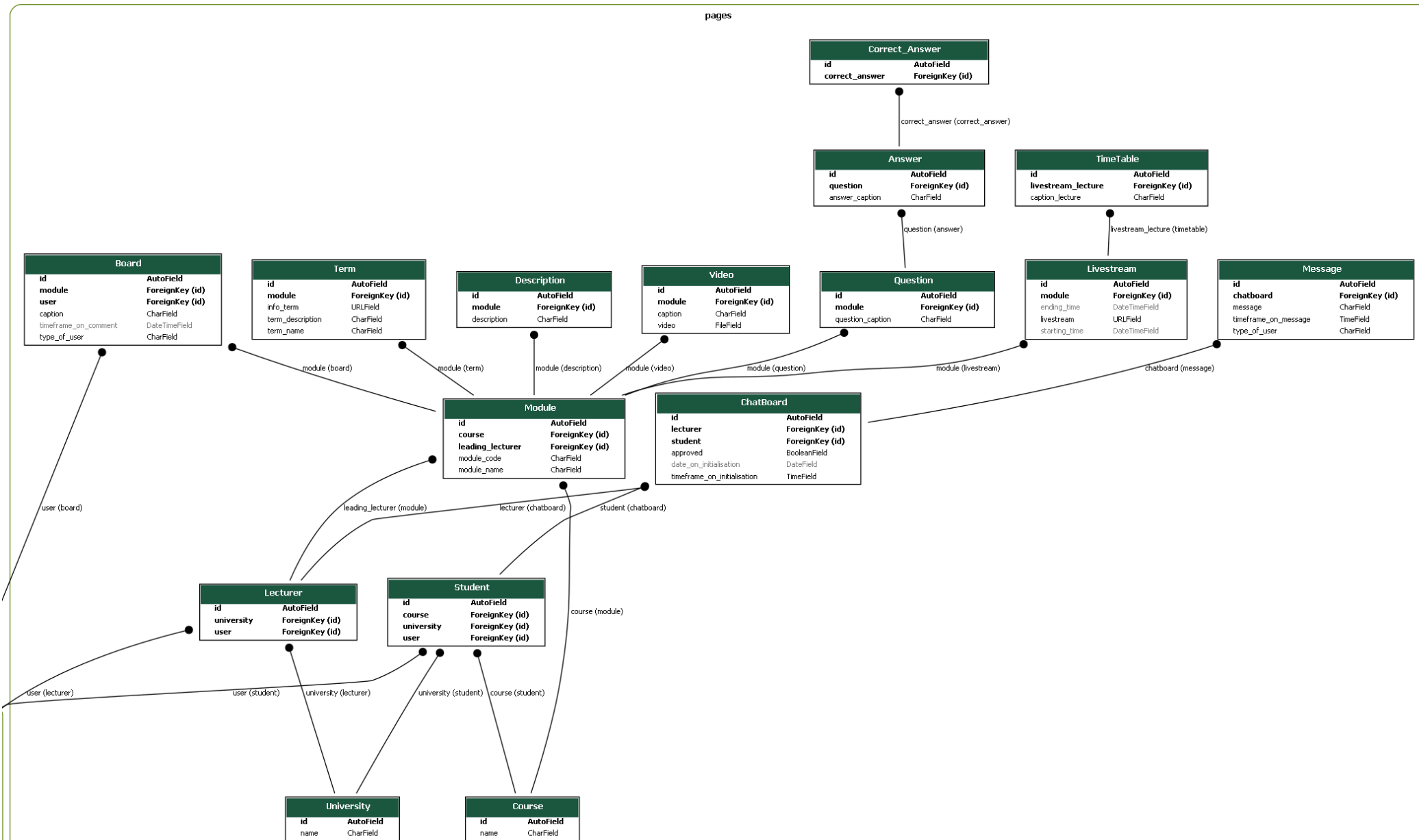
<p>Prototype No: 15</p>	
<p>Description</p>	<p>The user having access over the navigation bar(Both Lecturer and Student interface)</p> <p>The navigation bar has different options depending on the interface:</p> <p>Lecturer:</p> <ul style="list-style-type: none"> • Home screen • Modules screen • Approve a meeting with a student screen • Logout screen <p>Student:</p> <ul style="list-style-type: none"> • Home screen • Modules screen • Book a meeting with a lecturer screen • Chatbot screen • Logout screen <p>Related use cases: N/A</p>

Since I needed a basic MVP before I started implementing the system, I did not carry a heuristic evaluation using Nielsen's usability principles. The screens portrayed gave me a better understanding of what my task is and how I should design the interface. The project is considered as a proof-of-concept and the base screens

Implementation

Database design

ER (Entity Relationship) Diagram - Pages



Description of the ER Diagram – Pages

In Django, the pages section is for the main models/tables of the application. Portrayed in the diagram are the main models/tables of the web platform and the relationships in between them. The design of the database is in 3NF (Normalisation form) in order to reduce data duplication, avoid data anomalies, simplify the management of the data, and ensure integrity. To do so, the relationships between the tables are primarily many to one. In the following section, I will look further into the tables and their properties to justify the creation of the specific tables. Having said that, I would also discuss the relationships between the tables/models to justify them, as well.

Table	University
Properties	-name: CharField ()
Relationships	N/A
Justification	Fields: The university could have had many different properties, but in the specific case I did not need anything more, but the name of the university Relationships: N/A

Table	Lecturer
Properties	-user: ForeignKey (User) -university: ForeignKey (University)
Relationships	-University table -User table
Justification	Fields: N/A Relationships: The table of lecturer has a direct relation with the university that would store the university the lecturer is a part of. The primary User table that the Lecturer is related to will be discussed further into the Authentication part of the database design.

Table	Course
Properties	-name: CharField ()
Relationships	N/A
Justification	Fields: The course could have had many different properties, but in the specific case I did not need anything more, but the name of the course Relationships:

	N/A
--	-----

Table	Module
Properties	-course: ForeignKey (Course) -leading_lecturer: ForeignKey (Lecturer) -module_code: CharField () -module_name: CharField ()
Relationships	-Course table -Lecturer table
Justification	Fields: -module_code – is the specific module code of the module -module_name – is the specific module name of the module Relationships: The module table has two relationships with the course table and the lecturer table. I assumed that one lecturer could lead the module and that the module could be part of one course. That way every module has a course and leading lecturer.

Table	Student
Properties	-user: ForeignKey (User) -university: ForeignKey (University) -course: ForeignKey (Course)
Relationships	-User Table -University table -Course table
Justification	Fields: N/A Relationships: The student has three relationships with the primary user table, the university table, and the course table. I assumed that one student could be a part of one university and just one course.

Table	Video
Properties	-module: ForeignKey (Module) -video: FileField () -caption: CharField ()
Relationships	-Module table
Justification	Fields: -caption – is the title caption of specific the past recording

	-video – file field where the past recording is stored Relationships: The video table has one relationship. I assumed that one video (past recording) could be a part of just one module.
--	--

Table	Livestream
Properties	-module: ForeignKey (Module) -livestream: URLField () -starting_time: DateTimeField () -ending_time: DateTimeField ()
Relationships	-Module table
Justification	Fields: -livestream – is the URL to the live streaming lecture -starting_time – is the starting time of the live streaming lecture -ending_time – is the ending time of the live streaming lecture Relationships: The live stream table has one relationship. I assumed that a live streamable lecture could be a part of just one module.

Table	Board
Properties	-caption: CharField () -user: ForeignKey (User) -module: ForeignKey (Module) -timeframe_on_comment: DateTimeField () -type_of_user: CharField ()
Relationships	-Module table -User table
Justification	Fields: -caption – the comments caption of the discussion board -timeframe_on_comment – the time at which the comment was posted -type_of_user- is the type of user, either a student or a lecturer Relationships: The discussion board table has two relationships. I assumed that the discussion board should be

	module-specific. The user table is related as I assumed that comment on the discussion board should be related to a specific user.
--	--

Table	ChatBoard
Properties	-student: ForeignKey (Student) -lecturer: ForeignKey (Lecturer) -timeframe_on_initialisation: TimeField () -date_on_initialisation: DateField () -approved: BooleanField ()
Relationships	-Student table -Lecturer table
Justification	Fields: -timeframe_on_initialisation-date of the chatroom meeting -date_on_initialisation- date of the chatroom meeting -approved- Boolean field indicating if the chatroom meeting is approved by the lecturer Relationships: The ChatBoard table has two relationships. I assumed that for a chatroom meeting there should be a specific student and specific lecturer attending.

Table	Message
Properties	-ChatBoard: ForeignKey (ChatBoard) -message: CharField () -timeframe_on_message: TimeField () -type_of_user: CharField ()
Relationships	-ChatBoard table
Justification	Fields: -message-message at the chatroom meeting -timeframe_on_message- the time at which the message was posted at the chatroom meeting -type_of_user - is the type of user, either a student or a lecturer Relationships: The message table represents the messages sent in the chatroom meeting by both the lecturer and the student. I assumed that to make it meeting

	specific the message would need a chatroom meeting.
--	---

Table	Description
Properties	-description: CharField () -module: ForeignKey (Module)
Relationships	-Module table
Justification	Fields: Description – Short description of the module Relationships: The description table represents the description of a specific module. That is why I assumed that a description could have one module to refer to.

Table	Term
Properties	-term_name: CharField () -term_description: CharField () -module: ForeignKey (Module) -info_term: URLField ()
Relationships	-Module table
Justification	Fields: -term_name – name of the specified terminology -term_description – short description of the specified terminology -info_term – URL field leading into the Wikipedia page for the specified terminology Relationships: The table of terms represents the terminology studied in a module. I assumed that a term could be related to just one module.

Table	Timetable
Properties	-caption_lecture: CharField () -livestream_lecture: ForeignKey (Livestream)
Relationships	-Livestream table
Justification	Fields: -caption_lecture- the title caption of the lecture Relationships:

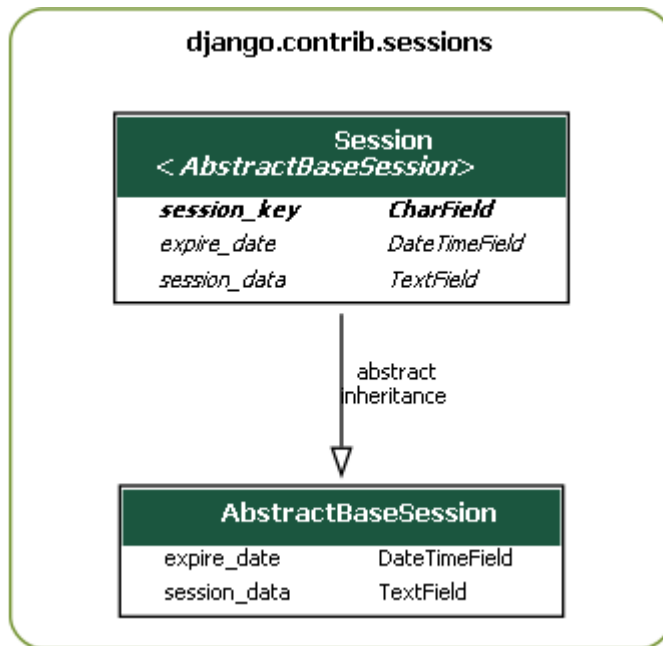
	The timetable table represents the timetable of a student. I assumed that a title of a live-streaming lecture would refer to just one lecture in particular
--	---

Table	Question
Properties	-module: ForeignKey (Module) -question_caption: CharField ()
Relationships	-Module table
Justification	Fields: -question_caption – the caption of the specific question Relationships: The question table represents the questions portrayed to the user when self-assessing. I assumed that a question should refer to a single module.

Table	Answer
Properties	-question: ForeignKey (Question) -answer_caption: CharField ()
Relationships	-Question table
Justification	Fields: -answer_caption – caption of the answer Relationships: The answer table represents the answers portrayed to the user when self-assessing. I assumed that an answer could refer to a single question.

Table	Correct_Answer
Properties	-correct_answer: ForeignKey (Answer)
Relationships	-Answer table
Justification	Fields: N/A Relationships: The correct_answer table represents the correct answer to a question of a user portrayed when self-assessing. I assumed that the correct_answer could be just one answer.

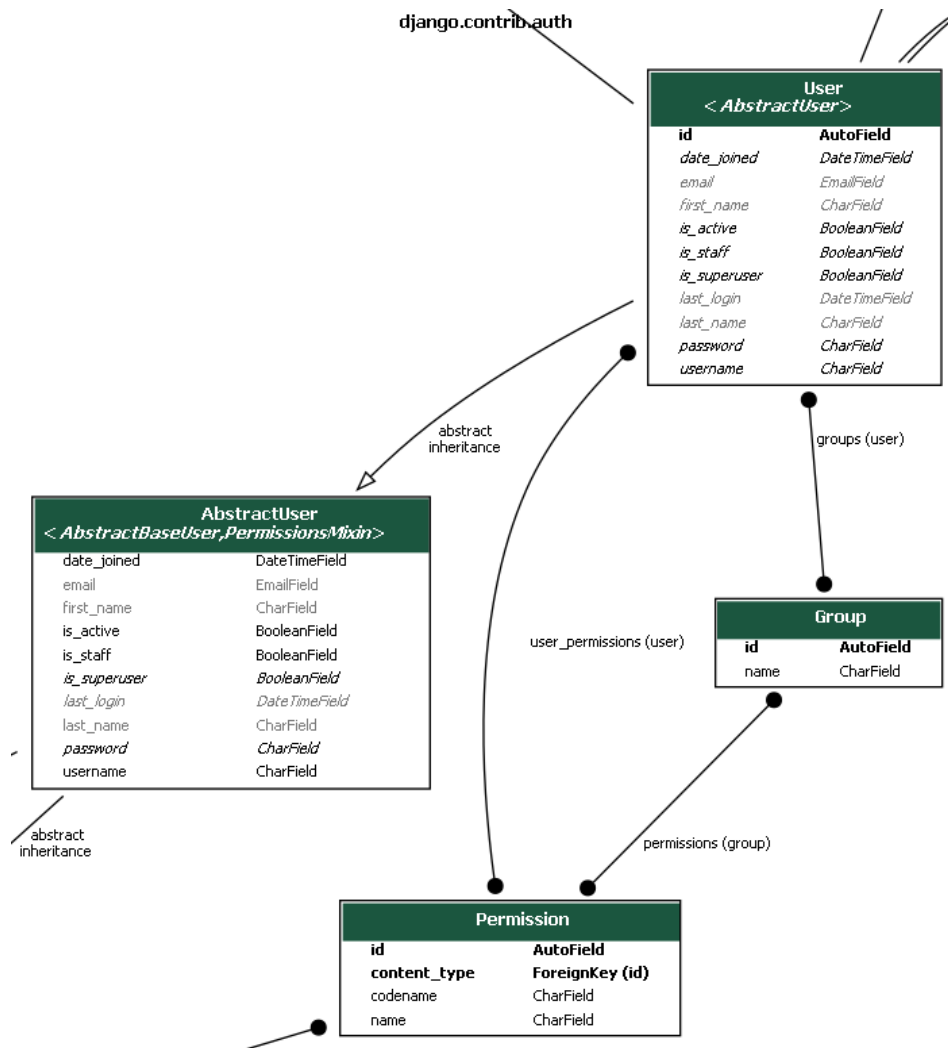
ER (Entity Relationship) Diagram – Sessions



Description of the ER Diagram – Sessions

In general, sessions are a group of user interactions within the platform. A single session might vary from logging in and uploading past recordings on the platform (Lecturer perspective) to logging and seeing them (Student perspective). Django as a framework provides full support for anonymous sessions. The session framework lets the admin store and retrieve arbitrary data on a basis per-site-visitor. It stores data on the server side and abstracts the sending and the receiving of cookies. Cookies contain a session ID, but not the data itself. By default Django stores the sessions on the default database – but it could be set up to save the data on a separate database. The sessions in Django work the same way as in other web frameworks such as (.Net) and (flask).

ER (Entity Relationship) Diagram - Authentication

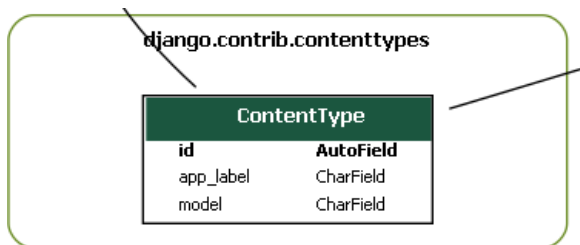


Description of the ER Diagram – Authentication

Django has a reliable authentication system. It handles user accounts, permissions, groups, and cookie-based user sessions. The Authentication system of Django handles both authentication and authorization. In general, authentication is verifying that the user is who he claims to be and authorization is what privileges are given to the user with respect to the functionality of the platform.

The authentication system consists of users, groups, permissions (binary flags of whether the user is permitted to do a specific task), restricting content and logging in users.

ER (Entity Relationship) Diagram – Content types



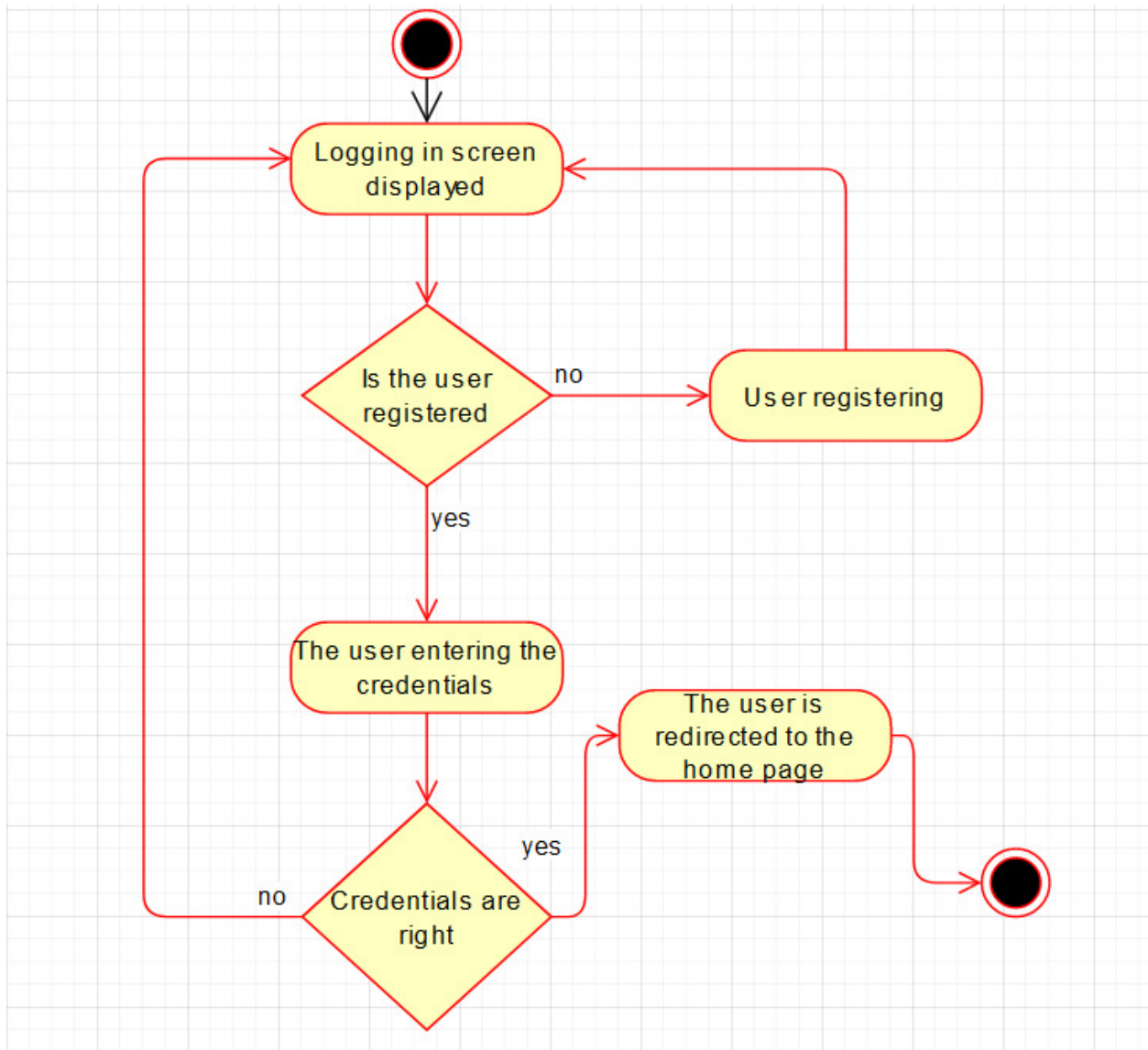
Description of the ER Diagram – Content types

Django includes a content types application that tracks the models installed in the project, providing a high-level, generic interface for working with the models/tables. Every model/table is represented as a row in the content type table, maintained and created by Django. There are many manipulations and operations that could be performed to a model/table using the Content-type module.

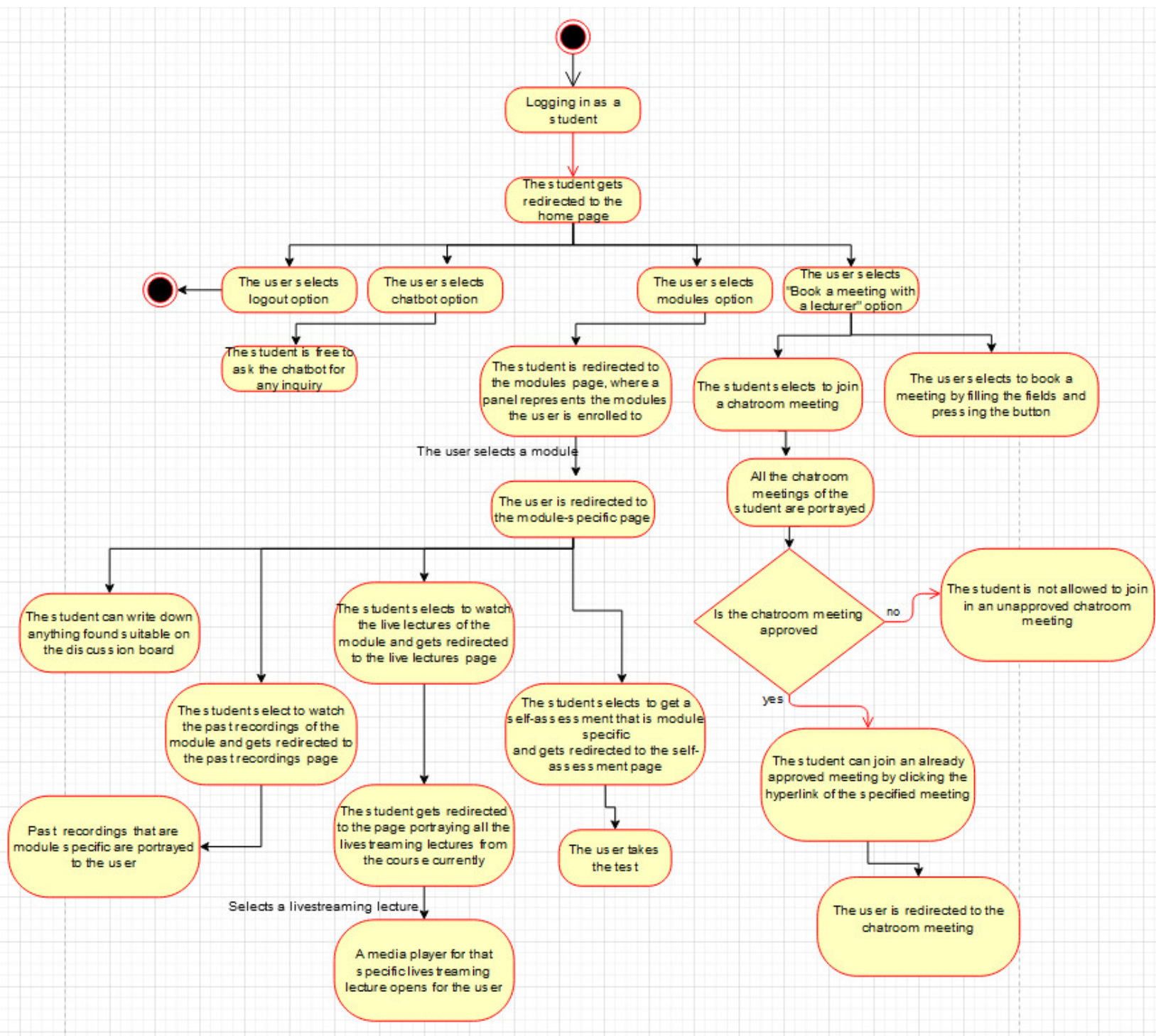
What Django does is using the content types table to reference a model or a table and link it with an id.

Activity diagrams

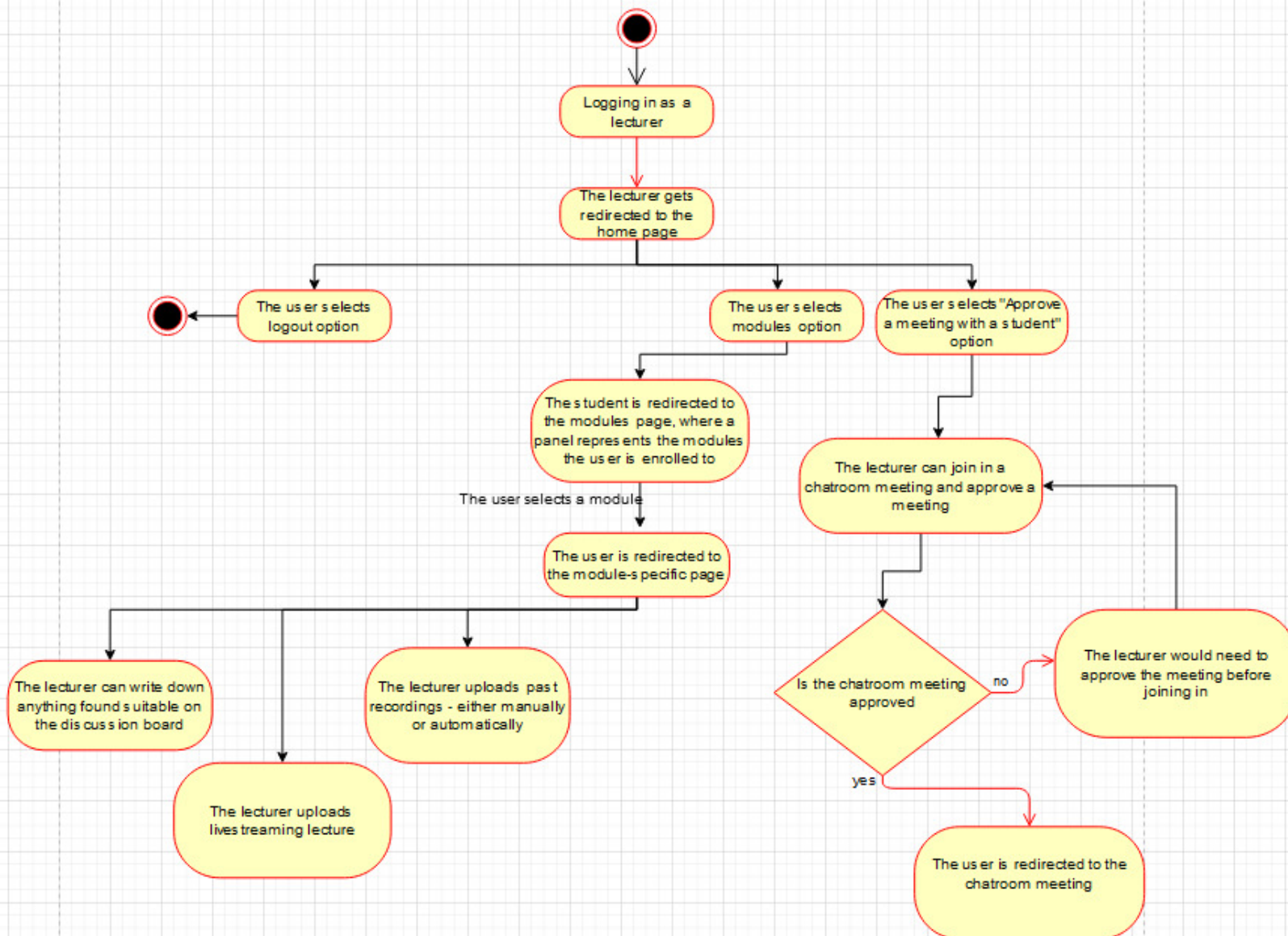
User logging in (Both Student and Lecturer interface)



Student interface



Lecturer interface



Overview of the design and implementation practices

There are several good practices when working with models/tables in Django. The database is one of the main aspects of the web platform and these practices were very useful when it came down to implementing the system. Furthermore, the database and the chatbot (The machine learning algorithm underneath) are the most complex topics of the platform and good practices should be held in mind in implementing.

The following good practices were taken into consideration whilst developing the platform:

- Correct Model/Table naming – Generally it is recommended to use single nouns for model naming
- Correct related-naming – It is reasonable to indicate as a plural the related-name that returns and query set
- Adding the models via migrations – By the commands makemigrations and migrate the admin keeps the database up to date
- Field duplication in Model Forms – When developing the model forms the model fields are not duplicated
- Flags misuse – In the models/tables the flags are not to be used as often, only when necessary

However, there are good practices when it comes down to code-writing as well. I implemented methods that would be commonly used in the system beforehand, so I could reduce code duplication. Furthermore, I did not hard-code any of the solutions to the problems I faced. During the development of the platform, I tend to try to write efficient code whilst it is readable. I used git as a version control system and keep my project as updated as possible, keeping a backup of it if needed. I have avoided deep nesting during the development process, also.

Detailed implementation

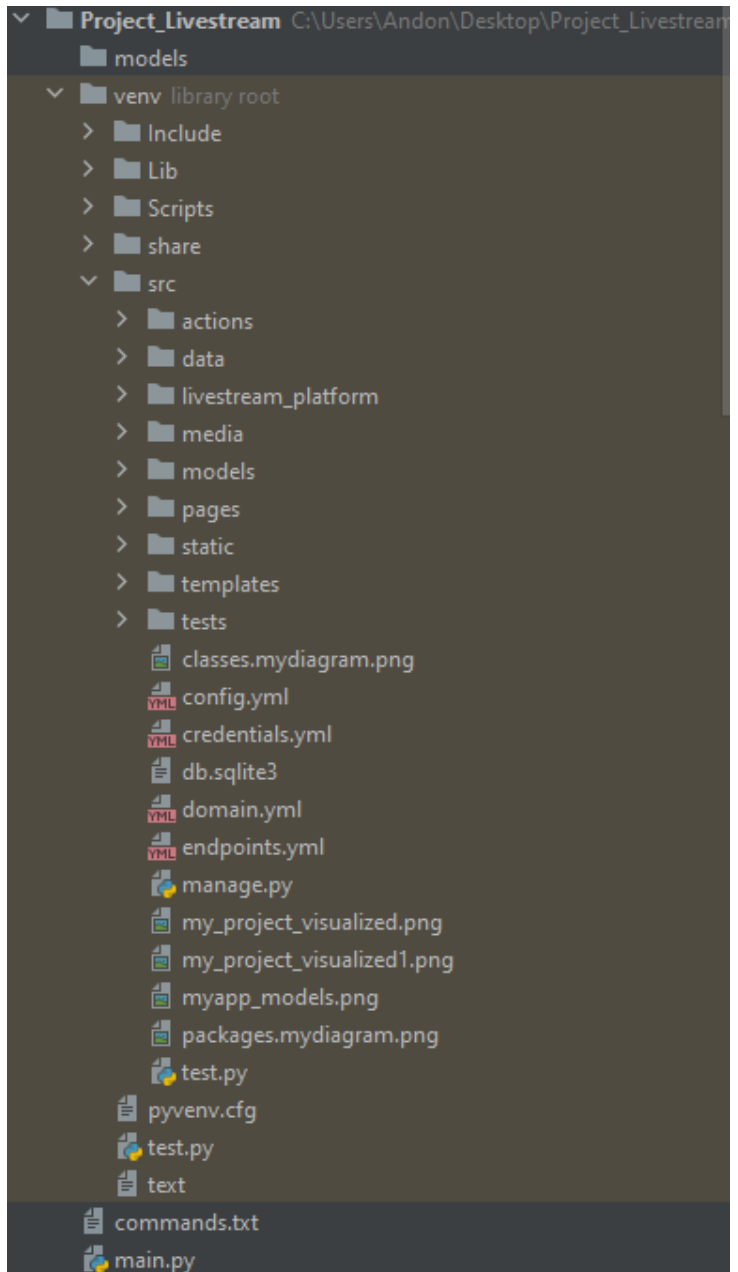
The Django platform

Overview of the implementation structure

The portrayed image is showing the structure of the project. Inside the virtual environment, there are five directories. In the include directories, I was running different scripts locally to test them before uploading them to the production of the project. Inside the Lib directory is all the libraries used in the project as packages. Having said that, the Scripts directory stores all the scripts for the libraries. Last, but not least in the src directory I have stored all the needed source files for the project. It stores actions, models of the machine learning algorithm and data – needed for the chatbot. On the other hand, in templates, static, pages, media, tests and livestream_platform are stored the Django files needed for the project.

The following data is stored in these directories:

- templates directory – the static HTML files
- static directory – the static javascript and bootstrap CSS
- pages – the main files of the Django project
- media – the media or more specifically the past-recordings
- livestream_platform – the configuration files of Django
- tests – the unit tests that I did not use in particular, but if I decide on working on the project afterwards I would.



User logging in and registering(Both lecturer and student interface)

Sign in

Sign up

Your username

username

Your password

Login

Register

First name

Last name

Email address

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

Enter the same password as before, for verification.

University*

▼

Course*

▼

Register

```

def login_view(request,*args, **kwargs):
    if request.method == "POST":
        username = request.POST.get("username")

        password = request.POST.get("password")

        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect("/home")
    return render(request, "login.html", {})

def register_view(request,*args, **kwargs):
    form = CreateUserForm()
    studentform = CreateStudentForm()
    if request.method == "POST":
        form = CreateUserForm(request.POST)
        studentform = CreateStudentForm(request.POST)
        if form.is_valid() and studentform.is_valid():
            user = form.save()
            student = studentform.save(commit=False)
            student.user = user
            studentform.save()
            return redirect("/login")
    return render(request, "register.html", {"form":form, "studentform":studentform})

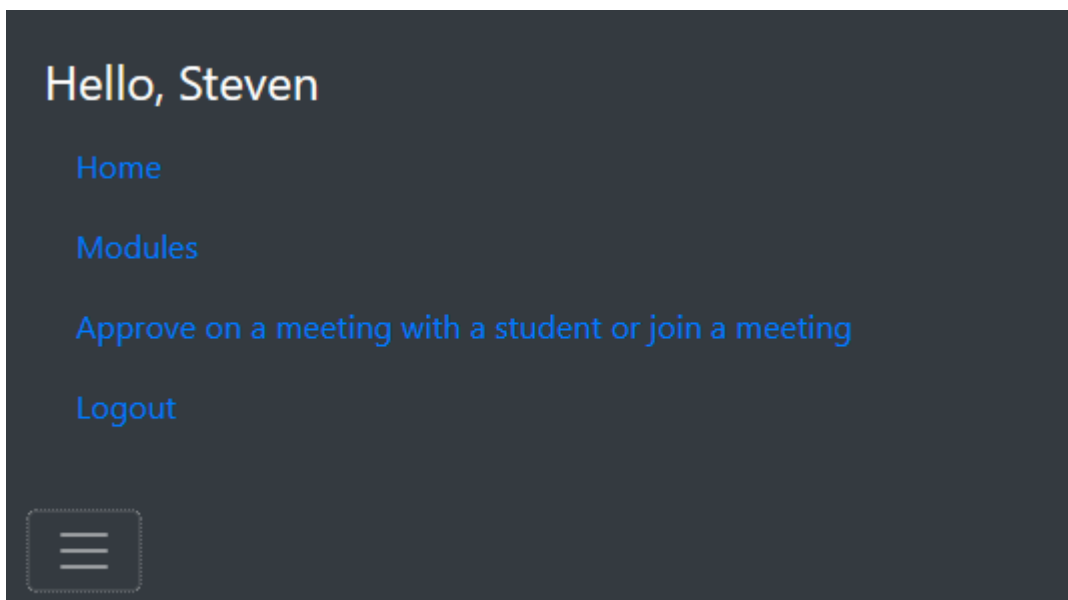
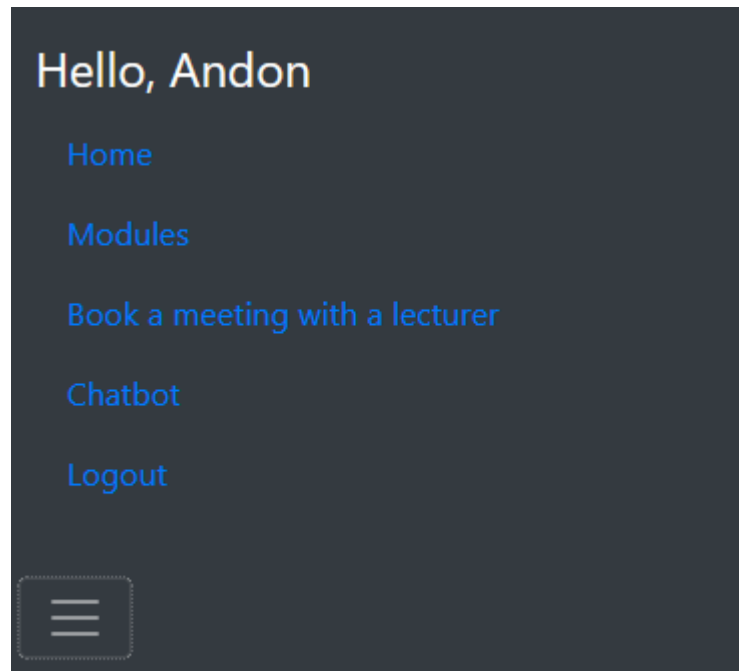
```

The shown pictures portray how to use is supposed to register and log into the system. The form asks for the first name, last name, email address, username, password, university and course as the first name, last name and email are part of the Django base user model.

In the shown code snippet are portrayed the login view and register view. As said earlier Django's MVT (Model – View – Template) allows the view to be the functional part of the framework. In the code snippet, the user is logged in by authenticating the password and the username with the system. If the user is not none and such user exists the login sequence is completed and the user is redirected to the home page.

On the other hand, when the user registers there are two forms to be filled. One is for the base user of Django, which is a UserForm and a student form – StudentForm. If both forms are valid the base user is filled with the details and so is the student user. If the registration is successful the user is redirected to the login page, where he/she would log into the system.

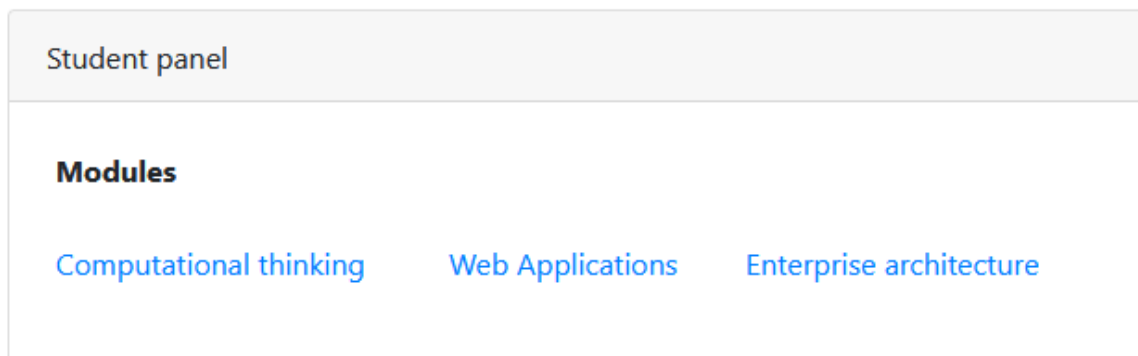
Navigation around the platform and home page(Both Student and Lecturer interface)



The navigation all around the platform is done through a navigation bar at the top of the pages. It is different to the type of user having the session.

Both the student and the lecturer could enter the modules panel by clicking on the module section of the navigation bar. The student can press the “book a meeting with a lecturer” hyperlink and get redirected to the book a meeting with a lecturer panel. Also, the student can enter into the chatbot page, where the chatbot could be asked many questions. Both the student and the lecturer have a logout section in the navigation bar if they would like to close their session. On the contrary, the lecturer can press the button “Approve on a meeting with a student or join a meeting” where the lecturer would get redirected to the meeting's panel directly.

Modules panel(Both Student and Lecturer interface)



```
@login_required(login_url="/login")
def modules_view(request, *args, **kwargs):
    modules_of_the_user = []
    modules_of_the_user.clear()
    students = Student.objects.all()
    lecturers = Lecturer.objects.all()
    modules = Module.objects.all()
    course = None
    type_of_user = None
    current_user = None
    for student, lecturer in zip(students, lecturers):
        if student.user == request.user:
            course = student.course
            current_user = student
        elif lecturer.user == request.user:
            current_user = lecturer
    for module in modules:
        if module.leading_lecturer.user == request.user and module.leading_lecturer.university == current_user.university:
            modules_of_the_user.append(module.module_name)
        elif module.course == course and module.leading_lecturer.university == current_user.university:
            modules_of_the_user.append(module.module_name)
        else:
            continue
    type_of_user = validation_user(request, students, type_of_user)
    return render(request, "modules.html", {"modules_of_the_user": modules_of_the_user,
                                             "type_of_user": type_of_user})
```

The panel portrays the enrolled modules of the user if a student, or the modules that the user is a part of if a lecturer. The code - extracts the current user and then for every module in the database checks if the user is part of the course or if is enrolled on the course. All of the modules portrayed on the panel are university-specific as that is one of the main requirements as otherwise, it might cause potential security, privacy and copyright issues.

Module-specific page (Student interface)



Specific to the Module

[Past recordings of lectures](#)

[Lectures live](#)

[Self assessment of the material covered](#)

Discussion board

Discussion Board:

Feb. 26, 2021, 4:58 p.m. - Lecturer Steven Hristov : dasdas

Feb. 26, 2021, 4:59 p.m. - Student Andon Gribachev : dsadas

Feb. 26, 2021, 5:16 p.m. - Lecturer Steven Hristov : dsadas

March 2, 2021, 12:57 p.m. - Student Andon Gribachev : I have a question

March 2, 2021, 1 p.m. - Lecturer Steven Hristov : Alright

March 7, 2021, 10:35 p.m. - Student Andon Gribachev : Zdraveite

April 1, 2021, 4:20 p.m. - student Andon Gribachev : dsadsadas

April 8, 2021, 11:29 p.m. - Student Andon Gribachev : dsadsadasdasdasdas

April 8, 2021, 11:29 p.m. - Student Andon Gribachev : dsadsadasdasdasdas

Caption*

Submit

Add a past lecture

Video*

Browse...

No file selected.

Add the video

Automatically upload past recordings

Discussion board

Discussion Board:

Feb. 26, 2021, 4:58 p.m. - Lecturer Steven Hristov : dasdas

Feb. 26, 2021, 4:59 p.m. - Student Andon Gribachev : dsadas

Feb. 26, 2021, 5:16 p.m. - Lecturer Steven Hristov : dsadas

March 2, 2021, 12:57 p.m. - Student Andon Gribachev : I have a question

March 2, 2021, 1 p.m. - Lecturer Steven Hristov : Alright

March 7, 2021, 10:35 p.m. - Student Andon Gribachev : Zdraveite

April 1, 2021, 4:20 p.m. - student Andon Gribachev : dsadsadas

April 8, 2021, 11:29 p.m. - Student Andon Gribachev : dsadsadasdasdasdas

April 8, 2021, 11:29 p.m. - Student Andon Gribachev : dsadsadasdasdasdas

Caption*

Submit

Add a livestream

Add URL to the livestream
Livestream*

Add caption to the livestream lecture
Caption lecture*

Add starting time of the livestream
Starting time

Add ending time of the livestream
Ending time

Add the livestream

71

Discussion of the module-specific page(Both Student and Lecturer interface)

```
@login_required(login_url="/login")
def module_view(request,module, *args, **kwargs):
    modules = Module.objects.all()
    current_module = current_module_return(modules, module)
    videos = Video.objects.all()
    discussion_board_form = None
    videoform = None
    livestreamform = None
    automaticvideoform = None
    timetableform = TimeTableForm()
    discussion_board = Board.objects.all()
    discussion = []
    type_of_user = None
    students = Student.objects.all()
    lecturers = Lecturer.objects.all()
    for board in discussion_board:
        if board.module == current_module:
            discussion.append(board)
    if request.method == "POST":
        timetableform = TimeTableForm(request.POST)
        videoform = VideoForm(request.POST, request.FILES)
        discussion_board_form = DiscussionBoardForm(request.POST)
        livestreamform = LivestreamForm(request.POST)
        automaticvideoform = AutomaticVideoForm(request.POST)
        if discussion_board_form.is_valid():
            discussion_board_form.instance.user = request.user
            discussion_board_form.instance.timeframe_on_comment = datetime.now()
            discussion_board_form.instance.module = current_module
            type_of_user = validation_user(request, students, type_of_user)
            discussion_board_form.instance.type_of_user = type_of_user
            discussion_board_form.save()
```



```

elif videoform.is_valid():
    videoform.instance.module = current_module
    videoform.save()
elif livestreamform.is_valid() and timetableform.is_valid():
    livestreamform.instance.module = current_module
    livestreamform.instance.livestream = transform_yt_link(livestreamform.instance.livestream)
    timetableform.instance.livestream_lecture = livestreamform.instance
    livestreamform.save()
    timetableform.save()
elif automaticvideoform.is_valid():
    pathing = join(os.getcwd(), "media\\pastrecordings\\")
    pathing_dirs = listdir(pathing)
    absolute_path_to_video = []
    for dir in pathing_dirs:
        if dir == current_module.module_name:
            pathing += current_module.module_name
            pathing_dirs = listdir(pathing)
            onlyvideo = [v for v in pathing_dirs if isfile(join(pathing, v))]
            for video in onlyvideo:
                absolute_path_to_video.append(pathing + "\\" + video)
    if videos:
        url_of_videos = [v.video.path for v in videos if v.module == current_module]
        for vidPath in absolute_path_to_video:
            if vidPath in url_of_videos:
                continue
            else:
                automaticvideoform.instance.pk = None
                automaticvideoform.instance.module = current_module
                automaticvideoform.instance.video = vidPath
                automaticvideoform.save()

```

```

        else:
            for vidPath in absolute_path_to_video:
                automaticvideoform.instance.pk = None
                automaticvideoform.instance.module = current_module
                automaticvideoform.instance.video = vidPath
                automaticvideoform.save()

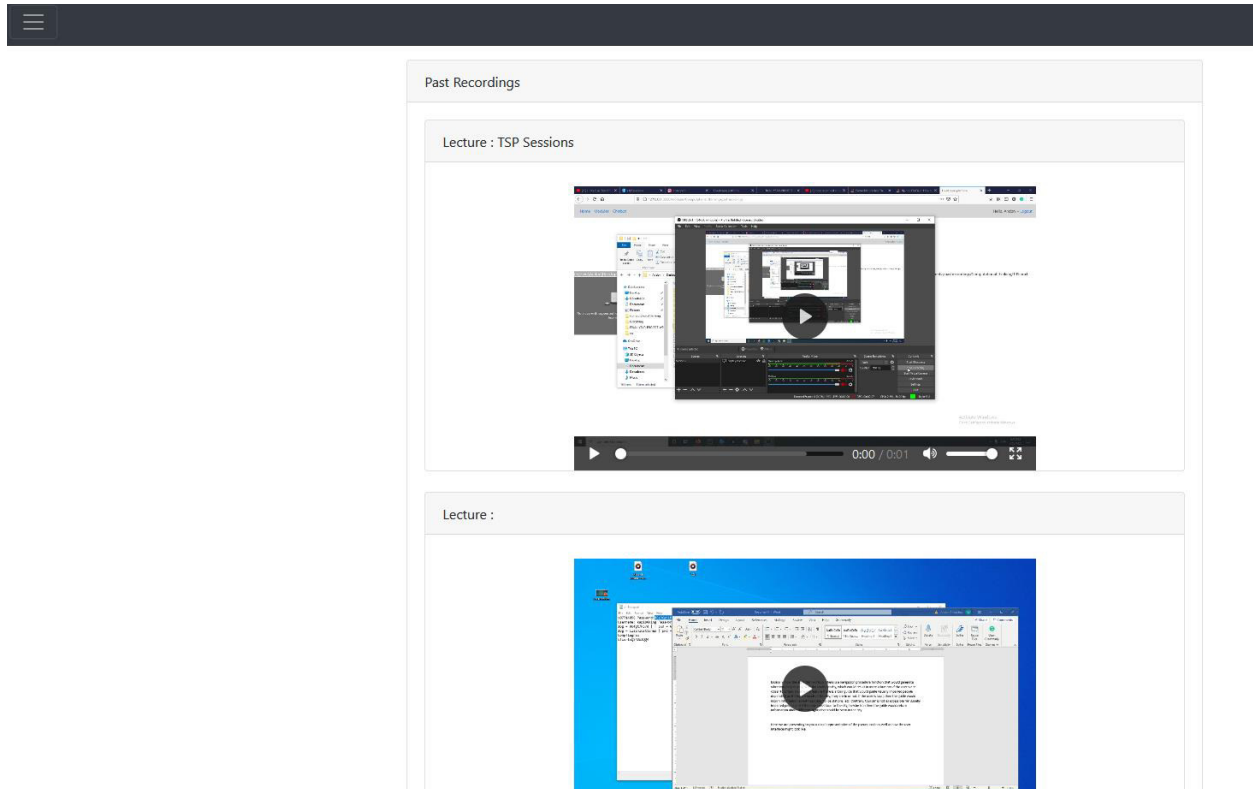
    else:
        livestreamform = LivestreamForm(instance=current_module)
        videoform = VideoForm(instance=current_module)
        discussion_board_form = DiscussionBoardForm(instance=current_module)
        type_of_user = validation_user(request, students, type_of_user)
        return render(request, "module.html", {"discussion": discussion,
                                                "discussion_board_form": discussion_board_form,
                                                "type_of_user": type_of_user,
                                                "videoform": videoform,
                                                "livestreamform": livestreamform,
                                                "timetableform": timetableform,
                                                "current_module": current_module.module_name})

```

The code snippet generates the module-specific page for the user. Depending on if the user is either a student or a lecturer, different forms are portrayed on the interface as shown in the images. As could be seen in the two images of the interfaces, the student has access to the module-specific panel of live lectures, past recordings and self-assessment with the discussion board. When it comes down to the lecturer, however, the discussion board is also portrayed, but the lecturer can upload past recordings both manually and automatically. Also, the lecturer can upload a live-streaming lecture.

Even though, the code snippet might look quite large its' functionality is quite basic. It generates 5 forms for all the users and checks if they are valid. The more complex part of the code is how the lecturer can upload past recordings automatically. Within the media directory, there is a sub-directory of past recordings, where each module has a directory. If the lecturer puts the video file into a specific module directory then the algorithm would take the file from the said directory and would import it automatically without the caption/title of the past recording. The algorithm is executed when the lecturer clicks on automatically upload past recordings. Since it checks the database first for the videos within it, no duplications of module-specific past recordings could be uploaded.


Past recordings page (Student interface)



```
@login_required(login_url="/login")
def module_past_recordings_view(request,module, *args, **kwargs):
    students = Student.objects.all()
    type_of_user = None
    type_of_user = validation_user(request,students,type_of_user)
    modules = Module.objects.all()
    current_module = current_module_return(modules, module)
    videos = Video.objects.all()
    videos_for_specific_module = []
    for k in videos:
        if k.module == current_module:
            videos_for_specific_module.append(k)
    return render(request, "module_past_recordings.html", {"videos_for_specific_module":videos_for_specific_module,
                                                            "type_of_user":type_of_user})
```

On the past recordings page, all the module-specific past recordings are displayed with their titles/captions. Shortly, the code snippet shows that only the module-specific past recordings are returned to the front-end bit of the platform.

Live lectures panel (Student interface)



Livestreamable lectures

[Basics of Web computing](#)
Led by Steven Hristov at April 2, 2021, 8:06 a.m.
[Basics of the HTTP](#)
Led by Steven Hristov at April 5, 2020, 4:22 p.m.

```

@login_required(login_url="/login")
def module_live_streaming_view(request,module, *args, **kwargs):
    modules = Module.objects.all()
    current_module = current_module_return(modules, module)
    livestreams = Livestream.objects.all()
    students = Student.objects.all()
    type_of_user = None
    type_of_user = validation_user(request,students,type_of_user)
    livestream_object = None
    hyper_link_for_livestream = []
    timetables = TimeTable.objects.all()
    for livestream in livestreams:
        if livestream.module == current_module:
            livestream_object = livestream
    for timetable in timetables:
        if timetable.livestream_lecture.module == current_module:
            hyper_link_for_livestream.append(timetable)
    len_hyper_link = len(hyper_link_for_livestream)
    return render(request, "module_live_stream.html", {"hyper_link_for_livestream":hyper_link_for_livestream,
                                                         "len_hyper_link": len_hyper_link,
                                                         "type_of_user":type_of_user})

```

As portrayed in the image, the caption/title of the live stream is represented as a hyperlink. Only the live-streaming lectures that are module-specific are shown to the user and this is the main functionality of the view

Livestream page (Student interface)

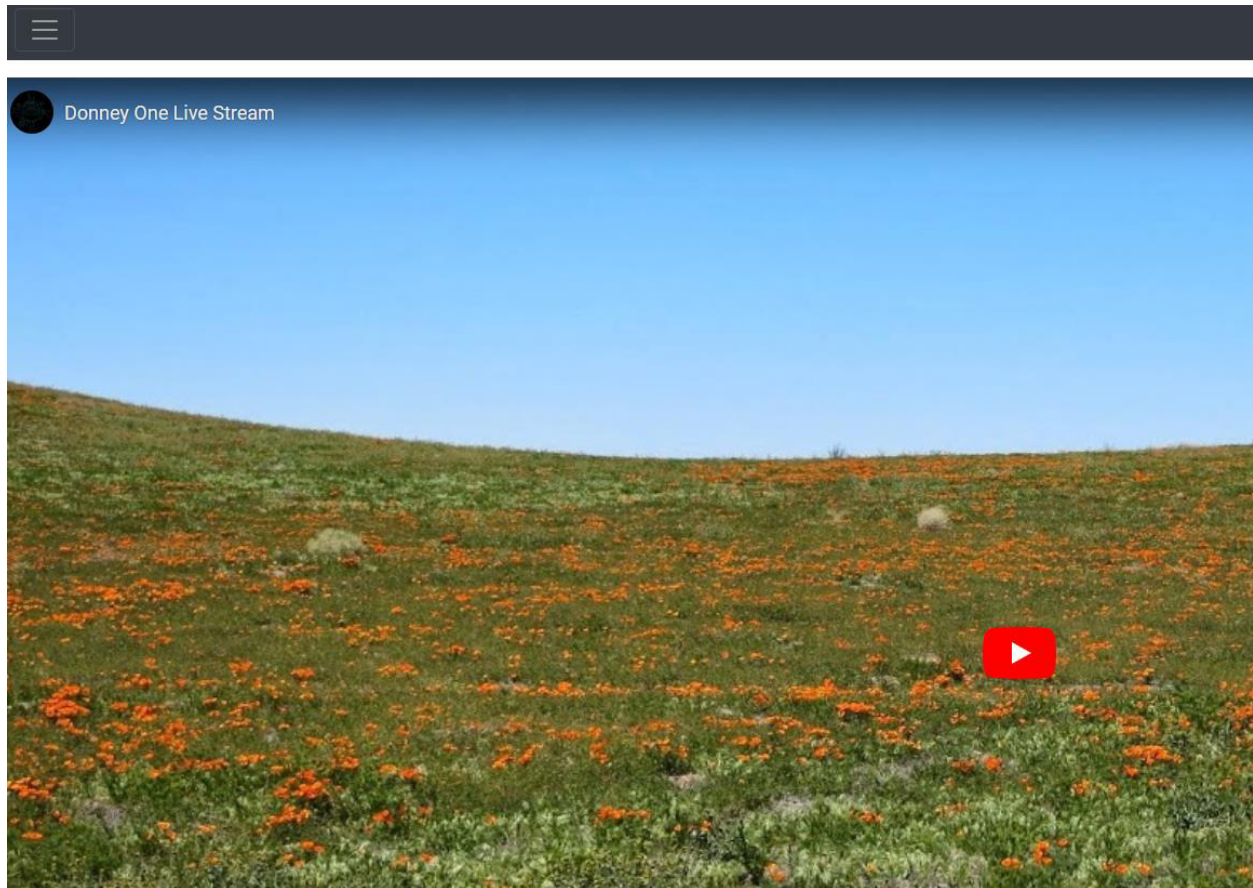
```

@login_required(login_url="/login")
def live_stream_page_view(request,module,caption, *args, **kwargs):
    url_of_livestream = None
    modules = Module.objects.all()
    livestreams = Livestream.objects.all()
    current_module = current_module_return(modules, module)
    timetables = TimeTable.objects.all()
    timetable_object = []
    students = Student.objects.all()
    type_of_user = None
    type_of_user = validation_user(request,students,type_of_user)
    for timetable in timetables:
        if timetable.livestream_lecture.module == current_module:
            timetable_object.append(timetable)

    for object in timetable_object:
        if object.caption_lecture == caption:
            url_of_livestream = transform_yt_link(object.livestream_lecture.livestream)
    return render(request, "live_stream_page.html", {"url_of_livestream":url_of_livestream,
                                                         "type_of_user":type_of_user})


```

```
def transform_yt_link(link):  
    yt_transform = link.split(".")  
    return yt_transform[0] + "be.com/" + "embed" + yt_transform[1][2::]
```



As portrayed in the image of the interface, the live-streaming lecture is being represented to the student. The view functionality includes further checks if the lecture is module-specific and uses the method “transform_yt_link” to manipulate the string to make it the appropriate hyperlink to the live streaming lecture on YouTube.

Self-assessment page (Student interface)



Self-assessment

What is ORM?
 ffffffffffffffffff ☐

Object-relational mapper ☐

ddddddddddddddddd ☐

aaaaaaaaaaaaaaaaa ☐

Which is a HTTP request method?
 ffffffffffffffff ☐

xxxxxxxxxxxxxxxxxxxx ☐

cccccccccccc ☐

POST ☐

Submit

```

@login_required(login_url="/login")
def self_assessment(request,module, *args, **kwargs):
    questions = Question.objects.all()
    correct_answer = Correct_Answer.objects.all()
    answers = Answer.objects.all()
    modules = Module.objects.all()
    questions_answers = []
    questions_specific_to_module = []
    answers_specific_to_questions = []
    temp = []
    temp_ans = {}
    temp_correct_answer = []
    counter = 0
    current_module = current_module_return(modules,module)
    random_questions = []
    students = Student.objects.all()
    type_of_user = None
    type_of_user = validation_user(request,students,type_of_user)
    for q in questions:
        if q.module == current_module:
            questions_specific_to_module.append(q)
    for a in answers:
        if a.question in questions_specific_to_module:
            answers_specific_to_questions.append(a)
    amount_of_questions = len(questions_specific_to_module)
    try:
        r = random.sample(range(1, amount_of_questions+1), 2)
        for x in r:
            random_questions.append(questions_specific_to_module[int(x-1)])
        random_questions_len = len(random_questions)

```

```


        if a.question in questions_specific_to_module:
            answers_specific_to_questions.append(a)
    amount_of_questions = len(questions_specific_to_module)
    try:
        r = random.sample(range(1, amount_of_questions+1), 2)
        for x in r:
            random_questions.append(questions_specific_to_module[int(x-1)])
        random_questions_len = len(random_questions)
        for c in correct_answer:
            temp_correct_answer.append(c.correct_answer.answer_caption)
        for que in random_questions:
            temp_ans2 = []
            for ans in answers_specific_to_questions:
                if ans.question == que:
                    temp_ans2.append(ans)
                    temp_ans[que] = temp_ans2
    except:
        pass
    if request.method == "POST":
        for corr in temp_correct_answer:
            for x,y in request.POST.items():
                if y in corr:
                    counter += 1
        messages.info(request, f"You have scored {counter} out of {random_questions_len}")

    return render(request, "module_self_assessment.html", {"temp_ans":temp_ans,
                                                            "type_of_user":type_of_user})

```

On the image of the interface is portrayed the self-assessment page, where the student can self-assess given skillset acquired in a specified module. The algorithm underneath the page works by filtering the module related questions and answers. Then it randomises the questions from the database and their specified answers to present the user unique answers and questions. If the user fills in the form/self-assessment test and presses submit the counter would count how many correct answers did the user have and return the result of the self-assessment to the user.

Chatbot page (Student interface)



Personal student assistant

Hello, what could I help you with, Andon?

less than a minute

SUBMIT


```

@login_required(login_url="/login")
def chatbot_view(request, *args, **kwargs):
    students = Student.objects.all()
    type_of_user = None
    type_of_user = validation_user(request, students, type_of_user)
    url = 'http://localhost:5005/conversations/default/tracker/events?include_events=NONE'
    myobj = {"event": "slot", "name": "User_id", "value": request.user.id, "timestamp": 0}
    json_object = json.dumps(myobj, indent=4)
    x = requests.post(url, data=json_object)
    url2 = 'http://localhost:5005/conversations/default/execute'
    myobj2 = {"name": "action_greeting"}
    json_object2 = json.dumps(myobj2, indent=4)
    z = requests.post(url2, data=json_object2)
    return render(request, "chatbot.html", {"x": x,
                                             "type_of_user": type_of_user})

```

On the first image, it could be seen the chatbot interface. To make it user-specific I had to write a script that would be used for the two servers: - the chatbot server; – the Django server; to correspond. As seen in the code snippet I send data from the Django server two times. The first time is to send data to the chatbot server and the second one is to give a command over the first action that the chatbot should execute. The chatbot page is a bit different from the other pages on the system. It is different by the way it works. The user might need 5-6 seconds before the two servers connect and send the data in between them. The chatbot itself would be discussed further in the next section.

Book a meeting with a lecturer page/panel (Student interface)



Book a meeting with a lecturer

Timeframe on initialisation*

Date on initialisation

Lecturer*

Book a meeting

Join a meeting

```
@login_required(login_url="/login")
def chat(request, *args, **kwargs):
    students = Student.objects.all()
    type_of_user = None
    type_of_user = validation_user(request, students, type_of_user)
    lecturers = Lecturer.objects.all()
    chatboard_form = ChatBoardForm()
    for student in students:
        if student.user == request.user:
            type_of_user = "Student"
            specific_student = student
    if request.method == "POST":
        chatboard_form = ChatBoardForm(request.POST)
        if chatboard_form.is_valid():
            chatboard_form.instance.student = specific_student
            chatboard_form.instance.approved = False
            chatboard_form.save()
    return render(request, "chat.html", {"chatboard_form": chatboard_form,
                                         "type_of_user": type_of_user})
```

As portrayed on the image of the interface the student can book a chatroom meeting with a lecturer by filling in the date of the meeting, the time of the meeting and the lecturer - the student wants to meet with. The code snippet is not as complex it has one form and validation of the form itself. The form is used to store a chatroom meeting between the student and the lecturer.

Approve a meeting with a student or join a chatroom meeting page/panel (Lecturer interface)



Chats initiated

Chatroom meeting between Andon Gribachev and Steven Hristov at: 14:22:00 on 2001-12-04
Time when it takes places in: Dec. 4, 2001 2:22 p.m.

Chatroom meeting between Andon Gribachev and Steven Hristov at: 14:22:00 on 2001-12-04
Time when it takes places in: Dec. 4, 2001 2:22 p.m.

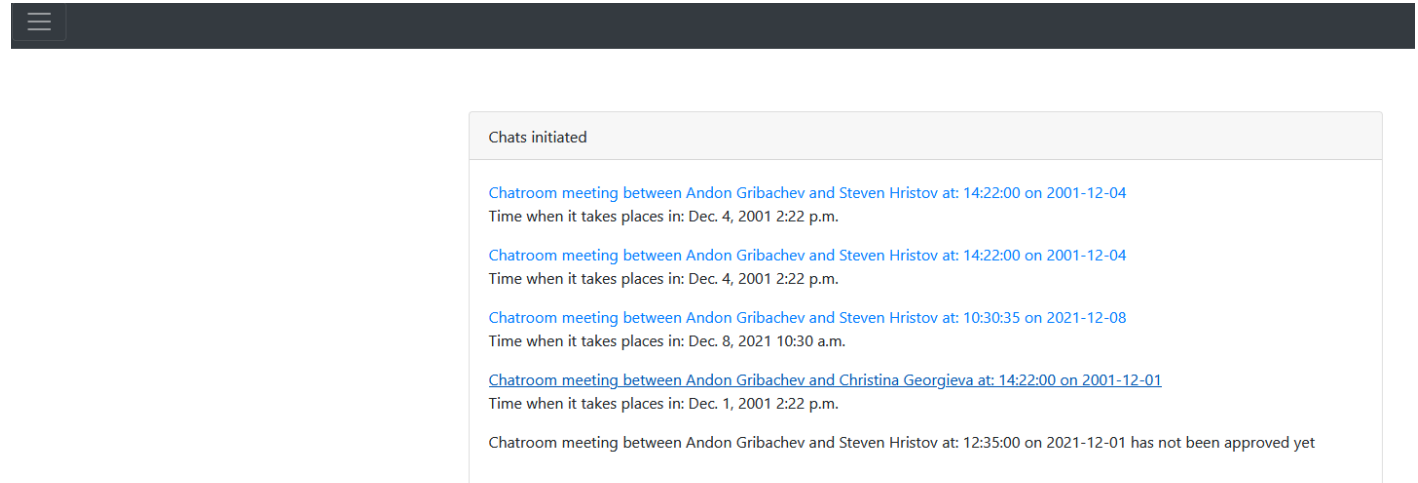
Chatroom meeting between Andon Gribachev and Steven Hristov at: 10:30:35 on 2021-12-08
Time when it takes places in: Dec. 8, 2021 10:30 a.m.

Chatroom meeting between Andon Gribachev and Steven Hristov at: 12:35:00 on 2021-12-01 Approve on this meeting

```
@login_required(login_url="/login")
def meetings_lec(request, *args, **kwargs):
    meetings_chatboard = ChatBoard.objects.all()
    students = Student.objects.all()
    lecturers = Lecturer.objects.all()
    meetings = []
    lec_chatboard_form = Lec_ChatBoardForm()
    for lecturer in lecturers:
        if lecturer.user == request.user:
            type_of_user = "Lecturer"
            specific_lecturer = lecturer
        else:
            type_of_user = "Student"
    for meeting in meetings_chatboard:
        if meeting.lecturer == specific_lecturer:
            meetings.append(meeting)
    if request.method == "POST":
        lec_chatboard_form = Lec_ChatBoardForm(request.POST)
        for meeting in meetings:
            if request.POST.get(str(meeting)):
                meeting.__setattr__("approved", True)
                lec_chatboard_form.instance = meeting
                lec_chatboard_form.save()
    return render(request, "meetings_lec.html", {"meetings":meetings,
                                                "lec_chatboard_form":lec_chatboard_form,
                                                "type_of_user":type_of_user})
```

The chatroom meetings are presented to the lecturer as hyperlinks as could be seen on the interface image. The lecturer has permission to approve the meeting. The code snippet represents just that as it is not as complex. The lecturer is allowed to change the flag of the chatroom meeting and set it to approved. After the meeting is approved, it would appear to both the student and the lecturer as a hyperlink to the chatroom meeting.


Join a meeting (Student interface)



```
@login_required(login_url="/login")
def meetings(request, *args, **kwargs):
    meetings_chatboard = ChatBoard.objects.all()
    students = Student.objects.all()
    lecturers = Lecturer.objects.all()
    meetings = []
    for student in students:
        if student.user == request.user:
            type_of_user = "Student"
            specific_student = student
    for meeting in meetings_chatboard:
        if meeting.student == specific_student:
            meetings.append(meeting)
    return render(request, "meetings.html", {"meetings":meetings,
                                             "type_of_user":type_of_user})
```

As portrayed the unapproved meetings are presented as a text and the approved ones as a hyperlink. The code snippet is very readable as the only thing it does - is to return the meetings of the specified user and not all of them.

Chatroom meeting page (Both Lecturer and Student interface)



Chatroom

1:02 p.m.:dasdasdasdas
Sent by Lecturer - Steven Hristov
1:07 p.m.:dsadasdas
Sent by Student - Andon Gribachev
12:32 p.m.:hello
Sent by Student - Andon Gribachev
Message*

Submit

```

@login_required(login_url="/login")
def meeting(request, meeting, *args, **kwargs):
    meetings_chatboard = ChatBoard.objects.all()
    students = Student.objects.all()
    lecturers = Lecturer.objects.all()
    messages = Message.objects.all()
    chat_form = Chat_Form()
    messages_specific_to_meeting = []
    for student in students:
        if student.user == request.user:
            type_of_user = "Student"
    for lecturer in lecturers:
        if lecturer.user == request.user:
            type_of_user = "Lecturer"
    for meeting2 in meetings_chatboard:
        if str(meeting) == str(meeting2):
            meeting = meeting2
    for m in messages:
        if str(m.chatboard) == str(meeting):
            messages_specific_to_meeting.append(m)
    if request.method == "POST":
        chat_form = Chat_Form(request.POST)
        if chat_form.is_valid():
            chat_form.instance.chatboard = meeting
            chat_form.instance.timeframe_on_message = datetime.now()
            chat_form.instance.type_of_user = type_of_user
            chat_form.save()
    return render(request, "meeting.html", {"chat_form": chat_form,
                                             "messages_specific_to_meeting": messages_specific_to_meeting,
                                             "type_of_user": type_of_user})

```

The chatroom meeting itself is not as complex. Within the chatroom meeting, there is one form that records the message of the user and stores it directly to the database. There is a validation of the form, also. The form stores dynamically the time, the specific meeting it is sent to and the type of user that sent the message.

The chatbot

For the chatbot implementation, I decided on using the RASA API library in python. It works with machine learning algorithms beneath the surface of it. In particular, the RASA API comes with in-built actions (which could be executed at certain stories with intents), intents (which are defined by different keyword examples for the NLU (Natural language understanding) engine), stories (which define the sequence of executing certain actions when certain intent is detected) and domain – where everything is defined. Unfortunately, the RASA API does not work autonomously as the admin would need to train the bot after every change in the intents or the stories is made.

Example of intent:

```
- intent: module
  examples: |
    - Module of {Module}
    - {Module}
    - module description of {Module}
    - description of {Module}

- intent: terms
  examples: |
    - Terms in {Module}
    - More info on terms in {Module}

- intent: timetable
  examples: |
    - Timetable
    - show me my timetable
    - my timetable

- intent: leading_lecturer
  examples: |
    - Module leader of {Module}
    - leader of {Module}
    - leading lecturer of {Module}
    - The team that leads the {Module}
```

Example of action:

```
class ActionLeadingLecturer(Action):
    def name(self) -> Text:
        return "action_leading_lecturer"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        mes = ""
        x = query_method(
            "SELECT first_name, last_name, module_name FROM pages_lecturer l, pages_module m, auth_user u WHERE u.id = l.user_id AND m.leading_lecturer_id = l.id"
            , "db.sqlite3")
        for module in x:
            if module[2].lower() in tracker.latest_message.get('text').lower():
                mes += module[0] + " " + module[1]
        return [SlotSet("leading_lecturer", mes)]
```

Example of story:

```
- story: module
  steps:
  - intent: module
  - action: action_module
  - action: utter_module

- story: terms
  steps:
  - intent: terms
  - action: action_terms
  - action: utter_terms

- story: timetable
  steps:
  - intent: timetable
  - action: action_timetable
  - action: utter_timetable

- story: leading_lecturer
  steps:
  - intent: leading_lecturer
  - action: action_leading_lecturer
  - action: utter_leading_lecturer
```

Example of domain:

```
version: "2.0"

intents:
  - greet
  - goodbye
  - affirm
  - deny
  - mood_great
  - mood_unhappy
  - bot_challenge
  - database
  - module
  - terms
  - timetable
  - leading_lecturer

actions:|
  - action_modules
  - action_module
  - action_terms
  - action_timetable
  - action_leading_lecturer
  - action_greeting

entities:
  - User_id

slots:
  User_id:
    type: text

  Modules:
    type: text

  Module:
    type: text

  Terms:
    type: text

  Timetable:
    type: text

  leading_lecturer:
    type: text

responses:
  utter_greet:
    - text: "Hey! How are you?"

  utter_cheer_up:
    - text: "Here is something to cheer you up:"
```


Domain

The domain is where everything that works within the chatbot assistant is defined. Slots, intents, entities, actions and responses are all stored within the domain file. To discuss later the actions, the intents and the stories I would explain in further details the slots and the entities, so it could all come down.

Slots

The slots are the bot's memory. They do act as a key-value store (just as a dictionary in python) – where information about the user (e.g. their ID on the Django platform) could be stored and data gathered from the outside world (e.g. Query sets). The slots are defined in the domain file by specifying a type of the slot, which can vary from text, boolean, float, etc.

Entities

The entities are the section list that could be extracted from the NLU pipeline. Entities are structured data inside the user message. Training data should be specified in order for the entities to work.

Intents

The intents are the section where with the use of entities example sentences are derived and the sentences are used for further NLU training process. Intents could include entities as long as they are put in between “{}” brackets. When the user interacts with the chatbot the NLU pipeline will calculate the confidence with which specified intent is selected. When specific intent is selected, the storyline will be executed and actions that follow in the sequence will be executed returning data to the responses and then the responses would be delivered back to the user.

Stories

A story is a representation of an example sequence in the conversation with the user. It could include responses, intents, custom and inbuilt actions. The stories are also used to train the NLU pipeline so the bot knows the sequence when calculating the confidence of every chosen intent.

Actions

The actions of the RASA run on a separate server, where they are being executed and only a slot set is returned to the bot, with which the bot would work. Having said that, when RASA calls upon a custom-built action it calls the server to execute the specified action and the data returned is the responsibility of the action's server. The actions server has an inbuilt debugger, which makes debugging the actions quite easy.

Returning the timetable of the student

Personal student assistant

Hello, what could I help you with,Andon?

less than a minute

my timetable

less than a minute

Timetable info: Module: CM3102 Lecture of: Basics of the HTTP starting at: 2020-04-20 14:22:00 ending at: 2020-04-20 17:22:00

less than a minute

SUBMIT

Returning the module leader of a specified module

Personal student assistant

Hello, what could I help you with,Andon?
1 minute

my timetable
1 minute

Timetable info: Module: CM3102 Lecture of: Basics of the HTTP starting at: 2020-04-20 14:22:00 ending at: 2020-04-20 17:22:00
less than a minute

module leader of Web Applications
less than a minute

Module leader: Steven Hristov
less than a minute

Returning terminology information

Personal student assistant

my timetable

9 minutes

Timetable info: Module: CM3102 Lecture of: Basics of the HTTP starting at: 2020-04-20 14:22:00 ending at: 2020-04-20 17:22:00

9 minutes

module leader of Web Applications

9 minutes

Module leader: Steven Hristov

9 minutes

terms info of Web Applications

less than a minute

Terms info: Object-relational mapper - More info on: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping
Hypertext Transfer Protocol - More info on: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

less than a minute

SUBMIT

Returning module description

Personal student assistant

Hello, what could I help you with, Andon?

1 minute

my modules

1 minute

Modules: Computational thinking, Web Applications, Enterprise architecture,

1 minute

module description of Web Applications

less than a minute

Module info: The studies of Web Applications

less than a minute

SUBMIT

Enrolled modules of the student

Personal student assistant

Hello, what could I help you with, Andon?

less than a minute

my modules

less than a minute

Modules: Computational thinking, Web Applications, Enterprise architecture,

less than a minute

SUBMIT

Test cases

Function requirements test cases

Test case No: 1	Functional requirement No. 1: The system would provide the user its own secure space, which could only be personally accessed. (Student interface) – Type: Must have
Test description	Acceptance criteria: <ul style="list-style-type: none"> o User can log in with a username and a password to the system. o Method of signing up would be provided to the users. Test results: The student can both sign up/register and can log in with a username and a password PASS

Test case No: 2	Functional requirement No. 2: The system would allow the user to watch past recordings. (Student interface) – Type: Must have
Test description	Acceptance criteria: <ul style="list-style-type: none"> o After entering the past-recordings interface the user should be able to see all past recordings from the module o Video player should be generated for every lecture/past-recording o Only lectures from the course in the university of the specific user(student) should be allowed to be seen Test result: The student can see past recordings that are module-specific. Video player is generated for every lecture. Only lectures from the course in the university of the specified student are to be seen. PASS

Test case No: 3	Functional requirement No. 3: The system would allow the user to watch a live-streamed lecture. (Student interface) – Type: Must have
Test description	Acceptance criteria: <ul style="list-style-type: none"> o After entering the live-streaming interface the user should be able to choose in between live-streamed lectures in the current moment o Video player should allow the user to see the content o Only Livestream lectures specific to the user's course should be seen Test result: The user (student) can see and choose in between live-streaming lectures. Only students part of the course and the module can see the live lectures. Media player could be opened for all the live-streaming lectures. PASS

Test case No: 4	Functional requirement No. 4: The system would allow the user to interact with the chatbot (Student interface) – Type: Must have
Test description	Acceptance Criteria:

	<ul style="list-style-type: none"> o The chatbot, as well as the platform, is considered as a proof-of-concept and the chatbot would be limited to five use-cases. o The chatbot should be able to retrieve the timetable that is student-specific o The chatbot should be able to give more information on different terms that are discussed during the lectures <p>Test result: The chatbot is indeed limited to five use-cases and could retrieve information for both the terminology in a specified module and the timetable of the student</p> <p>PASS</p>
--	---

Test case No: 5	Functional requirement No. 5: The system should allow the user to book a chatroom with the intent of meeting a specific tutor (Student interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o Booking a chatroom to meet with a specific lecturer would require a timeframe of initialisation and date on the initialisation of the chatroom. Also, it would require the specific lecturer that the student would like to meet. If the meeting is approved by the qualified personnel to do so (Lecturers), a chatroom would be created. <p>Test result: The chatroom meeting could be booked exactly with the specified data and upon approval the chatroom meeting in inaccessible</p> <p>PASS</p>

Test case No: 6	Functional requirement No. 6: The system should allow the user to participate in discussion board discussions (Student interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The student would be able to join in the already approved chatroom. o The student would be able to discuss and chat with the lecturer. o The chatroom itself should be unique and discussion between lecturer and student should remain confidential as that might cause privacy issues if violated. <p>Test result: The chatroom meeting is unique and enables the user to discuss whatever found suitable. Both of the users can join to a chatroom meeting if the meeting is approved</p> <p>PASS</p>

Test case No: 7	Functional requirement No. 7: The system should allow the user to participate in discussion board discussions (Both Lecturer and Student interface) – Type: Must have
Test description	Acceptance Criteria:

	<ul style="list-style-type: none"> o The student would be able to discuss freely on the discussion board that would be specific to the module. <p>Test result: The user is indeed able to discuss freely on the discussion board.</p> <p>PASS</p>
--	---

Test case No: 8	Functional requirement No. 8: The system should allow the user to take a self-assessment specific to the module taken (Student interface) – Type: Should have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The student would be able to have a module-specific self-assessment that would generate random questions in the context of the module. <p>Test result: The student is allowed to have a self-assessment that is module-specific.</p> <p>PASS</p>

Test case No: 9	Functional requirement No. 9: The system would provide the user its own secure space, which could only be personally accessed. (Lecturer interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o User can log in with a username and a password to the system. o Lecturers can only log in with admin-regulated credentials. They should only be given an account to work with, instead of registering as lecturers as that might cause potential authentication issues. <p>Test result: The lecturer can log in to the system by filling in his/her credentials. Lecturers cannot register to the system, only the administrator could give them profiles for the platform.</p> <p>PASS</p>

Test case No: 10	Functional requirement No. 10: The system should allow the user to upload past recordings both automatically and manually (Lecturer interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user/lecturer would be able to upload past recordings both manually – where the lecturer would attach the lecture and upload it and automatically – where the lecture would be taken straight from the folder and uploaded into the database. <p>Test result: I managed to create an algorithm, which did allow the lecturer to upload automatically past recordings. When it comes down to manually uploading past recording, the lecturer is allowed to do so.</p> <p>PASS</p>

Test case No: 11	Functional requirement No. 11: The system should allow the user to upload a live streamable lecture for specific module (Lecturer interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user would be able to upload live streamable lecture with fields being required such as caption of the lecture, URL of the Livestream, starting time and ending time <p>Test result:</p> <p>By filling the fields, the lecturer can upload a module-specific live-streaming lecture.</p> <p>PASS</p>

Test case No: 12	Functional requirement No. 12: The system should allow the user to approve a chatroom meeting with a student (Lecturer interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user would be able to approve a chatroom meeting with a student. After approval, the meeting room will be created. <p>Test result:</p> <p>The lecturer is permitted to approve a chatroom meeting with a student. Upon approval, the chatroom meeting would be created.</p> <p>PASS</p>

Test case No: 13	Functional requirement No. 13: The system should allow the user to participate in discussion board discussions (Lecturer interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user would be able to participate in discussion board discussions freely. <p>Test result:</p> <p>The user can participate in all the module-specific discussions on the respective discussion boards.</p> <p>PASS</p>

Test case No: 14	Functional requirement No. 14: The system should allow the user to join in a chatroom meeting (Lecturer interface) – Type: Must have
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user would be able to join in a chatroom meeting with a student and write freely messages that are chatroom specific. <p>Test result:</p> <p>Upon approval, a chatroom meeting is initialised and the lecturer can freely join and interact within its' scope.</p> <p>PASS</p>

Non-functional requirements test cases

Test case No: 18	<p>Non - functional requirement No. 1: The system would store entries such as:</p> <ul style="list-style-type: none"> • Data needed to store the live streamable lecture and portray it back to the students. • Data needed to store the past recordings • Data needed to store the self-assessments specific to the modules • Data needed to store the chatroom meetings • Data to store the discussion board specific to the module <p>– Type: Must have</p>
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o Once a new entry is submitted it should be stored instantaneous and be portrayed back to the user. <p>Test result:</p> <p>Data is indeed stored instantaneous onto the database, however, I could not manage to make the lecturer create a self-assessment test for the students, resulting in only the admin adding self-assessment tests.</p> <p>PARTIAL FAIL</p>

Test case No: 19	<p>Non - functional requirement No. 2: The system will be intuitive and easy to use. (Student and Lecturer interface) – Type: Must have</p>
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The User interface should be appropriate and consistent to use. <p>Test result:</p> <p>Whilst creating the MVP – I had an image of how the interfaces might look like and I tried my best into implementing them in such a way that they allow easy and consistent usage.</p> <p>PASS</p>

Test case No: 20	<p>Non - functional requirement No. 3: The user will have accessibility over the past lectures (Student interface) – Type: Must have</p>
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user will be able to see all the past recordings <p>Test result:</p> <p>The student can see all the past recordings that are part of a module he/she is enrolled on.</p> <p>PASS</p>

Test case No: 21	<p>Non - functional requirement No. 4: The user will have accessibility over the live-streamed lectures (Student interface) – Type: Must have</p>
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user should be able to see the live-streamed lectures that are part of the modules in his/her course <p>Test result:</p>

	<p>The student is allowed to see all the live-streaming lectures that are part of the modules he/she is enrolled on.</p> <p>PASS</p>
--	---

Test case No: 22	<p>Non - functional requirement No. 5: The user will have the given permissions to upload past recordings and live-streamable lectures. (Lecturer interface) – Type: Must have</p>
Test description	<p>Acceptance Criteria:</p> <ul style="list-style-type: none"> o The user should be able to upload both past recordings and live-streamable lectures to the platform. <p>Test result:</p> <p>The lecturer is indeed allowed to upload both past recordings and live streaming lectures as long as he/she is part of the module.</p> <p>PASS</p>

Comparing my solution to already existing ones

It is quite hard and unrealistic to compare my platform to already existing ones (as learning central). With the help of my supervisor, I was able to produce a product, however, it cannot be properly compared to platforms developed by teams of developers over a long period, because the project's goal was a proof-of-concept platform.

On the contrary, I implemented interesting features to my platform that I am unsure if many other platforms have. From an automatic uploader of past recordings to the chatbot, I tried to vision a student portal/platform and make it as usable as possible for the end-users (both lecturers and students). Having said that, there certainly are features that I find quite interactive on the platform I created.

On the other hand, my platform is nowhere near as user-friendly as learning central is. I could safely assume that from the developers' point of view it is a much more complex system. I am unsure what technologies are used in the process of development of learning central, in particular, but I could say it has features such as the assessments for the modules, which I do envision implementing at some point on my platform.

I do believe that my platform is not even close to the level of learning central, but there are features on my platform that I find very interactive and innovative for such a student portal/platform.

Conclusion and Future work

The project intended for the creation of a proof-of-concept web platform – student portal. The usage of the web platform is mainly for students and lecturers. The project aimed to create such a web platform that would prove the concept and that the solution could be implemented by many universities that would like better student satisfaction. The project aimed to provide the following features:

- Having two separate user interfaces (UI) for lecturers and students
- The lecturers should be able to live-stream lectures
- The module-specific live-streamed lectures should be visible to, only the enrolled to the course the module is in, students
- The lecturer should be able to both automatically and manually upload past recordings
- The past recordings should be available to the students that are enrolled to a course that the module is in
- There should be discussion board specific to every module
- There should be chatrooms that could be booked by students to meet with the specific lecturer if needed. These chatrooms should only be accessible by the student if the specific lecturer has approved of such meeting. If approved, unique chatroom should be initiated
- There should be chatbot that would interactively answer questions of the students (limited to 5 use-cases)
- There should be dynamically generated module-specific self-assessments for the students

All the main functionalities of the web platform were set out to be developed and implemented during the course of the project. When it comes down to the personas and their potential goals, they were achieved as well.

However, there were some limitations. Co-vid 19 disrupted the project quite a lot since I was not able to get as many resources and usability testing of the platform. Furthermore, I could not manage to implement more than five use-cases for the chatbot. Even though I was planning for the same five use cases, I was thinking I could implement further use cases. Also, there were features that I visioned to be implemented on the platform, however with the time restrictions I did not physically have the time to start and finish developing.

Overall, I am satisfied with the outcome of the project since all the initial requirements were met. However, many things could be changed for a better platform in terms of both usability and functionality. There is potential in the project and I might keep on developing the platform, even after the project finishes. I was a bit stressed that the time might not be enough for me to finish the platform, which resulted in small cosmetic (front-end) issues, that I could've avoided for better usability.

I have several future work assignments on the project, which are the following:

- Create a lecturer registration page, that could only register the lecturer if the admin has approved the profile
- Broaden the range of functionalities the chatbot has
- Having the platform be an easier to navigate

- To make a database that would be able to fit large portions of data and be better for the production process – Oracle
- Allow the lecturer to watch their recorded past recordings and live lectures
- Better security measures
- Allow students to upload different assignments on the platform
- Allow the lecturers to assess the assignments uploaded by the student

All these future work assignments would have shaped the platform into a better product. One of the things I could not manage to develop, which could have been crucial for the platform, is that lecturers should be able to watch their recorded and uploaded lectures as well as their live lectures (not just on the YouTube tab opened on the browser).

Innovation

The project itself required me to be innovative. I came up with the idea of the lecturer uploading past recordings automatically. I designed the algorithm and implemented it as a solution. The algorithm as a solution was not specifically hard, but it came down as one of the most innovative features of the platform. Furthermore, I decided on implementing a chatbot based on a machine learning algorithm, which is one of the “hottest” topics in the technology world being used in self-driving cars, google translate, fraud detection, etc. ^[12]

Also, I decided on implementing a self-assessment feature, where the students are allowed to self-examine their skills acquired at a specific module. The questions are being stored in the database with their respective answers. When the user enters the self-assessment page, the questions are getting randomised to protect against fraud.

Besides that, I decided to implement chatroom meetings between lecturers and students as a feature. I find that very interactive as many students might find themselves in hard times, where they would have priority over the privacy of the information they are sharing. Every chatroom is unique and every message is stored on the database.

Reflection on the learning process

The entire project was both an exciting and tremendous part of my experience. It has improved significantly my organisational and time-management skills, which are crucial for my development in the future. Furthermore, I am glad that I had the motivation to work on the same project for nearly a third of a year. The dedication shown by me during the development process is crucial to me continuing the project.

I significantly improved and enhanced my existing Python coding skills. The project allowed me to use techniques and skills learnt over the years studied. Creating a web platform was something new for me and I personally loved it. I started the project with little to no previous web development experience. The project not only motivated me to implement solutions that I had to research upon but helped me build a reasonable skill set when it comes down to web development with Django.

I faced many issues along the way of implementing from personal ones to project-based. I had to learn server-side connections to connect the chatbot to the Django platform. One of the toughest challenges I had, was to work part-time and yet deliver a product. As I said early, the project was a major challenge for me, which I am relieved and happy to finish.

Regarding the organisation of the project, I had to change my initial plans completely. In the early stage of the development process, I was thinking that the main priority would be security, but as a proof-of-concept platform, Django's encryption policies were enough to prove the concept. Other than that, I believe I kept the standard high by meeting regularly with my supervisor – Dr Daniela Tsaneva and showcasing the platform and the improvements I had made every week.

However, if it comes down to me taking the module again, I would have carried out more research. I would have spoken to potential users and make a questionnaire to get appropriate feedback from potential end-users. Besides, in the beginning, I started quite strong by developing the main functionality of the platform quite early. Instead of motivating me, that demotivated me and there were weeks where I did not perform at my best for certain. Specifically, there were weeks when things went wrong, and the positive thing was that I was constantly using version control on git. That proved to me once again how important version control is and that is an irreplaceable part of any project.

Overall, I am grateful to be given the opportunity to develop the platform. I was a bit anxious at the beginning as I was working on a self-proposed project. That meant that I had to build something unique and useful at the same time, which was quite fulfilling in the end. Last, but not least I am proud of what the project turned out to be. Those challenges, all helped me to progress as a person and proved to me that I can find a solution to all the problems - questions if I am determined enough to seek the appropriate answer.

References

1. OBS web platform - <https://obsproject.com/>
2. G2 Crowd - <https://www.g2.com/categories/live-stream>
3. Rasa API framework documentation - <https://rasa.com/docs/rasa/>
4. Django REST - <https://www.django-rest-framework.org/>
5. Django REST - <https://www.djangoproject.com/>
6. Django 1.10 Tutorial - <https://overiq.com/django-1-10/mvc-pattern-and-django/>
7. MVC Tutorial for Beginners: What is, Architecture & Example - <https://www.guru99.com/mvc-tutorial.html>
8. Katie McLaughlin - 24 Nov 2017 - An introduction to the Django ORM - <https://opensource.com/article/17/11/django-orm>
9. Jakob Nielsen- Apr. 24, 1994; Updated Nov. 15, 2020 - 10 Usability Heuristics for User Interface Design -<https://www.nngroup.com/articles/ten-usability-heuristics/>
10. Alexander Stepanov - Best practices working with Django models in Python - <https://steelkiwi.com/blog/best-practices-working-django-models-python/>
11. Rasa documentation - <https://rasa.com/docs/>
12. Innovative Applications of Automated Machine Learning - Mar 16, 2020 - AIT Staff Writer-<https://authority.com/ait-featured-posts/5-innovative-applications-of-automated-machine-learning/>