FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Mikko Koivula
Joonas Sutinen

IMPROVING CONTENT AUTHORING USER EXPERIENCE IN THE LOVELACE LEARNING ENVIRONMENT

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
June 2021

Koivula M., Sutinen J. (2021) Improving Content Authoring User Experience in The Lovelace Learning Environment. University of Oulu, Degree Programme in Computer Science and Engineering, 44 p.

ABSTRACT

This thesis provides the analysis, planning, execution and evaluation of a new back- and frontend prototype of the online learning environment Lovelace created by the project group. The pre-requisite for the prototype was to utilize the same technologies as the current live version, and the project groups first major task was to investigate, comprehend and execute them. In addition, the project group was advised not to review the code of the live version to ensure fresh perspective into execution of the new version.

The design takes influence from other sources such as Moodle. This thesis covers the process of the design from first sketches and analysis of the set-out requirements. These requirements include extracting the current editing functionality from separate administrator page to the easily accessible lecture pages editing widget, the static website contents caching, support for the existing Lovelace markup text and many others. The implementation phase starts by following the plan created in the design part which made the process more streamlined. Technical aspects of the development are handled in the implementation part of the thesis. Polymorphism, the way the content is rendered to the viewer, explanation and representation of how content forms and caching works are explored here.

The evaluation of the finished prototype was executed in form of measurement of websites load times with addition of an expert evaluation meeting with experienced user of the live version of Lovelace. The meeting consisted of different test cases which the attendee had to complete on both old and new versions. These tasks were all timed and the results were vastly better with the project groups prototype. All of the tasks completed with less time with the prototype, and in some cases even twice as fast. Comparing the end result with the pre-requisites, the requirements were met well, and the improvements were proven to be a success.

Keywords: B.Sc. degree, online learning environment, project development, Lovelace, expert evaluation thesis

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

UML	unified modeling language
UX	user experience
UI	user interface
SQL	Structured Query Language
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
ORM	Object-Relational Mapping
PDF	Portable Document Format
WDF	Web Development Framework
CMS	Content Management System
ICT	Information and Communication Technology
CSRF	Cross-site request forgery
XSS	Cross-site scripting
REST	Representational State Transfer

1. INTRODUCTION

Web-based learning environments are a new concept when looking at the whole history of education. Moving from assignments given and executed on paper to having the whole process completed on the web releases resources from schools and allows them to be placed elsewhere. The project focuses on the learning environment Lovelace which has been designed and created by Miikka Salminen.

The project group has experience of the website from the perspective of the student. The website has great potential to be the main tool for teaching programming to students.

To make the learning experience of students more engaging, the lectures are usually interactive. The length of time the students can concentrate on the lecture is the most important measurement in evaluating the experience. Within every exercise there is live feedback on how the written code worked and if there were any problems in, for example, executing the code. This feedback comprises of manually written hints written by the teacher and an automatic code reviewer. This review also includes automatic analysis by using lint software which ensures that the code is consistent and resolves basic errors in the code. This allows the student to quickly have some idea on how to fix the problems they come across.

The Lovelace front page welcomes the user with the whole catalogue of courses. Within each course there are different previous variations (e.g., 2020 Autumn and 2021 Spring instances). On the left side of the page thee user can find all the terms used inside the specific courses. In programming for example, there is great number of terms to learn. When the student finds all the terms quickly from the side panel, the effectivity of learning is greatly increased. The course instances have a table of contents at the top, from which the user can find the specific content they are after. All of the headers in the table are links to the sections to enable quick access.

The teachers point of view differs greatly from the normal user. The administrating user or teacher has to access a another admin interface to create, edit or delete content on the pages. If, for example, the author wants to change the photo inside the exercise, the admin page of the used web development framework Django must be accessed and the whole page must be edited instead of just the module in question due to the whole page being a single text file. This solution is very cumbersome and requires many unnecessary steps, which ultimately slows down the effective usage of the website.

The main task for the project is to have a new vision for how to edit the content that is displayed for students on Lovelace. The goal is to be independent from Django's admin page and have the content be modular instead of being dependent on the entire page and its content. Ideally the new design should speed up content creation and editing on the webpage while making it more intuitive and easier to use.

2. RELATED WORK AND MOTIVATION

The online learning environment Lovelace has been used by students and teachers to move the course related exercises online. The course instances are divided into parent pages, which include the lectures, interactive tasks and downloadable files and images. If the user want to edit the content on the website, currently they need to access a separate admin page. Constantly accessing the admin page is inefficient and tiresome in the long run, which makes the Django administrator page poor for this kind of content management. One solution to this issue would be the creation of convenient embedded editing tools.

The project is about reworking the management and database handling of the website Lovelace from a scratch. In order to make the new version, the utilization of tools Django, PostgreSQL, Redis and existing CSS layouts is required. Information about these technologies is found in the following sections.

2.1. The Current Lovelace Version

2.1.1. Lovelace Content Model

The website has a well-defined hierarchy. A singular course instance contains links to the media used in the content, links to the desired content and the terms used. Content in the current live version consist of lectures, text field exercises, exercises that require files to be up- and downloaded and routine exercises. At the root there is the course which contains the required sub objects. Every time the website gets updated the site saves the snapshot of the previous version and gives it an id. This can be found from a catalogue of all the changes made since the creation of the original piece. The user signs in with their university account and depending on the user, they can be either a student or a teacher. The teacher can create, edit, and delete content inside the courses.

The current version of Lovelace has many parallel course instances of the same lecture. If these course instances existed as separate copies of the page it would make the management of the content very difficult. To help with easier management of content each course instance only contains references that can be used to determine which version of the content is displayed.

2.1.2. Archiving

All changes from a certain version exist under the same revision number, which in the case of Lovelace can be referenced by a date. This makes it possible to easily change what version of the content is displayed simply by changing which date the content on the page is supposed to be from.

The archive of all the content is incredible feature the live version of Lovelace offers. All the changes to the courses and lectures are stored as snapshots for later exploration. This is done by utilizing Djangos reversion extension, which provides version control for model instances. If the administrator causes some unwanted changes and problems, django-reversion is able to be used for rollbacks and recovery actions.

2.1.3. Lovelace Markup and Rendering

The markup in the current Lovelace is used to enable adding multiple different types of content in one document. With markup it's possible to add images and embedded content into the same text field as all the paragraphs, headers and other text content. This makes it easy to add a lot of content quickly by opening a single document in a text editor and then dropping all the text and links to embedded content into the same document. A downside is that you need to implement validation to ensure that the document doesn't have links to content that doesn't exist in the database, and also all of the embedded context links need to be kept up-to-date.

When a page is rendered it's done in two passes. The first pass renders all information that's not user specific. The static content such as text and images are structured as just a string of HTML that gets cut every time there's embedded content or calendars. The static content inside calendars and embedded content are also added in the first pass using dictionaries with keys for different pieces of content inside the embedded content. The point of adding the static content first is to make it possible to cache the result of this first pass. since the content in the first pass won't change unless the content is edited. In the second pass the list is looped through again and information specific to the user is rendered where appropriate and puts a pre-rendered HTML there using the embed-frame template and a template tag which compiles the context information.

Problem with the current database solution is that the content page itself is a whole table. There are no different tables for the previously presented exercises. This is due to the live versions model of using proxy models [1]. A proxy model is a subclass of the database-table. The model does not get a separate table to work with, so the queries, deleting, creating, and editing all effect the original database model. This is mostly to save time in the development, as there is no need to create different tables for each type of content for example content containing images. Creating the different tables make the aggregation of all the different types much more cumbersome if the ORM is used.

2.2. Web Development Framework

Web development is becoming increasingly complicated at a time when time constraints are becoming tighter. This gives the developers less time to focus on their specific tasks. To combat this phenomenon, web development frameworks became a staple asset across the world. A web development framework is a set of resources and tools for software developers to build and manage web applications, web services and websites. Basically, a web framework is a full library of code that makes the development of software both faster and less cumbersome. For example, web framework can provide the developer with methods to handle URL routing for their application. When a user requests a specific URL, the methods handle the routing and renders the correct view based on the rules set out for the URL Routing. To prevent writing the same code repeatedly, the website does it on behalf of the developer.

Such a framework includes templating capabilities for presenting information within a browser, the programming environment for scripting the flow of information and the

application programming interfaces (APIs) for accessing underlying data resources. The framework also provides the foundations and system-level services for software developers to build a content management system (CMS) for managing digital information on the Web. Developers can use the framework to define the 'out-of-the-box' content management capabilities, user authentication features, and administrative tools.

There is great amount of useful web development frameworks [2] freely available online, but they all come with their own pros and cons. Example of a more simplified framework is Flask [3]. Its structure if quite light-weight and thus is also regularly referred to as a micro-framework. High extensibility offers the developer an unlimited number of plugins and functionalities to be added to the application at hand. This ability makes the framework very versatile as the developers can keep control of the applications core very easily. Flask also comes with its own, although somewhat limited, effective API. It contains tons of pre-defined functions and with these the developer can for example register custom template functions or register a rule for routing incoming requests.

The more relevant web development framework in the scope of the project, is Django [4]. Just like Flask, Django takes care of many basic functionalities to develop both maintainable and secure websites. Third party plugins are not required as Django comes packed with most of the things normal developers require. Great amount of documentation and information can be found across the internet. Django is a lot simpler to scale at any level because of its component-based framework which means that each layer is independent of each other. Both Flask and Django are written in programming language Python.

Web frameworks make it easier and a lot faster for back-end and front-end developers to reach the finish line, which is great for business and great for ensuring that programmers do not spend resources on having to write the same code repeatedly. [4] In our project, we will use Django, a popular python web development framework, to implement the system.

2.2.1. Common Features

There are some characteristics in web development frameworks that set them apart from one another. Some frameworks come with frameworks for authorization which ensure the security of the website. By using the tools the author can choose the parameters from which the framework knows how to handle restrictions in different circumstances. Django provides great security [5] not only in the scope of authorization, but also against SQL injection attacks, CSRF [6], XSS [7], clickjacking and many other types of malicious acts. Some frameworks provide the developer with ROA or resource-oriented infrastructure to give the ability to develop software using REST interfaces. These interfaces are components or data structures which can be used in various conditions. Multiple frameworks have the support for this infrastructure and Django is one of them. There are many ways web development frameworks handle mapping of URL. Django does this by matching the requested URL with the url.py file. The "urlpatterns" tuple contains the mapping between URLs and the actual views. AJAX is short for Asynchronous JavaScript and XML, which is a technique sometimes

used in web development. It allows the applications to be both faster and dynamic. Django includes this feature as well, making it extremely versatile framework.

2.3. ORM (Object-Relational Mapping)

When a developer works with back-end, using SQL can become very cumbersome as it differs greatly from the programming language used in the main software. To combat this, object-relational mappers enable the developer to use their preferred programming language instead of SQL. This is done by using dedicated libraries.

Relational database is a a database that stores data in a way that enables related data to be accessed from a data point. A relational database contains rows, which are unique records with their own ID, which is called the key. The columns in a relational database contain different attributes, with each record typically having a defined value for each different attribute. These attributes can then be accessed from the ID[8].

ORM or Object-Relational Mapping is a technique for storing, retrieving, updating and deleting from an object-oriented program in a relational database. Object-relational mapper is typically a code library which allows automated transformation of data from database into an object that's usable in programming of applications. This allows the programmer to work with the data in the database using programming languages with a high level of abstraction such as python instead of having to use relational database specific languages like SQL[9]. An Object-relational mapper basically functions as a translator between relational databases and high-level abstraction programming languages that are object-oriented.

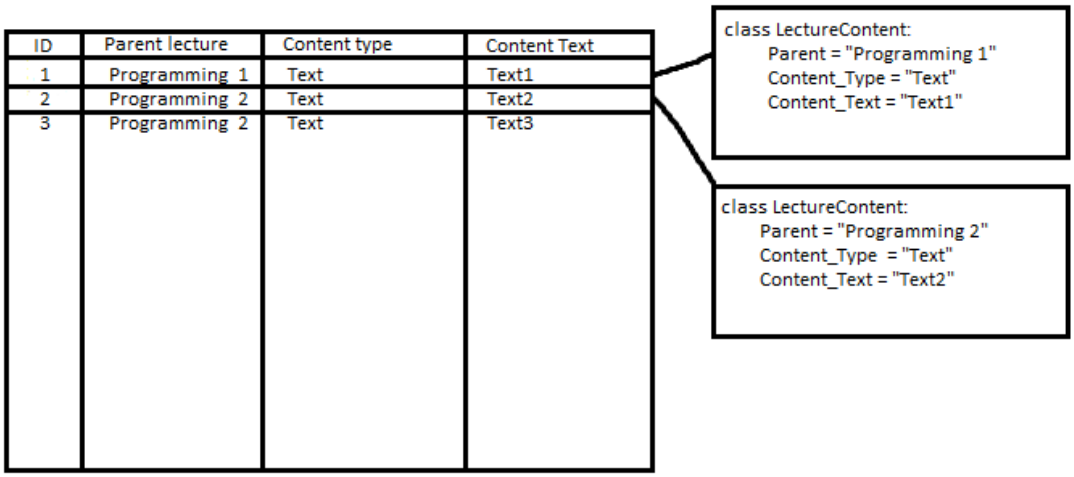

Figure 1. Example of how object-relational mapping would work in our project

The benefit of Object-relational mapping is the fact that avoiding the usage of languages like SQL can make the project easier to work on. Generally speaking it's easier to work with one programming language than it is to switch between different programming languages like python and SQL for example. Overall using an ORM generally results in quicker development, since you don't have to write declarative paradigm SQL statements.[9].

Disadvantage of using Object-relational mapping is that it can be slower than using SQL directly, since the high level programming language statements need to be translated to SQL there's typically some performance loss[9]. In larger databases it is often better to use raw SQL to make sure the queries are performed quickly and efficiently. Using an ORM can also disallow the programmer from learning SQL, which can be a hindrance on the development of the overall skill level of a programmer.

The tool which manages the flow of information between the database and the object-oriented program is called a data layer. The programmer never sees beyond the data layer. Data can be edited by selecting a row in the database and utilizing data layer. The Lovelace v2 project will utilize Django's [10] ORM -feature, which allows the writing of Python code instead of SQL to read, create, update and delete data in the database.

Django officially supports many databases like MariaDB, MySQL and Oracle, but the project is based on PostgreSQL. It is a open source and free object-relational database system which is highly extensible.

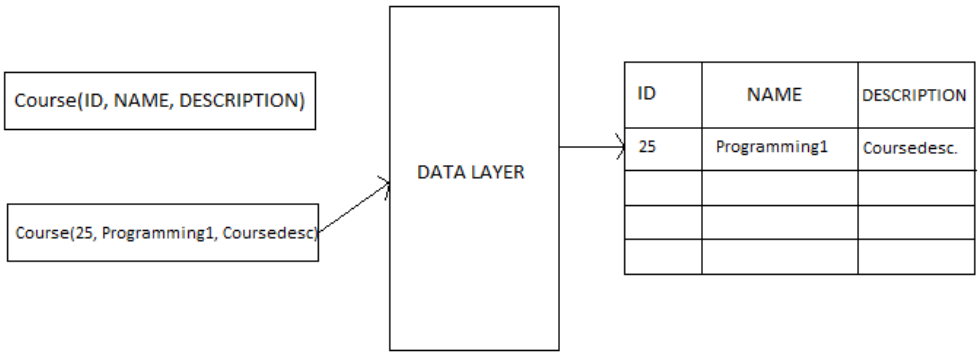

Figure 2. Example of saving and inserting an object.

2.4. Polymorphism in Software

The term polymorphism itself has multiple usages depending on the context but the main idea behind it always stays the same. In nature, polymorphism appears everywhere one might explore. When observing a creature like a tiger, which has the DNA to have a chance to have different kinds of patterns on its skin, the tiger is a polymorphic species. This same logic can be used in the world of programming and databases. Just like when the real life "function" creates a tiger, the parameters it takes define the exact version of the object. A British ecological geneticist E.B Ford in his text "Polymorphism" [11] explains that polymorphism is the occurence together in the same habitat of two or more distinct forms of a species in such proportions that the rarest of them cannot be maintained by recurrent mutation.

Polymorphism is a crucial concept when working with object-oriented programming. This allows for example the usage of a singular function and returning of multiple data types based on the parameters or the objects inserted into

the function. One example of these kinds of functions is the already existing python function len( ) which, depending on the input, gives the desired return. Inserting a string "ABC" into the function returns the number of characters in the presented string, which in this case is the integer 3. However, if the function receives a list of strings as a parameter, the amount of items inside the provided list is returned. In case of len(["A", "B", "C", "D", "E"]) the function returns value 5. The function has also the capability of reading a dictionary and returning of the number of keys.

The topic extends beyond just the basic functions. When dealing with classes which are very common among the users of python, the concept of polymorphism can be used to have functions with identical names on multiple different classes. Following example shows the possibilities of polymorphism in methods in classes.

There are two classes, class Course and class Content. Each of these classes contain two methods: a method information(self), which prints information about the specific class and a constructor which takes two parameters, a name and an index. Next the new objects are created.

- course1 = Course("course1", 1)
- content1 = Content("exercise1", 1)

Now according to polymorphism, looping through the list of objects should be possible and even when the name of the function information() is the same in both of the classes, objects know what is the current context.

- for thing in (course1, content1):
- thing.information()

This is possible thanks to polymorphism. Even though the example is fairly simple the different adaptations of this in software and in the projects case web-development, is incredibly useful. In databases polymorphism means that a single column can contain data of different types. For example, in a JavaScript Object Notation file a field contains data regardless of the type. There can be a field that contains an integer in one document, but it can also contain a string in another document. While Django is the main framework the new version of Lovelace is created on top of, the main language python allows the polymorphism to be implemented fluently.

2.5. Caching

Caching is a way of speeding up an application by reducing latency and network traffic. Caching works by saving results of common database queries in-memory so they can be served to the client without a need to perform database queries. Serving the data from cache instead of querying the data from database is a lot faster, which is why it can help increase the performance of an application. Everything happening on the server will increase the page load time, which in turn will make the website less appealing and overall slower to use. The caching can be handled on the client side, or on the server side. In web applications the browser typically creates a folder where it stores

the data on the client's computer if the caching is client side, and if it's server side then the data is stored on the server.

Caching is used between the client and the server. When the client requests a page for the first time from the server, it is saved into the cache. The next time the page is requested by the client, the data can be retrieved from the cache instead of querying the database for the data again. This makes the page load faster, but might be problematic if the page is supposed to change often. To allow the data on the page to change the programmer needs to give the cached data an appropriate time to live (TTL) which is the time that the data will be cached until it's deleted, and thus the page will have to be queried from the database on the next refresh. Generally the programmer can also manually clear the cache in the code in parts where the rendered content might change, in order to avoid storing out of date content in the cache. In the current existing implementation the cache lasts forever and only gets cleared if some changes are made to the content. This will be important to implement correctly in our project, since our editing tools would ideally be used to make a lot of changes to the content on the website, meaning the caching needs to work properly to make sure that the displayed content is up to date.

In the project we are using Redis (Remote Dictionary Server) as the caching engine. Redis is an open source in-memory data structure store[12]. The data that's commonly requested from a database can be saved in-memory on the server, meaning read and write operations can be done much quicker, and unnecessary database queries can be avoided. In-memory means that the data is stored in the main memory (RAM) of the server, instead of disks or ssds, which are slower and require a round-trip to disk when doing most operations. than This results in a faster page load time for the user. The difference in loading speeds can be very significant especially if the page that is loaded does lots of database queries normally. Redis can also be used to reduce client-side load by handling resource intensive operations on the server. In Lovelace for example. Redis is used to send evaluations for answers to embedded exercises.

Other ways of reducing load times are reducing file size of the HTML documents, optimizing the way javascripts and style sheets are included, compressing images, minimizing the size of javascript files and removing unused css styles from the css file[13]. In the case of Lovelace the reason caching is very useful and easy to implement is that most of the content on it is static, meaning it doesn't need to change, therefore once it's cached the cache doesn't need to be updated unless changes are made to the content. Using the other mentioned methods to reduce load times can also further help us reduce the load time, which is why they are worth taking into consideration when designing the new system.

2.6. Online Learning Environments

The increase of internet usage and the improving communications technologies have made online learning environments a possible way to educate students. An online learning environment can be defined as an interactive learning environment where the content is available online and where automatic feedback is provided to the student's learning activities[14]. They often include methods of communication between students and teachers, methods for the teachers to share learning material with the

students, and methods for the students to complete online assignments and/or return completed assignments as files, among other features.

The online learning environments can have forums and other communication methods which allow the students and professors to communicate with each other from anywhere at any time. These communication methods would allow the students to ask for help and receive quick feedback from the professor. They would also allow the teacher to inform the students quickly if there is some urgent information that they need. This quick communication is a huge advantage that the online learning environments have over normal face to face learning environments.

Learning material can be quickly distributed and accessed through an online learning environment. The learning material can be distributed as files in different formats, such as PDF files, word documents, images or slideshows, which the students can download and access at any time. Alternatively, the online learning environment could also have the learning material as text and images on the website, which would eliminate the need for the students to download the content as files. However, this could be challenging if the website does not have a good content management system.

Students can complete and return assignments online through the online learning environment. At the most basic level the students could upload their completed schoolwork into the online learning environment, which would then allow the teacher to check it and give feedback to the student. Additionally, the online learning environment can have quizzes and other forms of assignments that could be automatically graded, which leads to quicker results and less work for the people who are typically responsible for checking the assignments.

The success and popularity of an online learning environment is correlated to how many desirable features it has[15]. Desirable features include features such as Online exams, multiple language support, ease of installation, tools for content development and management of content installation, Advanced search and header hiding ability, video conferencing[15]. Moodle for example is one of the most popular open source learning environments, and in terms of features it beats most of it's open source competitors[15]. The large number of features enables many different teaching styles for the users, making it more usable for the teachers and students using it. It was also noted, that even though the large number of features increased the complexity of Moodle, it did not make the website harder to use due to the modular nature of the features. It is clear that the users of online learning environments appreciate having a wide range of features on their online learning environment of choice. Therefore, if you were to create an online learning environment, it would be wise to implement these desirable features in order to increase the likelihood that it would become popular.

In contrast to all the advantages that online learning environments provide, there are some downsides that come with them. One of them is the fact that the usage of online learning environments often comes with increased installation and support costs compared to traditional learning environments[15]. Proprietary software requires paid personnel to develop and maintain the software, as well as provide support for the users of the software. These personnel costs can be avoided by developing the online learning environments as open source software, where anyone can openly contribute to the development of the software. Popular online learning environments such as Moodle use open source software for this reason, which has enabled it to be free for

it's users, therefore being a cheap way for schools to use online learning environments as part of their teaching. [15].

2.7. Prevalent Environments

Based on the advantages and demand mentioned in the Section 2.5 , many online learning environments are developed. In [15], the author selected four most prevalent learning management systems including Moodle, ATUTOR, DOKEOS, OLAT from fifty free and open source learning management systems on the website of UNESCO(United Nations Educational, Scientific, and Culture Organization). The four learning management systems are analyzed in detail from multiple dimensions. According to the study, Moodle stands out with many excellent features such as wider options with different access possibilities, modular structure, advanced backup tools, more different type question support exams, multiple language support and multiple communication methods for learning.

The modularity of Moodle strikes as a very efficient way of controlling data inside lectures. The administrator tools allow the teacher to add or remove existing activities. If the teacher wants to add a new lesson activity to a page, they can do it without too large of an effort. This activity can be accessed by turning editing on and pressing "Add an activity or resource" button at the desired location. Accessing this feature provides the teacher great amounts of pre-defined activities ranging from surveys, quizzes, and lessons to games like sudoku. Adding the lesson of text and a title is very simple but if the administrator wants to, they can modify the appearance, availability, flow control, grades, common module settings and so on. The process of creating a lesson activity if visualised in the Figure 3.

On top of being very versatile with pre-existing content, the user can create their own Moodle plugins aswell as long has they have understanding of PHP and HTML coding. The basic structure of a plugin can be of several types. They can be like the before mentioned activity modules, themes, whole formats for courses, repository plugins or filters. There many different types of plugins one can create, but here are couple examples. For the plugin to work flawlessly with Moodle, the files have to be a certain type and in certain directories. Basic principle is that the Backup Folder is the place that define how the program is going to act after a backup or restoring is performed. The Land folder contains all the strings of characters used in the plugin. The logo which is shown next to the module has to be in the Pix folder. The simplicity and modifiable nature of Moodle makes it a great reference point for the following project. Especially the modular sections the lecture pages are comprised of will be a major influencer. In the next section of the thesis we discuss more about the references and design philosophies.

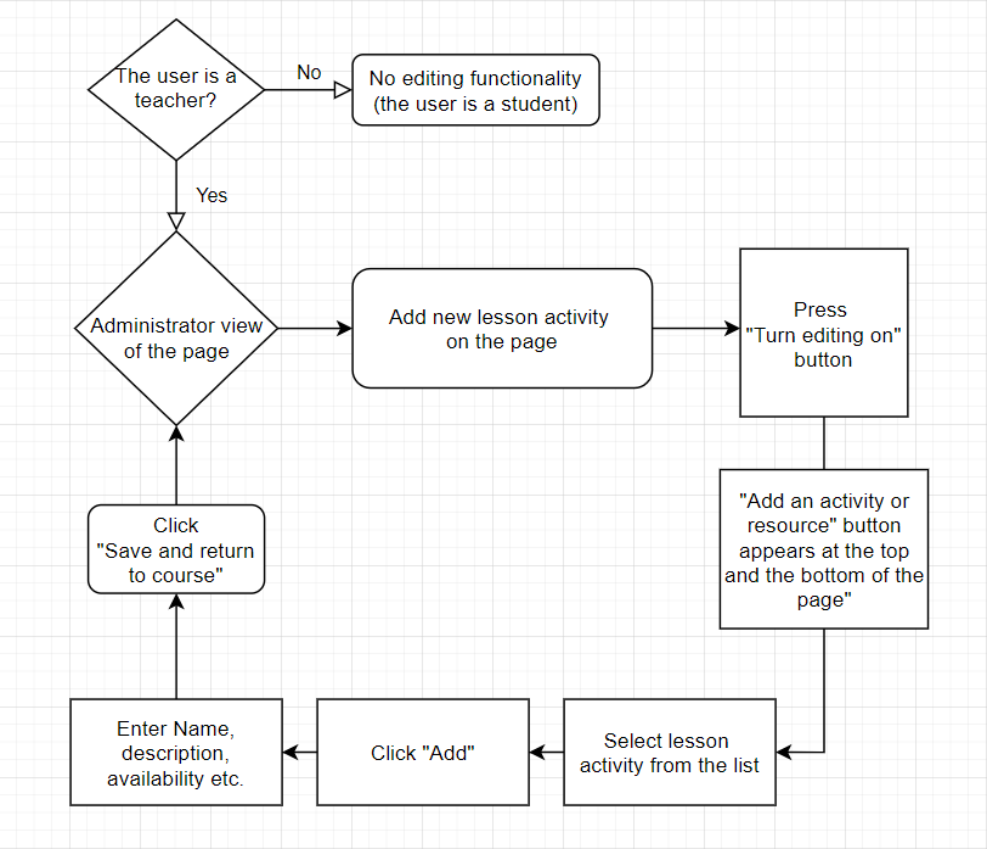

Figure 3. Flowchart of adding an lesson activity on the online learning environment Moodle

3. DESIGN

3.1. Main Goal and the Required Steps

The approach to the design was from the very beginning focused on easy creating, editing and deleting of content inside the desired instance of a course. The teams prior experience with web-development however, was not at the required level to start the work on projects main objectives straight away. First vital step was for the team to learn the basics and fundamentals of the Web-development framework Django, which played a major role in the whole body of the project. Introduction to Django also brought up other critical elements that required attention moving forward. Polymorphism in software will be focused more on in the next section.

3.2. Requirements

To guide us in how the editing tools should be designed we were given a few requirements, which we used to build a list of functional requirements for our project to help us outline the goal and purpose of the project. The functional requirements that we set for our project were:

- The system shall allow adding, removing and editing of content on the lecture page.
- The database for the system shall use some form of polymorphism to improve on the current polymorphism implementation of using proxy models.
- The static content on the webpage shall be cached, with embedded content rendered dynamically on top.
- The website shall support the existing lovelace mark-up.
- The editing tools shall allow several different types of content, including text, images and embedded content.

In addition to the functional requirements we also set a quality requirement of developing the code in a way which would make it easy to add different types of content into the editing tools to allow extensibility, since our prototype won't include every type of content that you might want from the tools.

3.3. UI Design

The UI design of our editing tools began with making sketches for the editing tools. The idea was to keep the appearance of the website similar to the appearance of the current version, while integrating some editing tools that would only be visible for admins or teachers who typically edit the content. Our idea was to make buttons that would allow adding, removing and editing of content on the web page itself rather than have a separate page where the editing is performed.

Moodle [16] is a good source of inspiration for UI design, since it is popular and has a similarly modular approach to how the pages are constructed. Moodle has a toggle-able edit mode, which we have a similar design for. In edit mode each module has their own edit buttons, which is also similar to how we designed our tools.

Our design differs in that we don't open a separate page for editing. We have a pop-up that opens on the page itself. We did not implement the tools in a way that it displays the result in real time like in Moodle where the form used to edit and add content has way more functionality. In our implementation the different types of content use different forms, where Moodle allows several different types of content in the same form. We designed it this way since it's easier to implement, while still having the ability to achieve the same end results.

In figure 4 we have a general idea of how we thought the web page would look with the editing mode enabled. In our design the page consists of different modules each with their own edit and deletion buttons. Cutting the page into smaller pieces makes it easier to precisely edit smaller parts of the page, and makes it so the edit window doesn't get flooded with text as it would if the entire page was edited all at once. It also makes it easier to have different types of content on the page. If the page was completely made from one text field things like images and embedded content would need to be added through mark-up. In our implementation the mark-up should still work in the text field, since we have imported the existing parser, but we also made separate content types with different input forms to make it easier to add different types of content into the same web page.

Figure 5 is our design for the pop-up that would be used to create modules for the website. We created a toolbar on top which allows switching between different content types, and then labeled fields for all different types of information that you would need to add for that specific type of content. On the page we placed the pop-up to the side of the content so it doesn't block the user from being able to see what the page looks like at the moment.

The result we came up with is a relatively simple and intuitive design and should be easy to use for anyone, even if they haven't previously used the tools. We made sure the buttons are labeled with symbols that are easy to understand, and we separated the different modules to make it clear which module you would be editing if you clicked the edit button.

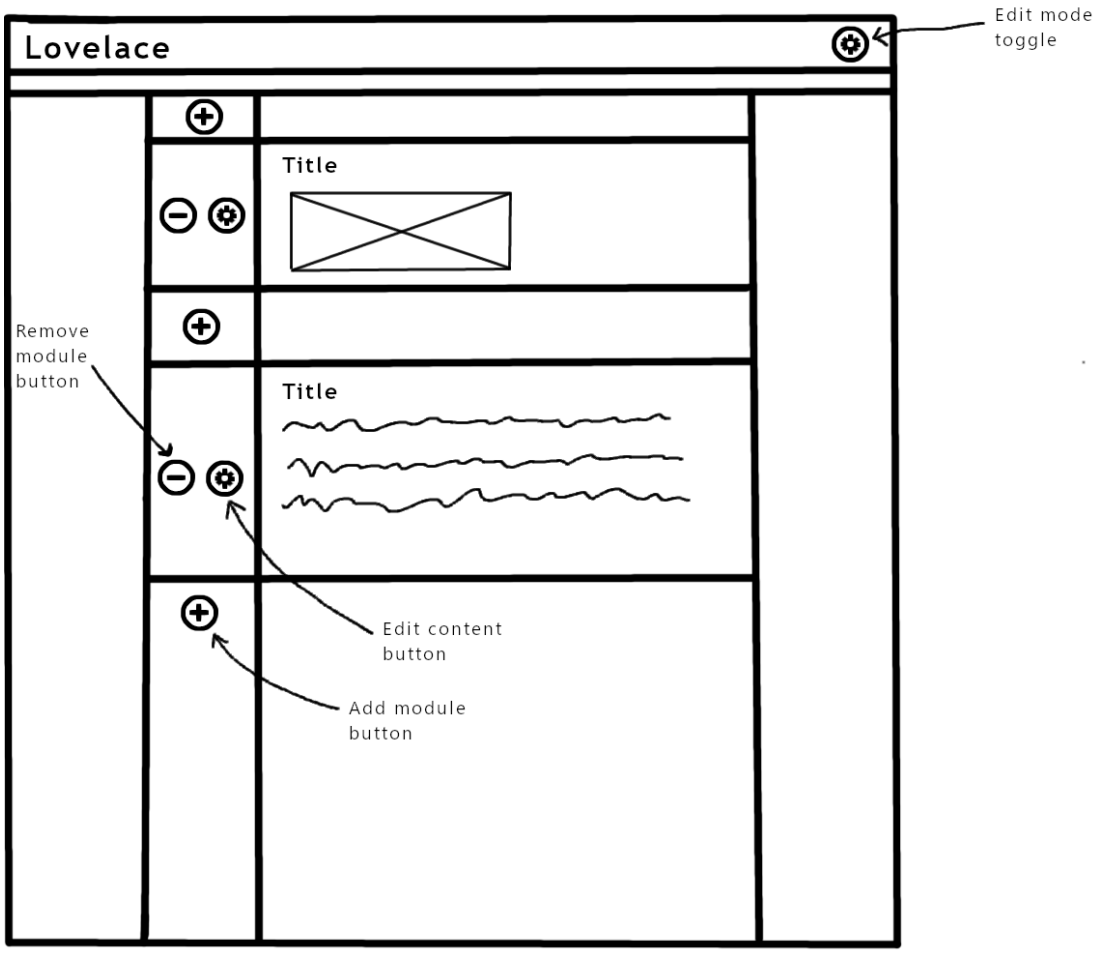

Figure 4. A sketch for the webpage with editing tools enabled

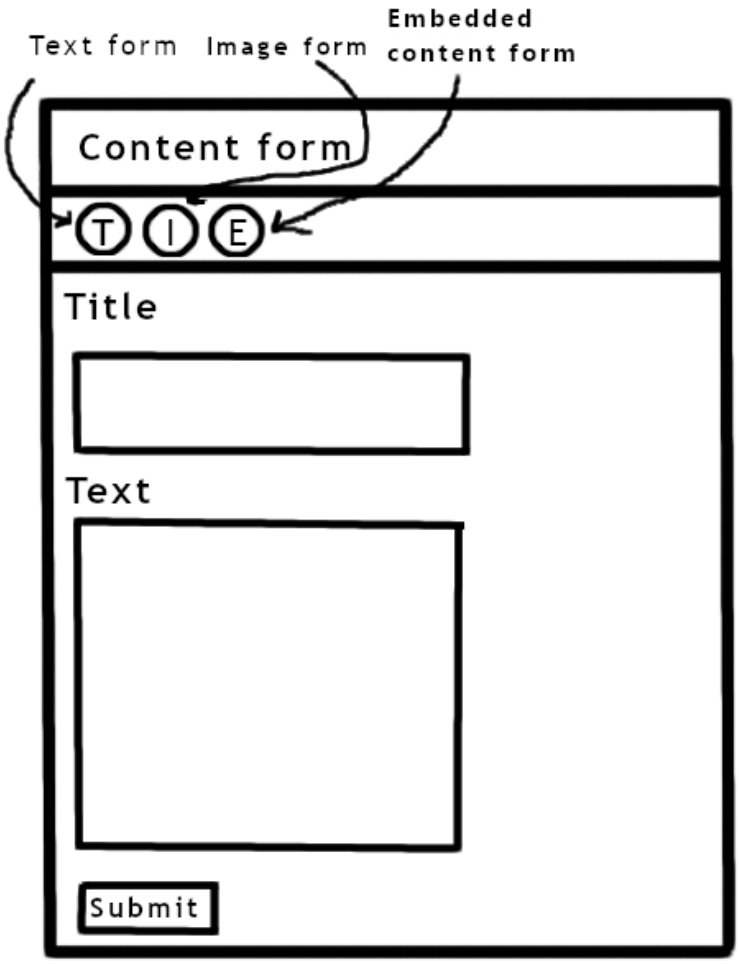

Figure 5. A sketch for the pop-up used to edit and add content

3.4. Django: Web-Development Framework

In our project we used the Django framework to create our prototype Lovelace content management system. Since the current Lovelace also uses Django it was necessary to use the same framework for our project so that it could work with the existing Lovelace. As a web-developement framework django has many features that makes it easier to develop web applications, such as models which make it easier to create a database, templates which allow easy modification of information on the website, and admin tools which are easy to use. The dynamic nature of the templates in particular is very crucial to the development of our content management system, which typically does require you to change the data that's shown on the website.

In our implementation we setup a database that the user can add different types of content to via the use of a pop-up with different forms, which are the way for visitors of a website to give input in HTML. Django has made the use of forms easier with it's own Form class. It automates many tasks that programmers would normally have to do themselves when dealing with forms. Django prepares and restructures the data for rendering, creates HTML forms for the data and processes the data received from the form.

Our current design includes forms for text content, a form for images, and a form which allows you to embed different templates into the page that you are viewing. These forms are each given an index based on which part of the website the user intends to add the content to. This index is used when the website is rendered from the template. The editable content of the website is added to the website through a for loop. which loops through a list with the content for the current lecture, which makes the content part of the website completely dynamic based on what content the user has added to the database for that specific lecture.

The user can also easily delete content from the website by pressing a delete button next to the content while in edit mode. This sends a request to remove the content from the database, and thus when the page is refreshed the content will not be loaded into the list of content which will be rendered. Editing content is also possible with a button next to the delete button. This opens a form with the fields for that type of content pre-filled with information that currently exists for the content at that index. The user can then perform changes to the data and submit it to get the changed view when the page is refreshed. Next section of the paper will explore the technical side of the implementation starting with polymorphism.

4. IMPLEMENTATION

4.1. Polymorphism Implementation

As stated before the way polymorphism was previously implemented in Lovelace was by using proxy models. The proxy models have had some problems in the past, which is why we were tasked to implement the polymorphism in some other way to improve on the previous design.

A key point of the Lovelace content model is to have the ability to reference all different types of content equally. Implementing the polymorphism in a way that conforms to this is not that straightforward however, since a foreign key can only reference one target table. The goal for us was to implement polymorphism that would support this key aspect of the Lovelace content model without utilizing proxy models.

Our solution for making the added content polymorphic was to use a simple "concrete model". We created a LectureContent parent class, which holds fields for Parent, which should link to the course instance that the lecture is part of, an index field which stores the index of the content on the page, and a ContentType field to specify what type that specific content is. All of these fields are necessary for every different type of content that could be added to the website, therefore we found this inheritance solution to be the easiest one to implement for our purposes, instead of for example an implementation that would use generic foreign keys, which would have been unnecessarily complex. A lot of our queries to the database are mainly for attributes that exist in the base class, therefore this implementation works well for us.

```
1 class LectureContent(models.Model):
2     Parent = models.CharField(max_length=100,)
3     Index = models.PositiveIntegerField()
4     ContentType = models.CharField(max_length=100,)
```

Listing 4.1. Parent class that all content classes inherit from

We created sub-classes for different types of content on the website. We have a class for text content, a class for image content, and a class for embedded content. All these content types have different fields that they use, which is why in order to remove redundancy from the database we created them as different sub-classes, to avoid populating data fields that are unnecessary for the specific type of content that is added. These sub-classes are mainly accessed in the template, in order to render the data on the page.

```
1 class TextContentModel(LectureContent):
2     ContentText = models.CharField(max_length=1000,)
3     ContentTextNotParsed = models.CharField(max_length
    =1000,)
4     ContentHeader = models.CharField(max_length=100,)
```

Listing 4.2. A subclass for text content

This inheritance implementation makes it easy to add new types of content through the creation of new sub-classes. If all the data was held in a single model, the addition of new content types would mean that all the pre-existing models in the database would need to have the new fields populated with possibly redundant data fields. This would still work with the way we've set things up, but it would mean that these redundant fields would use up space in the database.

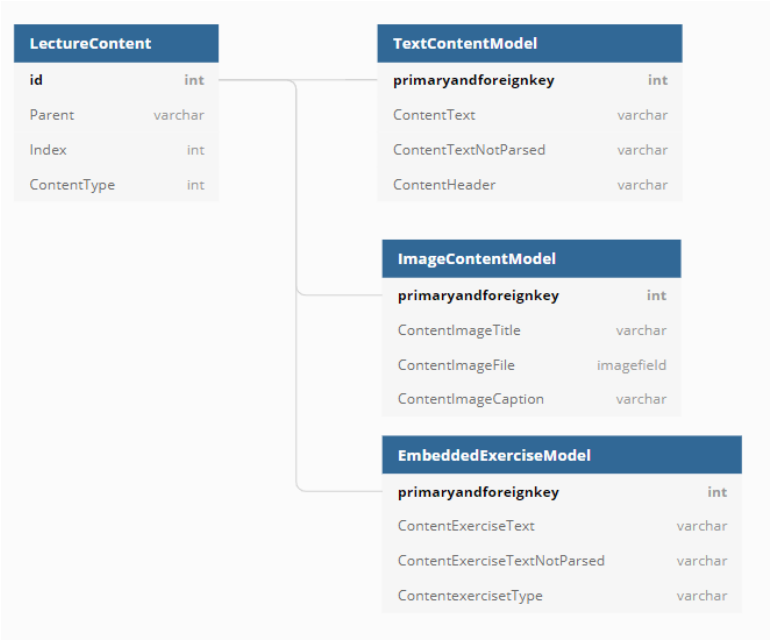

Figure 6. A diagram of our implementation of the database. All content specific tables point to the base table using a foreign key that also acts as the primary key of those tables

4.2. Content Rendering

Our content page is rendered by filtering through the database for LectureContent objects that have their parent field set to the lecture that the user wants to view. This queryset is then sorted based on the index, and appended into a list. This list is passed to the template, where there is a for loop that iterates through the list. Since the objects in the list are in order of index, the content will be rendered in the correct order. The template includes several if statements, which are used to determine how that specific piece of content will be rendered. Depending on what type of content it is the type specific database fields are placed inside different HTML tags. The content on the website is essentially pieced together every time the website is rendered, which means it's easy to dynamically change. The difference from the existing implementation is that each module is a separate piece on the page, where in the current implementation all static content is a single string of HTML which gets cut based on the location of the embedded content on the page.

```
1 for Content in ContentList:
```

```
2       if Content.ContentType == "Text":
3           <h1> Content.ContentHeader </h1>
4           <div> Content.ContentText </div>
5       else if Content.ContentType == "Image":
6           if Content.ImageHeader:
7               <h1> Content.ImageHeader </h1>
8           if Content.ImageFile:
9               <figure>
10                  <img src= "Content.ImageFile.url">
11                  <figcaption> Content.ImageCaption
12                  </figcaption>
13              </figure>
14      else if Content.ContentType == "EmbeddedContent":
15          <iframe src="Content.Index">
16          </iframe>
```

Listing 4.3. A pseudo-code demonstrating how the content page is constructed on the Django template using a for-loop every time the page is updated.

This way of rendering is what makes it possible for us to implement the editing tools. Rendering the content in smaller pieces allows our editing tools to remove or edit smaller parts of the website, without touching the parts that you don't want to edit. The only possible problem in this solution is the indexes being in wrong order or having missing indexes which would cause some problems with the editing. To counter this problem we have made functions that make sure that every time the page is rendered the indexes are in order and that the sequence of indexes doesn't include any missing indexes.

4.3. Content Forms

The current implementation of adding content uses a single text field that uses markup to display different types of content. In our implementation we decided to create different forms for different types of content to have a simple way to add them without using markup, and to make it function well with our modular design.

For every type of content we have created a separate form, which can be selected from buttons inside the pop-up that opens when the edit or add button are pressed. Pressing a button inside the pop-up makes that particular type of form visible, and hides all other types of forms using JavaScript. New buttons for new type of content can be relatively easily added into the pop-up.

If the user presses the edit button, the form gets automatically filled with pre-existing data from the database. The index field is always automatically filled based on which button is clicked, and it's only visible so that the user can confirm that the index is correct. For the text input field which uses Lovelace markup, we had to make 2 fields in the text object. One that has the text content parsed with the markup parser, which turns the markup tags into html tags, and one that is not parsed. This is because if we only saved the parsed version to the database and the user wanted to edit the text the

tags would be missing or incorrect. In our current implementation if the user wants to edit the text it loads the not parsed version of the text into the form, and the text that is rendered uses the version of the text that is parsed.

The forms are submitted through the post method, and the web page is refreshed through httpresponseredirect to the same page. The data from the form is submitted into the database if the request method is determined as POST in the views.py file, and the data on the form is valid. After the submission, since the page gets refreshed the list of content gets looped through and rendered again, and the new content that was added is instantly visible on the page.

The text input field specifically uses the current Lovelace markup. This allows stuff like making the text bold or italic using tags. Technically since we have the whole Lovelace markup imported, this text input field could be used to add images or other embedded content if it was integrated into the existing Lovelace. However, since we don't use any of the code from the existing Lovelace except for the css and the markup parser, it wouldn't work in our prototype.

4.4. Embedded Content

The exercises on the existing Lovelace are implemented as embedded pages. For our content editor we had to figure out a way to add embedded content to the page with our design of the editing tools. The goal was to allow adding of multiple different types of exercises to a single page using embedded pages that you can populate with information from our editing tools.

Our solution for implementing embedded content editing into our editing tools was to create a form that has 2 fields. The question you want to ask and the type of exercise it is. For demonstration we only created 2 types of exercises. A file upload exercise, and a text field exercise. The embedded pages use a separate exercise.html file which renders differently based on what type of content it is. If it's a text field exercise, it renders a text box that you can input text into, or if it's a file upload exercise it gives the user a file input.

In the parent page when the list of content for that specific lecture is created it also creates URLs for all of the different embedded pages that belong to that lecture. The URL given is the index of that piece of content on the page. The context for the embedded page is also given during the URL pattern creation, which decides what is actually rendered on the embedded website. For example, if there's embedded content at index 16, the URL that's created would be "/LovelaceContentPage/16". The LovelaceContentPage part would correspond to the URL for that specific course instance. Practically if in the future someone wants more data fields in the form for embedded content, it would probably be better to just give the parent lecture and index as context, and then query the database again in the exercise view function for entries matching the parent lecture and index, to avoid having to write a long list of context in the URL creation function. This would add some extra queries however, which would have a negative effect in performance, but would make it easier to create new types of content for the editing tool.

The design we have only includes a very basic embedded page with inputs that are not processed in the view function. For real exercise functionality the view function

for the exercises would need to be edited to enable handling of the requests from that embedded page, but it should work fine completely separately from our tools.

4.5. Caching

We used redis as the caching solution for the project. We cached the queried list of content for that specific lecture, and we also cached the result of the for-loop in the content page, as well as the forms used for adding content. The csrf token is the only part of the form we couldn't cache, since it has to be different every time the page is loaded.

The cache that basically contains the content that is added by editors of the page is cleared every time a valid form is sent by the user. This means that the page will always update every time something new is added or something is deleted. This is done because it's important to see the changes made to the page in real time if you are editing the page, because if the cache wasn't cleared the editor wouldn't be able to tell if the edits they made appear correctly on the page until the time to live of the cache expires.

By testing the load times we found that the caching improved our load times from being able to load 6 times a second to loading roughly 100 times a second on the same page with the same setup. This is a pretty significant improvement in performance, that might not make that much of a difference practically in smaller scale, but on a bigger page it could definitely make a noticeable difference.

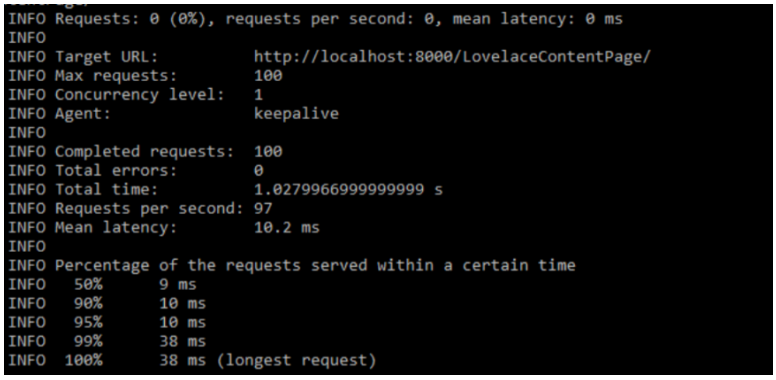

Figure 7. Loadtest results after implementing caching

5. EVALUATION

There are a lot of different UX (user experience) evaluation methods, of which 86 are listed in the *All About UX* website [17]. Since the purpose of this project was to update the current back-end and usability of Lovelace, the users we focused into are the teachers that create and maintain the content. Therefore the most obvious method of UX evaluation is the expert evaluation [18]. Rosenzweig defines the key factors in usability evaluations: *design*, *learnability*, *efficiency*, *user satisfaction* and *errors* [19]. These factor also play well in this project. In terms of learnability, when creating a new UI (user interface), it is important to take into account that how well the end user adapts the changes. Using these factors in the evaluation is a major part of this project. In their book Dumas and Redish state that there are five characteristics that every usability test share: 1. the primary goal is to improve the usability of a product, 2. the participants represent real users, 3. the participants do real tasks, 4. the output from participants is recorded and observed, and 5. the output data is analyzed and problems are diagnosed and fixed based on the findings [20].

This part of the thesis will be focusing on the specifics of the ways the project was evaluated. First parts of the section will dive deep into the specifics of how the evaluation was planned, what were the desired outcomes. After these topics the thesis moves on to the way the evaluation was executed in practice. The last part will be focusing on the analysis of the gathered data. This will include feedback analysis and comparing the before mentioned desired outcomes to the data gathered from actual testing.

5.1. The Plan for Evaluation

Since the main task in creating the new version of Lovelace was making the experience of the teacher more user friendly, testing the software in conditions comparable to reality is the best way of evaluating the new version. Also when caching is introduced, the effectivity of it has to be evaluated using the correct tools.

5.1.1. Tools for Measuring and Debugging

Django provides its users with a wide variety of possibilities to gather information about the product created using the framework. Built in Django Debug Toolbar offers panels from which vital information can be observed.

Usage requires the DEBUG value to be True in the settings.py file. The toolbar is designed to have as little hit as possible to the performance of the program. Depending on the project however, the impact may be severe and the root of this problem could be in one of two places, the gathering or the rendering phases. The Django toolbar has two phases it cycles through. In the gathering phase, data is gathered at the same time as Django is handling requests and storing the data in memory. The second rendering phase occurs when the saved data gets fetched from the database and gets displayed. Both of these phases have to be optimized so that the tools function at an acceptable standard.

5.1.2. The Process and Desired Outcomes

Having the fastest loading times as possible requires caching to be implemented correctly. To test this feature the evaluation is fairly straight forward. By using the before mentioned Django Debug Toolbar and the multiple default panels it provides precise measuring of processes is possible. By comparing the loading times between the website with caching enabled and disabled the efficiency of the feature can be determined and improved upon. The expert evaluation that will be executed with real teacher with background with the original live version of Lovelace will be very important data as well. Since the whole project is focused solely on the experience of the teacher rather than the student, real feedback is the best thing the project group could hope for when focusing on evaluation. The desired outcome would be positive feedback from the expert, with possible hints on how to fine tune the software.

5.1.3. Task-Based Expert Evaluation

As the quality of the final product is reflected solely on the experience of the users utilizing the software itself, comparing the old Lovelace with the project groups creation must be executed. In order to have an accurate representation of a normal use case the target group has to be previous user of Lovelace. This way the feedback and input received from the experience is topical and from the context of an actual teacher. The implementation is solely focused on making the experience of the teacher more fluent so the target group for the use cases is teachers with experience on basics of online learning environments.

Since the evaluation is done with test cases with real experts, the group of the participants can be much lower than with less experienced group. In practice the meeting will be approximately one hour long, and in that time the project group and the teacher will join together in Microsoft Teams [21] meeting. One of the group members will have the previous version and the new version of the Lovelace open, with identical content. Since the actual using of the prototype required RediSQL server to be active, this was the easiest way of giving the teacher access to the new prototype version without the need of installing additional software on the attending teachers computer. The advantage in using Microsoft Teams is that when one of the group members share their screen, they can give control to anyone inside the meeting. This includes the access to using the mouse and the keyboard. The whole meeting will be recorded for later investigation and reflection.

5.1.4. Test Case Format

After completing the given task with the old version, the following question will be asked before getting access to the new prototype

1. "What would you like to be improved in the Lovelace experience?"

After getting some information on the possible ideas the teacher has to improve the software, we dive into the given task. First the task is explained clearly to the attendee.

When teacher is fully aware of the task at hand, he will be given countdown. When timer hits zero, the teacher has to accomplish the task as fast as possible. This will be timed and referenced upon later. When teacher and the project group agree that the task is done, the timer will be stopped and the time will be stored. The following questions will be presented to the attendee.

1. "Are you happy with the total time the task took to accomplish?"
2. "What differences did you notice between the old and the new versions?"
3. "Were there something that was left to be desired?"
4. “Did you find the user interface hard or easy to understand?"

When the attendee has answered to these questions with the desired amount of information, the first iteration of the testing is over. There are in total three tasks, of which each has to be accomplished on the both versions of the Lovelace website. The total amount of iterations will be six. The following list are the different tasks given to the attendee.

1. Task: Remove one sentence from a lecture of the teachers choice. When this is done, the teacher has to return to the page and verify that the change has updated on the actual website.
2. Task: Remove an entire module which in this context is an header and the content below it. For example an section talking about a specific subject. This section has to be then replaced with any image the attendee chooses. When this is done, the teacher has to return to the page and verify that the change has updated on the actual website.
3. Task: The teacher has to locate an embedded page inside the lecture page and edit its content. For example the text in the page can be removed or overwritten. When this is done, the teacher has to return to the page and verify that the change has updated on the actual website.

This will be the end of the expert evaluation, and the next step from here will be rewatching of the recording and analysing the taken notes. The transcripts are available for exploration in appendix section.

5.2. The Evaluation Process

On the third of may, a meeting was able to be arranged from 17:00 to 18:00, and during this time the process would be the exact one explained in the previous section. The attendee is a Phd student and a researcher at the University of Oulu. The focus of his research consist of integrating ICT innovations into learning environments which is closely related to the projects scope. In case some of the questions do not appear at some tasks, the answer to them was either unclear or unimportant.

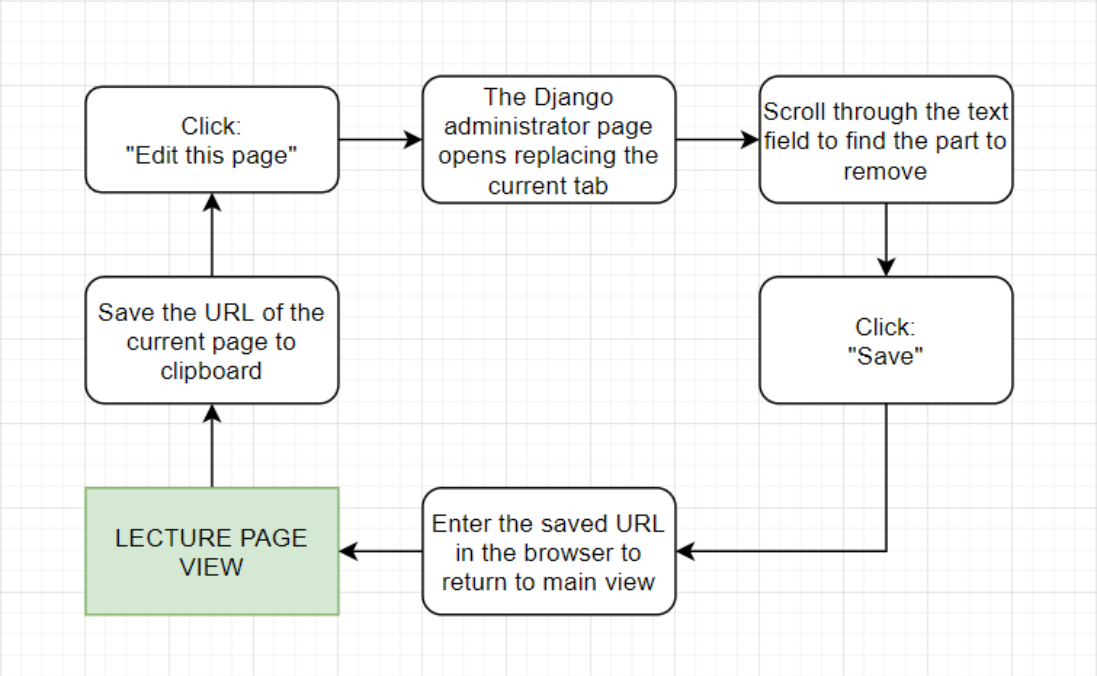

Figure 8. Flowchart of the Task 1 with current live version

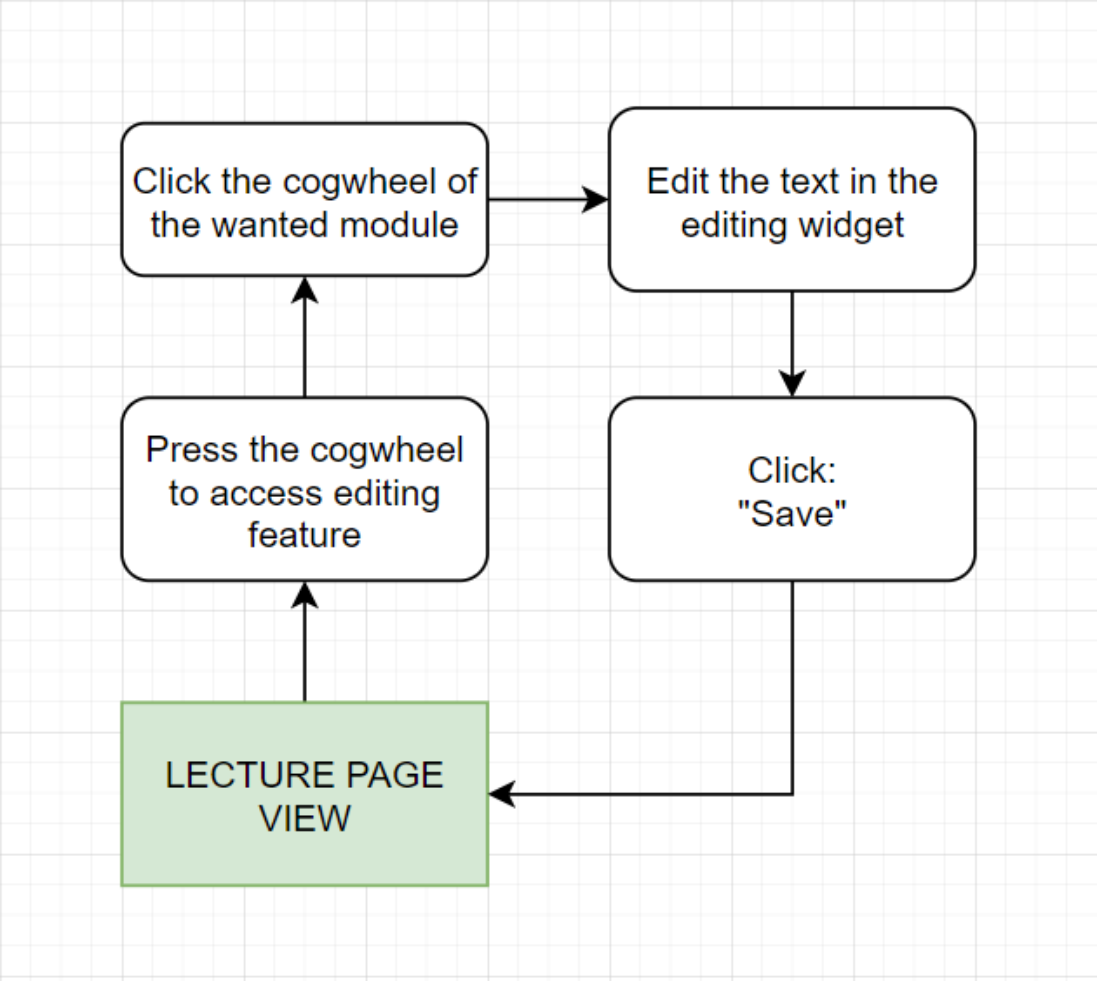

Figure 9. Flowchart of the Task 1 with the prototype

5.2.1. Task 1: Remove a Line of Text

The first task was to edit a line of text somewhere in the lecture page in the original Lovelace web page. We encountered small technical problems at first with the shared screen, but eventually we got it working. Attendee understood the task and timer started counting. The attendee completed the task like he has done many times in the past, by pressing the "edit this web page" button which directs the user to the Django admin page. This is a separate page where the user can edit the content of the site. He accessed the lecture pages text section, which includes everything from headers, text, images and embedded content. He couldn't however, find the publicly available web page to show that the page has updated. Our group had to intervene and enter the URL of the lecture page to show it again. Following the first task with the original Lovelace version, the before mentioned question were provided to the attendee.

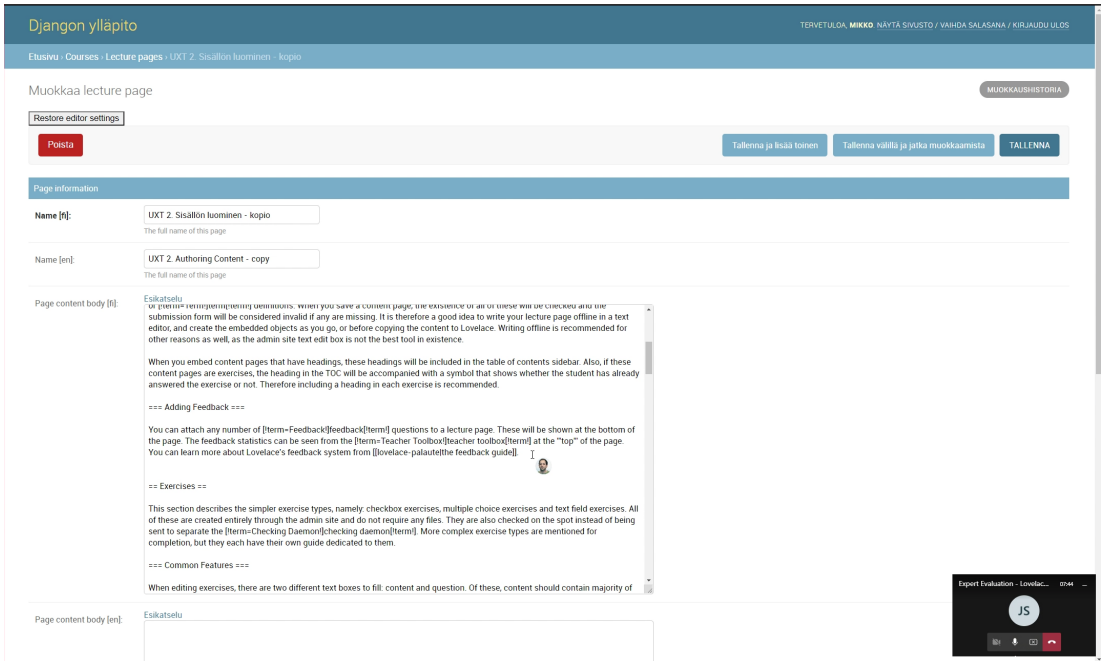

Figure 10. The attendee editing the page using the old Lovelace version.

Next step was to do the exactly same task but on the prototype version made by our project group. After the counting started and the attendee introduced himself to the new version, he had little trouble enabling the editing function which is a cogwheel in the top right corner of the page. After finding the button the editing went clearly much faster compared to the old version. Right after the page had been updated and the test was over, the attendee was clearly impressed. The first note that he said was "Much better, I can tell you". Continuing with our format, we asked the questions mentioned in 5.1.4.

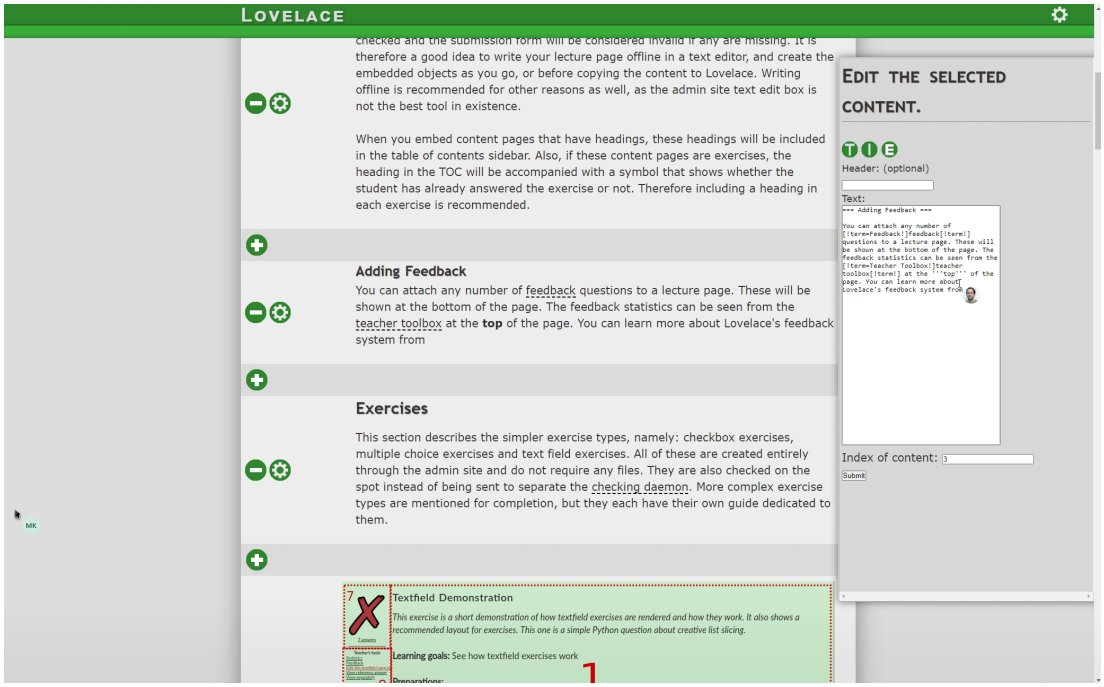

Figure 11. The attendee editing the page using the new Lovelace prototype.

5.2.2. Task 2: Remove a Module and Replace It with an Image

After completing the first task on both new and old versions of Lovelace, the next step was to introduce the attendee to the next task which is removal of a entire module and replacing it with an image of the attendees choice. The attendee immediately started contemplating if he even remembers how one can even add an image on the current live version. When the attendee started the task after the countdown began, he deleted the header and the text below it to achieve the removal of the so called module. Then began the hard part, when the attendee had little to no knowledge on how the image should be added syntax-wise. He mentioned that in normal circumstances he would have separate markup tips on a separate monitor. This is due to the difficult nature of the Lovelaces syntax compared to other learning environments. Since he couldn't add the image in place of the removed "module", he had to copy a reference from another part of the same lecture which had an image imported. After the image was imported, the attendee found himself in the same situation as in the last task. He could not enter the updated page, and we had to get him the URL of the page once more, so that the new content could be seen.

Then it was time for the prototype version. Counting started as it has before and the attendee found the editor from the cogwheel easily this time since he had some experience already. Pressing the fairly obvious minus logo, he deleted the module. Next the task was to add an image in place of the module that was just removed. This did not take too long either. He pressed the plus sign, the button labeled "I", and added an image. After this was done the countdown stopped and questionnaire began as usual.

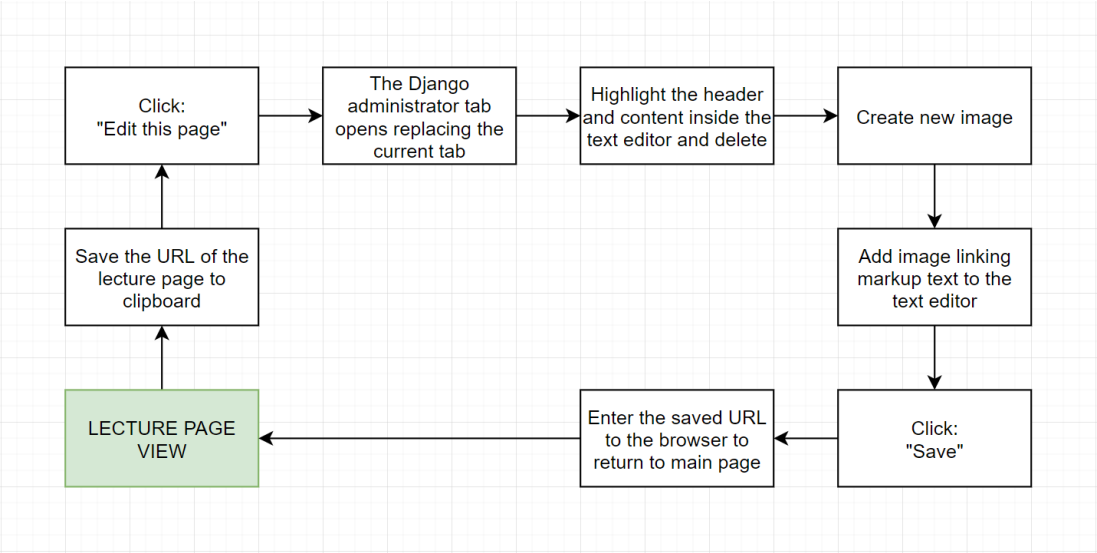

Figure 12. Flowchart of the Task 2 with current version

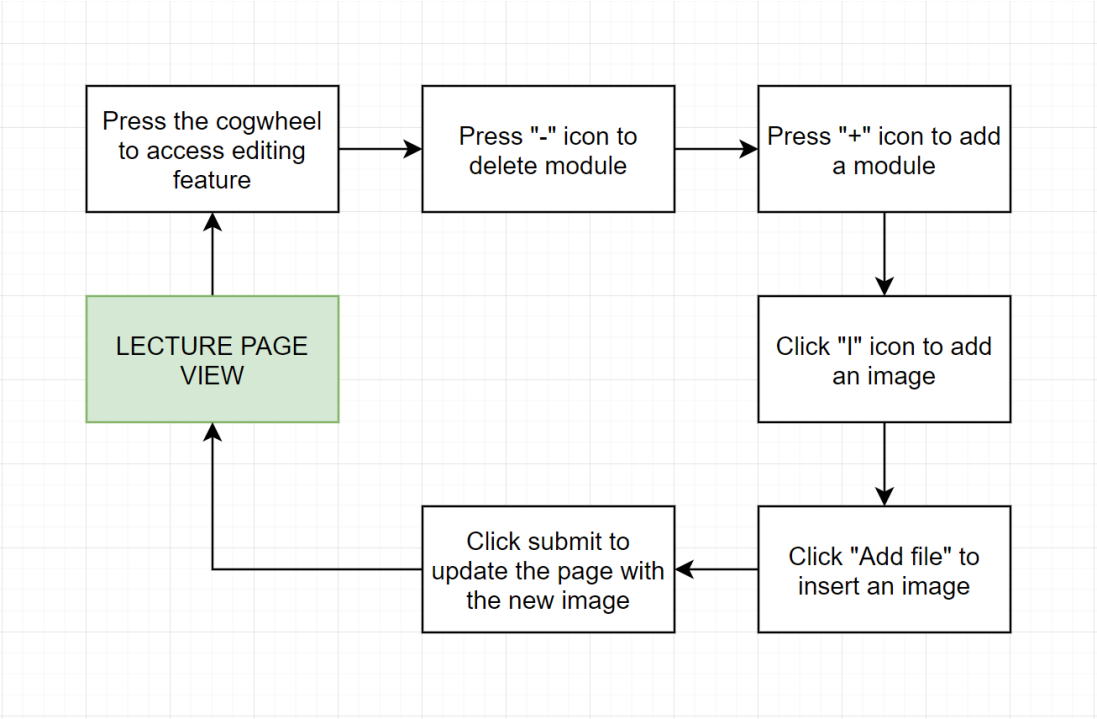

Figure 13. Flowchart of the Task 2 with the prototype

5.2.3. Task 3: Edit an Embedded Exercise

The last task was to edit an existing embedded page. In this context it means an exercise inside the lecture. This went quite smoothly from the attendee. He completed the task as instructed and it was time for the follow up question.

The last task of the session was to do the Task 3 with the new prototype. As the attendee had previous experience from the tasks before this one, he had no problem accessing the editor. After scrolling down to the exercise that required the editing, he quickly realised what needs to be done. The last task was done and then it was time for the last questions of the session.

The meeting was officially over and in the end the attendee pointed out that we as project group have done a good job and the Lovelace web page is going to look more usable. Lovelace was said to be quite a nightmare to edit from admin point of view. Overall this expert evaluation from the point of view of a experienced teacher was a huge success and the gathered data was very valuable in further analysis of the user experience. This ends the test case portion of the thesis and next is the full analysis of the data from both expert evaluation and the Django Toolset measurements. The transcripts of the entire expert evaluation meeting are can be found in the appendix section.

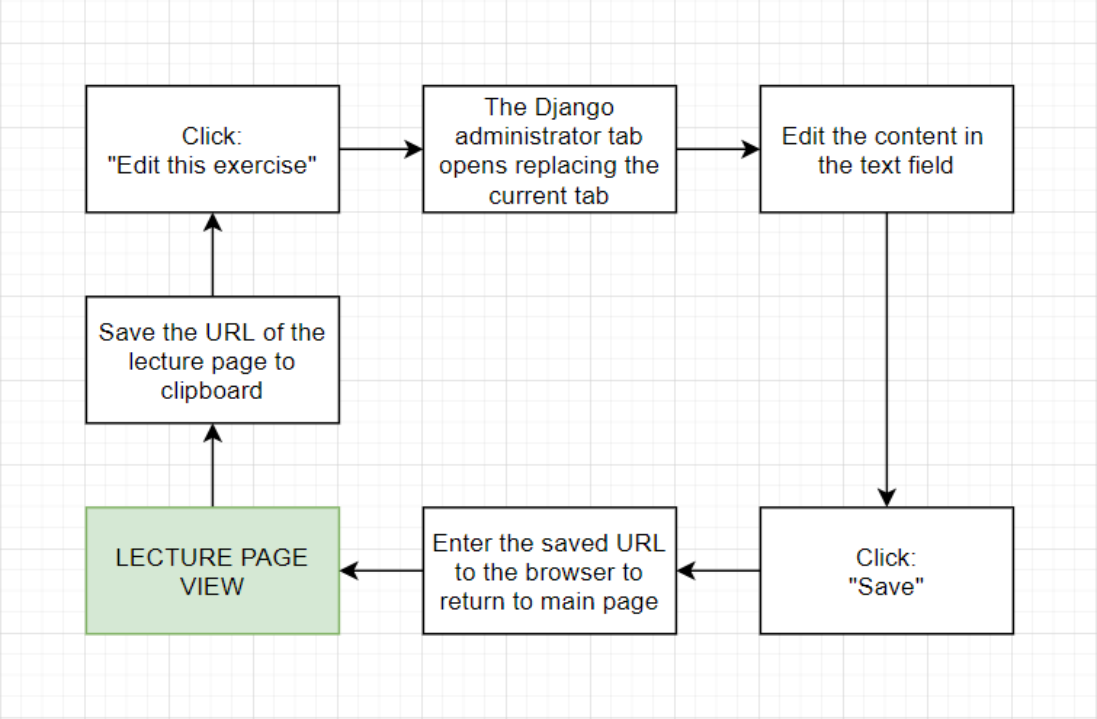

Figure 14. Flowchart of the Task 3 with current live version

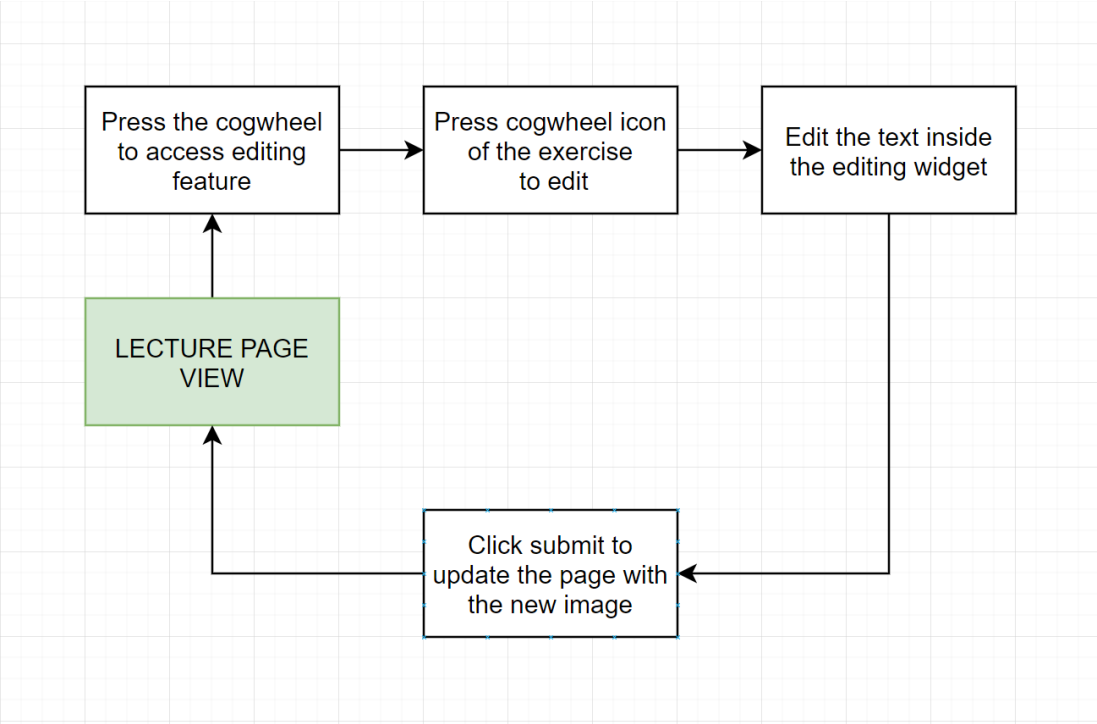

Figure 15. Flowchart of the Task 3 with the prototype

5.3. Data Analysis

The previously introduced test cases provided us with great amount of data regarding the improvements from the old version. As mentioned before, the performance was timed during every task, and from this we can derive quite confident conclusions. Comparing the first task between the old and the new, the difference between the efficiency of the web page was obvious. The live version took the attendee to complete 1 minute and 15 seconds. The prototype instead sped up the performance to only 34 seconds. That is a difference of 41 seconds. This is great news, since this clearly shows that our groups efforts were improving the workflow of the teacher.

Task 2 was no different from the first one. The live version took the teacher 1 minute and 47 seconds to complete, which in itself is quite underwhelming performance, since these are tasks that are done multiple times per average session. The prototype version however, took 47 whole seconds away from that time, with execution time of just one minute. The new version took only 56 percent of the time that the old time would've taken. That is very impressive.

Last but not least was the Task 3 with the embedded exercises editing. Here we did not see that big of a difference, since the live version took exactly one minute and the prototype took 50 seconds to complete the task. This does not change the fact that editing is much more enjoyable with the prototype, being able to see the page while editing. The live version requires the teacher to leave the actual page for the Django admin page, where the changes can be done.

6. DISCUSSION

The goal of this project was to enhance the usability of Lovelace, a online learning environment that is in use in the University of Oulu. The learning curve of the previous version of Lovelace was steep because the admin system of Django differences from the most regular ones. Previously the users (teachers) had to use a completely different page for editing the content. Our goal was to allow the creation and editing of new content directly from the page, so that the changes are instantly visible. To achieve the current state of the prototype, the project group had to go through intense learning of the various tools required for the task at hand. The tools have been introduced in earlier parts of the paper. Most of the time the group worked on the project was used studying the tools. Prerequisites for the project were very clear, so there were little to no flexibility in the tools used. For example, the web framework Django had been used in the previous version, thus the new prototype had to be built on that framework as well.

It is important to analyse the requirements that were set out for the finished prototype and how the group succeeded in achieving these goals. The tools that instructions set out for us were listed clearly before hand and looking at the current version of the prototype we stayed in the exact tools provided. We did not venture out to different tools, which was the desired result. The growth the group has gone through has been incredible. We were told to not look at the original code for the back-end of the live version so that the vision for the prototype would not be altered by the past. This made the work both difficult and easy at the same time. Biggest reason for the difficulties was the groups inexperienced past, and the less examples we looked at, the more intimidating the process was. This on the other hand allowed us to also be much freer to try different things and experiment.

When we look at the requirements that were set before we began working on the actual project we can now reflect on how well the requirements were met. The first requirement that was set was to have a system that allows adding, removing and editing of content on the lecture page. With our editing tools we have managed to fulfill this requirement, since in the current version it is possible to add and delete all the types of content that we are currently supporting. Editing is also possible for text content and embedded content, however the editing functionality for image content was a bit harder to implement, since it didn't seem possible to pre-select the previous image file in the form for images, therefore making editing for images mostly pointless since you would still need to go and find the image in your files again.

Regarding our implementation of polymorphism for our database, we did manage to implement it in a different way compared to the previous proxy model solution. The "concrete" model of polymorphism seems to work well for the tools so far, but it's not certain if it would be a better solution in the long run. We can't really say if it is objectively superior to the proxy model implementation either, since we do not have the experience of working with the proxy models.

The caching works according to the set requirements. We cached basically all the static content on the website. The static content inside the embedded content is not cached in our current implementation however and we do not have user specific content since the prototype doesn't have a user system or a system checking the submitted content on the exercises. The caching for the embedded content would have to be

implemented on the view function for the different exercises, and since the embedded content doesn't have much functionality in the prototype we didn't think caching it was that important at the time. The caching did manage to speed up the load time significantly either way so the desired result was achieved.

The prototype currently does support the Lovelace markup in terms of most of the functionality. You can do all the stylizing for the text and technically the embedded content adding and images would also probably work if the linked content did exist. Our tools are completely separate from the way the database for the existing Lovelace is setup so we can't use the markup to link content added with our tools unless we edited the markup parser. The functionality of adding images and embedded content through markup would clash with the basic idea of our tools since we have designed different forms to enable adding of different types of content instead. It could however be useful in the sense that it would allow easier mixing of different types of content so maybe in the future it would be useful to allow linking the content added by the tools using markup.

Based on the results of our evaluation we think we managed to improve the process of editing content on Lovelace. The expert that we interviewed noted that the editing on the page itself was a lot better than accessing the Django admin page, and our results suggested it's quicker to edit the website using the tools instead of using the admin page. The results matched the expected improvements that we had set for our project, and solved one of the major problems in the previous design of having to use the admin page. With these things in mind we believe that the end result of the project was successful. Here however it has to be kept in mind that 5.2.2 Task 2, which was to remove a module and replace it with an image, was not correctly executed on the old version. Since the attendee couldn't remember how to add an image, he had to copy markup text from existing section which compromises the end results from this task. The results also might be slightly different in the meeting compared to regular use because the Microsoft Teams meeting is not as comfortable of an environment compared to a home office with no people watching the task being done. The attendee mentioned, that in normal circumstances he has window about Lovelace markup information open on a second screen, which was not thought of before the meeting took place.

Features that would enhance the prototype from its current stage would be the addition of termbank which is in the live version as well as a find and replace feature. Find and replace would enable the teacher to look for certain word for example, and then replace all occurences of the word with what the teacher wants to. This would elevate the editing functionality even further and currently the live version beats the prototype in this regard by having the whole lecture page in a single text file. During expert evaluation the attendee mentioned that one problem with the prototype was that after editing the website updates and jumps to the top of the page. Letting the user to stay at the last edited section would streamline the experience further and is high on the priority list.

7. SUMMARY

This project contains the whole process of achieving a new back- and frontend prototype of the online learning environment Lovelace. The pre-requisite was to have roughly the same technologies as the current live version, and the project groups first major task was to tackle the learning of many of them, including web development framework Django, database software RediSQL, HTML, CSS and Javascript. The design of the website was the main part of the project so the project group went through many iterations of sketches of what the possible result would look like. The live version requires the admin to access a separate admin page, which makes the editing of the content very cumbersome. To make the editing less complicated, the prototype introduces a on-page editing system. This way the review and editing happen simultaneously, so that the admin must never leave the actual lecture page. The prototype was introduced to an experienced teacher with long background with Lovelace learning environment. They were made to do certain tasks on both new and old versions, so that the changes in effectivity would be as clear as possible. This expert evaluation ended up being fruitful and the data collected was very constructive but also positive. The project was difficult especially in the beginning with all the new the group had to learn, but started to flow towards the end.

8. REFERENCES

[1] Proxy models in Django. URL: https://docs.djangoproject.com/en/3.2/topics/db/models/#proxy-models. Accessed 4.6.2021.

[2] Plekhanova J. (2009) Evaluating web development frameworks: Django, ruby on rails and cakephp. Institute for Business and Information Technology , pp. 1–20.

[3] Grinberg M. (2018) Flask Web Development: Developing Web Applications with Python. O'Reilly Media.

[4] Web Development Frameworks. URL: https://www.plesk.com/wiki/web-development-frameworks/. Accessed 5.2.2021.

[5] Security in Django. URL: https://docs.djangoproject.com/en/dev/topics/security/. Accessed 8.6.2021.

[6] (2009) Threat modeling for CSRF attacks. 2009 International Conference on Computational Science and Engineering. Accessed 9.6.2021.

[7] What are XXS attacks. URL: https://portswigger.net/web-security/cross-site-scripting. Accessed 9.6.2021.

[8] What is a Relational Database (RDBMS)? URL: https://www.oracle.com/database/what-is-a-relational-database/. Accessed 6.6.2021.

[9] Object-relational Mappers (ORMs). URL: https://www.fullstackpython.com/object-relational-mappers-orms.html. Accessed 5.6.2021.

[10] Django Documentation. URL: https://docs.djangoproject.com/en/3.1/. Accessed 2.2.2021.

[11] E.B.Ford (1945) Polymorphism.

[12] Redis Introduction. URL: https://redis.io/topics/introduction. Accessed 2.2.2021.

[13] Manhas D. (2013) A study of factors affecting websites page loading speed for efficient web performance. International Journal of Computer Sciences and Engineering .

[14] Al-Ajlan A. & Zedan H. (2008) Why moodle. In: 2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems, pp. 58–64.

[15] Aydin C.C. & Tirkes G. (2010) Open source learning management systems in e-learning and moodle. In: IEEE EDUCON 2010 Conference, pp. 593–600.

[16] Moodle Documentation and Features. URL: https://docs.moodle.org/311/en/Features. Accessed 4.6.2021.

[17] All About UX. URL:http://www.allaboutux.org/. Accessed 26.5.2021.

[18] All About UX. URL:http://www.allaboutux.org/ux-expert-evaluation. Accessed 26.5.2021.

[19] Rosenzweig E. (2015) Successful User Experience: Strategies and Roadmaps. Morgan Kaufmann Publishers.

[20] Dumas J.S. & Redish J.C. (1999) A Practical Guide to Usability Testing. Intellect Books.

[21] Microsoft Teams. URL:https://www.microsoft.com/fi-fi/microsoft-teams/group-chat-software. Accessed 23.5.2021.

9. DISTRIBUTION OF CONTRIBUTION

The following table shows approximately how the work divided among the group members during the entire project. Originally the group consisted of three members, but due to personal reasons could not continue with the BSc thesis and only completed the Applied Computing Project I course.

Mikko Koivula spent approximately 40 hours for study and groundwork, 30 hours for development and 90 hours for writing the thesis

Joonas Sutinen spent approximately 10 hours for study and groundwork, 90 hours for development and 60 hours for writing the thesis

Group Member	Part	Completion	Hours spent
Mikko Koivula	1.0 2.0 2.1 2.2 2.2.1 2.7 3.4 5.0 5.1 5.1.1 5.1.2 5.1.3 5.1.4 5.2 5.2.1 5.2.2 5.2.3 5.3 6.0 7.0 10.0	BSc Thesis	160h
Joonas Sutinen	2.2, 2.3, 2.4, 2.5 3.2, 3.3, 3.5, 4.1, 4.2, 4.3, 4.4, 4.5, 6	BSc Thesis	160h
Mikael Karvonen	2.5, 5.0	ACP1 Course	x

10. APPENDICES

Appendix 1 Transcripts of Expert Evaluation

Task 1: Live Version

The Interviewer: What would you like to be improved in the Lovelace experience?
The Attendee: When I change something, it is really difficult to see what actually has been changed. It should have some kind of link, so that when something has been changed, the user sees automatically the updated page.

Task 1: Prototype Version

The Interviewer: Are you happy with the total time the task took to accomplish?
The Attendee: Yes, this is much better I can tell you.
The Interviewer: What differences did you notice between the old and the new versions?
The Attendee: The editing while seeing the original the page at the same time was a lot better than accessing the Django admin page like in the previous version. The editing widgets make it easier to see what part the user is actually editing. You can also see the changes automatically without any extra work.
The Interviewer: Were there something that was left to be desired?
The Attendee: One suggestion would be that instead of having the editing box on the right, it would be on the bottom of the screen.
The Interviewer: Did you find the user interface hard or easy to understand?
The Attendee: The logos from which you can choose between text and images etc. are hard to understand.

Task 2: Live Version

The Interviewer: What would you like to be improved in the Lovelace experience?
The Attendee: For instance if I want to add an image I would like to have a shortcut somewhere where I can add a image straight away. In removing process I would like to be able to remove the modules from a button instead of dragging a whole section (in text editor) and removing it.

Task 2: Prototype Version

The Interviewer: Are you happy with the total time the task took to accomplish?
The Attendee: Yeah.
The Interviewer: Did you find the user interface hard or easy to understand?
The Attendee: It is a little bit confusing. I was thinking that when removing a section there would be another option to add an image. Also when I opened the window

(editing window), I did not know what the T, I and E were. I was guessing that the T was for text and I was for image.
The Interviewer: Were there something that was left to be desired?
The Attendee: You could have some kind of contextual help. When you put your mouse on top you could see add text, add image etc.

Task 3: Live Version

The Interviewer: What would you like to be improved in the Lovelace experience?
The Attendee: Basically this is exactly the same as the previous one. There should be some way to access the edited page

Task 3: Prototype Version

The Interviewer: Are you happy with the total time the task took to accomplish?
The Attendee: Yeah.
The Interviewer: Were there something that was left to be desired?
The Attendee: The editing box is too small. I would like to have bigger space for editing. Another thing that I realised is that when you save, you go directly to the beginning of the page. Perhaps it would be better if you stayed in the same place because it is likely that you are going to edit something in the same area.
The Interviewer: Did you find the user interface hard or easy to understand
The Attendee: I would like to have bigger space for editing.