

# Design Report

## TWENTE HACKING SCHOOL

### Group 9

Edi-Cristian Berisha

Alexandra Gheorghe

Víctor Delgado Plácido

Cristina Maria Toader

Jaridza Tromp

Tim Wijma

**Supervisor**

Thijs van Ede

**Date**

17-04-2025



# Abstract

As part of our final year Design Project at the University of Twente, we were tasked by the team behind the Twente Hacking Squad with expanding their educational platform into a new platform called the Twente Hacking School. This cybersecurity training platform allows teachers to upload materials and create hacking challenges that run in isolated Docker environments and can be solved by students to practice their skills in a safe and realistic environment. In the context of this project, we extended the system's functionalities to, among others, allow the creation of courses, and improve the existing role-based system. Our goal is to make learning cybersecurity more interactive, practical and accessible, while also offering a platform where teachers can upload information and design creative challenges for their students. This report elaborates on the design process of the developed system, describing all of the phases that we went through while working on the project, such as extracting requirements, explaining our design choices and outlining the system architecture. While working on the project we used the SCRUM methodology, defining sprints and deadlines that we all kept notice of. In the future, this system might be deployed and used to replace the current system, the "Twente Hacking Squad".

# Contents

<b>Abstract.....</b>	<b>2</b>
<b>1. Introduction.....</b>	<b>6</b>
1.1 Project Plan.....	6
<b>2. Domain Analysis.....</b>	<b>7</b>
2.1 General Knowledge of the Domain.....	7
2.2 Problem Statement.....	7
2.3 Client Vision and System Goals.....	7
2.4 Stakeholder Analysis.....	8
2.5 Software Environment.....	8
<b>3. Global Design Process.....</b>	<b>9</b>
3.1 Design Methodology.....	9
3.2 Initial Concepts and Brainstorming.....	9
3.3 Requirement Gathering.....	10
3.4 Mock-ups.....	14
3.5 Preliminary Design Choices.....	14
3.5.1 Tools and Software used.....	14
3.5.2 Architectural design choices.....	14
3.5.2.1 Role system.....	14
3.5.2.2 Course shell.....	15
3.6 System Overview.....	15
3.6.1 Courses Dashboard Page.....	15
3.6.2 Course Homepage.....	16
3.6.3 Course Pages Overview Page.....	16
3.6.4 Course Page Page.....	16
3.6.5 Course Users Page.....	16
3.6.6 Course Highscores Page.....	16
3.6.7 Challenges Page.....	17
3.6.8 Challenge Creation and Editing Page.....	17
3.6.9 Users Page.....	17
3.7 Activity Diagrams.....	18
3.7.1 Creating a New Challenge.....	18
3.7.2 Creating a New Course.....	19
3.7.3 Adding Pages to a Course.....	20
3.7.4 Adding Users to a Course.....	21
<b>4. Implementation &amp; Development.....</b>	<b>22</b>
4.1 System Description.....	22
4.2 Front-End Diagram.....	24
4.3 Design Challenges and Solutions.....	24
4.3.1 Role permissions.....	24

4.3.2 Challenges in a course.....	24
4.3.3 Highscores in a course.....	25
<b>5. Testing.....</b>	<b>26</b>
5.1 Testing Strategy.....	26
5.1.1 Approach.....	26
5.1.2 Unit Testing.....	26
5.1.3 System Testing.....	26
5.1.4 User Testing (Usability Testing).....	26
5.1.5 Penetration Testing.....	26
5.2 Software risk issues.....	27
5.3 Functionalities to be tested.....	27
5.4 Pass/Fail Criteria.....	28
5.5 Schedule.....	28
5.6 Test Results.....	29
5.6.1.1 Unit tests.....	29
5.6.1.2 System Tests.....	29
5.6.1.3 User Tests.....	29
<b>6. Future Planning.....</b>	<b>32</b>
6.1 Features to be implemented.....	32
6.1.1 Responsive design.....	32
6.1.2 More options inside a page.....	32
6.1.3 Course, page, challenge and hint reordering.....	32
6.1.4 Email notifications.....	33
6.1.5 History of edits.....	33
6.1.6 “Vuetifying” previous components/pages.....	33
6.2 Future development.....	33
6.2.1 User roles.....	33
6.2.2 Frontend Architecture.....	34
6.2.3 Backend Architecture & API.....	34
<b>7. Conclusion &amp; Evaluation.....</b>	<b>36</b>
7.1 Conclusion.....	36
7.2 Evaluation.....	36
7.2.1 Must.....	36
7.2.2 Should.....	36
7.2.3 Could.....	36
7.2.4 Won’t.....	36
7.2.5 Non-functional requirements.....	37
<b>8. Bibliography.....</b>	<b>38</b>
<b>9. Appendices.....</b>	<b>39</b>
Appendix A: Mock-ups.....	39
Appendix B: User Test Scenarios.....	44

Scenario 1: Creating a challenge(Teacher).....	44
Scenario 2: Editing a challenge(Teacher).....	44
Scenario 3: Edit a course (Teacher).....	44
Scenario 4: Delete a course (Teacher).....	44
Scenario 5: Create pages in a course (Teacher).....	44
Scenario 6: Add a challenge to a course (Teacher).....	45
Scenario 7: Delete a page from a course (Teacher).....	45
Scenario 8: Add a student to a course.(Teacher).....	45
Scenario 9: View a course and all of the people that are part of that course.(Student).....	45
Scenario 10: Solve a challenge inside of a course & see rank in course highscores.(Student).....	46
Appendix C: Front-End Diagrams.....	47
Appendix D: Meetings with Client.....	50
D.1 Meeting 1.....	50
D.2 Meeting 2.....	50
D.3 Meeting 3.....	50
D.4 Meeting 4.....	50
D.5 Meeting 5.....	50
D.6 Meeting 6.....	50
D.7 Meeting 7.....	51
D.8 Meeting 8.....	51

# 1. Introduction

Cybersecurity is a constantly growing field that is in need of skilled professionals who are able to protect against real-world digital threats and vulnerabilities. The “Twente Hacking Squad” was created for people that are part of the University of Twente, to enable them to safely work on and learn about practical hacking. People are able to complete capture the flag challenges in a realistic environment and be rewarded with points, based on the difficulty of the challenge.

Currently, the “Twente Hacking Squad” platform does not support course management for teachers and students, while the content management system (CMS) provided by the university, Canvas, does not allow teachers and students to complete and create these hacking challenges. The scope of our project is to transform the existing “Twente Hacking Squad” platform into a Canvas-like content management system in order for students to have the opportunity to learn about practical hacking in a more structured way. Teachers benefit from a much-improved user experience, allowing them to provide students with the learning tools that they need for a deeper understanding of both theoretical and practical cybersecurity skills.

The “Twente Hacking School” is a role based system which introduced a couple of new features, such as course management, migration of challenge management functionality from the Django admin panel into the web interface, as well as a modified version of highscore tracking for each course. Some things that are missing from what you would usually expect from a content management system are assignments, announcements and grading.

In Chapter 2 we delve into the domain analysis, exploring the context of the system, the stakeholder and current limitations within the platform. Chapter 3 moves on to the conceptual stage, we talk about how we defined the requirements and overall architecture. We explain design choices and we share how ideas were shaped. In Chapter 4, we turn the concepts into a working solution, explaining the system structure, technical details and development process. Chapter 5 presents the testing phases, including system, unit and user tests. We compare the test plan and test results and talk about possible software risk issues. In Chapter 6 we suggest further improvements, and we draw conclusions in Chapter 7.

## 1.1 Project Plan

For this project implementation, it was decided to use a straightforward three-phase plan. During the first phase, an analysis of the current system was conducted, goals were defined, and all requirements for this project were gathered. The focus of the second phase was on designing the system, leading to the creation of several diagrams and mockups. The final phase was the implementation of the system. Throughout all these phases, weekly meetings were held with the client to ensure we stayed on the right track. (see Appendix D)

## 2. Domain Analysis

This chapter provides an overview of the system's domain, including the current state, existing challenges, client vision, key stakeholders, and the software environment. Together, these elements offer a clear understanding of the context in which the system operates and the goals guiding its development.

### 2.1 General Knowledge of the Domain

The current version of the web application, “Twente Hacking Squad”, provides a platform where students can engage with various cybersecurity challenges. After logging in, students can attempt to solve different challenges and earn points for each successful solution. The platform also includes a high score leaderboard that highlights the top-performing students. Currently, challenge creation and editing are handled through the Django Admin interface.

### 2.2 Problem Statement

The existing version of Twente Hacking Squad lacks essential features for managing cybersecurity courses in an organized and accessible way. Currently, there is no course structure or role-based access system, which limits how information and challenges are delivered to students. This makes it difficult to present supporting materials alongside challenges, reducing the potential for deeper understanding and structured learning.

Additionally, because challenge creation and editing are only possible through the Django Admin interface, this is not ideal for non-technical users. This approach can be intimidating and inefficient for teaching staff who are unfamiliar with backend systems. As a result, the current system is not well-suited for scalable, user-friendly course management and requires improvement to better support both students and educators.

### 2.3 Client Vision and System Goals

During initial discussions, the client expressed the desire to enhance the usability and accessibility of the platform, both for the students and for the teaching staff. The main goal was to transform the web-application into a Content Management System (CMS) that supports courses, pages and challenges. This way, information about the studied topic can be presented in the course, and challenges can be easily grouped on pages, alongside textual descriptions and relevant information. Additionally, the client aimed to move away from relying on the Django Admin interface in favor of a more intuitive and user-friendly, custom-built alternative. Furthermore, a role-based system was required, to ensure that

only authorized users-such as teachers and administrators-can perform certain tasks. These tasks include creating or deleting courses, adding participants, and managing challenges. To reflect this shift in purpose and functionality, the improved version of the web application will be referred to as "Twente Hacking School."

## 2.4 Stakeholder Analysis

The platform involves a diverse group of stakeholders, each with a distinct role in its development and use. **Thijs van Ede** acts as both the supervisor and client, overseeing the project's direction and alignment with educational goals. **Jerre Starink**, the original developer and maintainer of the application, provides valuable technical insight into the existing system. The primary users are **students**, who engage with the platform by participating in challenges and following courses. **Teachers** use the system to create and manage courses, upload challenges, and monitor student progress. **Administrators** are responsible for managing user roles and maintaining overall system functionality. In addition to internal users, the platform also supports **cybersecurity enthusiasts** outside of academic settings who are interested in improving their skills. Lastly, the **development team**, including current contributors, plays a key role in maintaining and evolving the system to meet user needs and technical standards.

## 2.5 Software Environment

This section provides an overview of the current technology stack and tools used in the development of the system.

- **Django**: A Python-based backend web framework used to build the REST API. It handles all incoming requests from the frontend, manages communication with the database, and performs authorization tasks. Django is also used for testing purposes throughout the development process.
- **Vue.js**: A JavaScript frontend framework based on reusable components. It enables efficient and modular development of the user interface.
- **Docker**: Utilized for both development and deployment, Docker allows the team to start up containers for the frontend, backend, and database with a single command. This ensures consistency across different environments and simplifies the setup process.
- **PostgreSQL**: A reliable and robust SQL database used to store application data. Django connects to the PostgreSQL database to perform data operations.



## 3. Global Design Process

This section outlines our design choices and planning conducted before implementation began. It describes our initial brainstorming phase and requirement gathering, details our preliminary design choices, tools and selected frameworks. Additionally, it provides an overview of the system, explaining the system pages and overall functionalities.

### 3.1 Design Methodology

To manage the development of our product we decided to practice a product development approach named “Agile design methodology”. This method emphasizes flexibility, customer feedback and effective group communication. Weekly group meetings (see Appendix D) were conducted to ensure good group communication and coordination. This development approach is client-centric, therefore numerous meetings took place to get their feedback.

To apply this method some roles related to specific management tasks were assigned to group members.

#### **Contact person** - Tim Wijma

Tim is the main point of contact and he will handle the communication with the stakeholders via email.

#### **Organizational coordinator** - Jaridza Tromp

Jaridza is responsible for keeping track of organizational issues such as taking meeting notes, monitoring deadlines, and facilitating coordination within the team.

#### **Scrum master** - Role rotation

Since we follow the Agile method, the Scrum master role will rotate weekly between all team members. The Scrum master is responsible for coordinating the stand up meetings, keeping track of each member’s individual progress and identifying challenges they might be facing.

### 3.2 Initial Concepts and Brainstorming

The “Twente Hacking School” was introduced to the development team as a content management system designed to provide students with a secure environment to practice and develop their hacking skills. The platform challenges students through structured Capture-the-Flag exercises.

Drawing inspiration from the content management system "Canvas", the platform's architecture was organized around the concept of courses. Each course is composed of multiple pages, which may contain interactive hacking challenges.

A role-based access control system regulates the generation and accessibility of content. Administrators are responsible for creating courses, teachers can populate them with educational material and challenges, and students enrolled in a given course are granted access to its content. Only users enrolled in a specific course will be able to view and attempt its challenges.

### 3.3 Requirement Gathering

After analyzing the initial idea for the project we conducted an analysis on the requirements that stakeholders would be interested to see implemented in the “Twente Hacking School” platform.

These requirements have been divided into functional and non-functional, focusing individually on each stakeholder in each section. The *MoSCoW* (*must-have, should-have, could-have and won't-have*) prioritization technique was implemented to categorize the requirements based on their priority level. Each requirement has been written according to the SMART standards, ensuring that any requirement written is specific, measurable, attainable, relevant, and timely to ensure feasibility of these.

#### **Functional requirements:**

##### **Must have**

1. As an admin, I must be able to assign roles of teacher/student.
2. As an admin, I must be able to create a new course.
3. As a teacher, I must be able to edit all the courses I am part of.
4. As a teacher, I must be able to delete courses that I am part of.
5. As a teacher, I must be able to view all of the courses that I am part of.
6. As a teacher, I must be able to create a new page within the courses I am part of.
7. As a teacher, I must be able to add a text description to a page.
8. As a teacher, I must be able to add challenges on a page to courses I am part of.
9. As a teacher, I must be able to modify challenges I have rights to.
10. As a teacher, I must be able to edit the pages within courses that I am part of.
11. As a teacher, I must be able to delete pages in the courses that I am part of.
12. As a teacher, I must be able to view all challenges.
13. As a teacher, I must be able to delete a challenge that I have rights to.
14. As a teacher, I must be able to add students to courses that I am part of.
15. As a student, I must be able to see the courses I am enrolled in.
16. As a student, I must be able to see the pages of a course I am enrolled in.

17. As a user, I must be able to see the highscores of a course I am part of.
18. As a student, I must be able to see the other people enrolled in a course I am enrolled in.
19. As a student, I must be able to do a challenge inside a course I am enrolled in.

### **Should have**

1. As a teacher, I should be able to add another teacher to any of the courses I am part of.
2. As a teacher, I should be able to use markdown in the text field in pages and upload images, videos, etc.
3. As a teacher, I should be able to add a new challenge through a form (separate page) instead of through the Django admin dashboard.

### **Could have**

1. As a teacher, I could create pages with interleaved challenges and descriptions (similar to a Python Notebook).
2. As a teacher, I could be able to reorder the pages within the courses I am a part of.
3. As a teacher, I could be notified when a modification (edit/delete) of a challenge that I am using in one of the courses that I am part of occurs.
4. As a teacher, I could be notified when a challenge I am deleting is being used by another course.

### **Won't have**

1. As a teacher, I won't be able to see the history of edits of my course.
2. As a teacher, I won't be able to create challenges through the django dashboard.

## **Non-Functional requirements:**

### **Performance**

1. As a user, I want the website to load within 1 second.

### **Security**

1. As a user, I want my data to be safely stored and only accessible by authorized people.
2. As a developer, I want the input to be sanitized in the front- and backend to prevent XSS and SQL injection.
3. As a developer, the backend should validate API requests and only send the requested data to authorised parties.

### **Usability**

1. As a user, I want the application to have an intuitive interface that requires no more than 5 minutes of training for new users.

### **Reliability**

1. As a user, I want the application to be consistently available with minimal disruptions, with an uptime of 99.9%.

### **Scalability**

1. As a developer, I want the system to handle a growing number of students, teachers, and courses without performance issues, so that we can scale to meet future demand.

### **Portability**

1. As a user, I want to be able to access the hacking school dashboard through any of my devices.
2. As a user, I want the system to be compatible with all major browsers.
3. As a user, the website should be responsive for all sized devices.

### **Compatibility**

1. As a teacher or admin, I want to upload challenges as Docker containers so that I can ensure they run in a consistent and isolated environment.

### **Availability**

1. As a user, I want the system to be available 24 hours a day.
2. As a user, I want the system to recover after a crash with no data loss.

### **Maintainability**

1. As a developer, I want the code base to be maintainable.

The list of requirements was presented to the clients to ensure that our interpretation of the system's design aligned with their desired concept. A use-case diagram was created to visually represent the requirements, making it easier for the clients to understand and validate the proposed design.

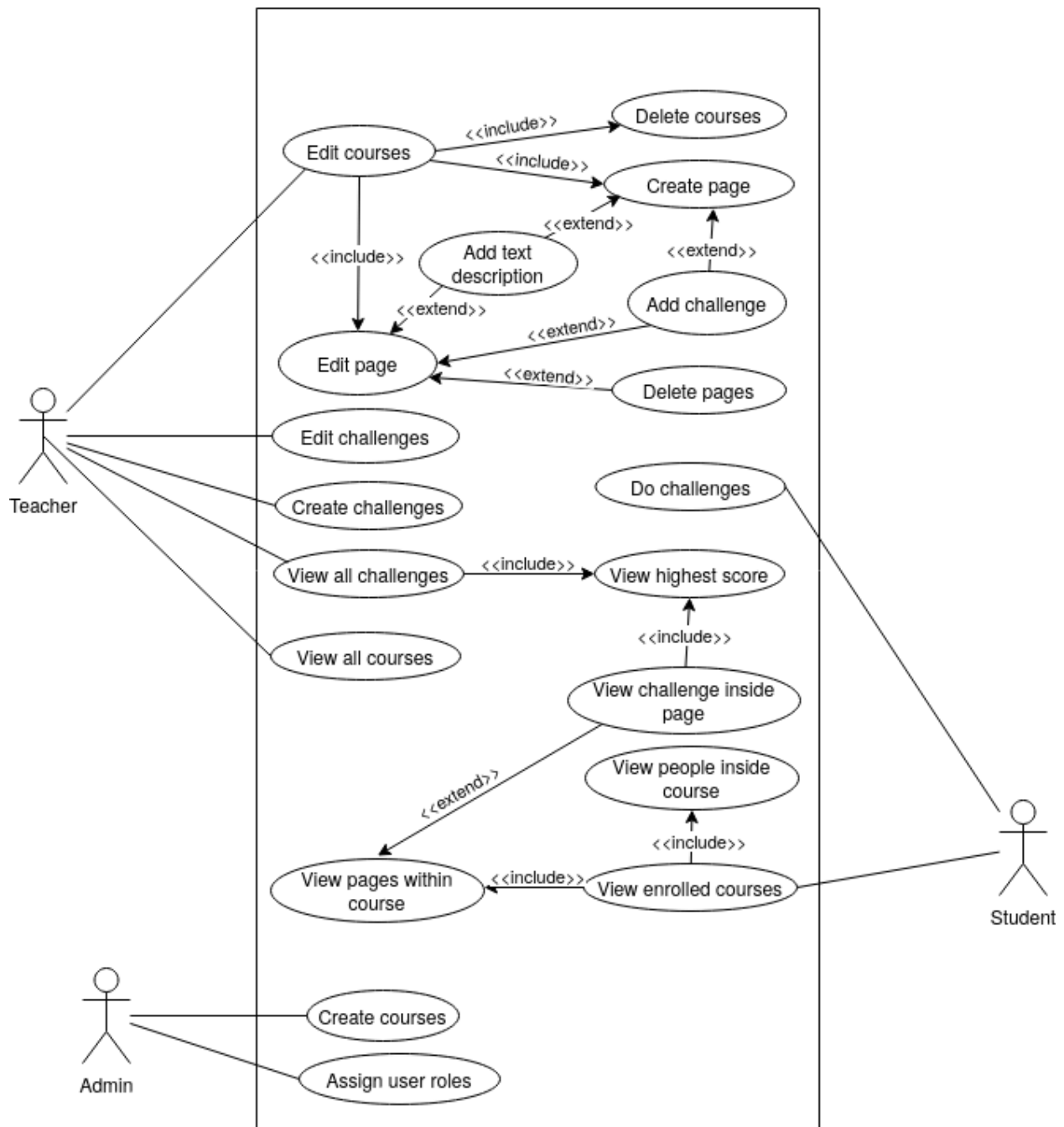


Figure 3.1: Twente Hacking School Use-Case Diagram

## 3.4 Mock-ups

After gathering all the requirements, these were used to create initial designs. These designs incorporated the requirements formulated in the earlier section (3.2). The mock-ups (see Appendix A) were presented to the client, and the client provided feedback, suggesting minor UI adjustments. Ultimately, the client was satisfied with the overall design direction.

## 3.5 Preliminary Design Choices

As there was already an existing application that we had to build on top of, many design choices, such as programming languages, frameworks, and database, had already been made. Although most project defining design choices had been made for us, we still had a few decisions to make and ensure they are compatible with the current application.

### 3.5.1 Tools and Software used

As mentioned in Section 3.4, before we started developing the content management system, we created a mock up of what the system should look like. We used **Figma** for this, a collaborative design tool used to easily create wireframes and prototypes. It helped us define the layout and structure of the frontend before implementation began.

During the development of the system we added the following *npm* packages:

- **Vuetify**. This is an exhaustive component library made specifically for vue. By using this component library all components automatically look uniform, without having to do any styling ourselves. It follows the design standards set by Google's *Material Design*, which ensures all components are also accessible for users with impairments.
- **Vuedraggable**. This package gives access to an easy to use drag-and-drop component, which we use to reorganize the text and challenge fields inside a page.

### 3.5.2 Architectural design choices

For the content management system not many additional architectural design choices had to be made, as most infrastructure and layout had already been made for us. However, there were a few important decisions to be made.

#### 3.5.2.1 Role system

First of all, the current system uses django's default role system, which distinguishes only between regular and staff users. For the content management system we needed to have more flexibility than what django offered, as we needed the role "Student", "Teacher", and "Admin".

To implement this we created a new **UserProfile** class. This is a class that simply connects a user and their role. Using django signals, we ensure that whenever a **User** object is created, it will automatically create a corresponding **UserProfile**.

With this role system in place, we needed to restrict access to certain pages based on the user's role. On the frontend, we implemented two methods in *router.js*:

*redirectUnauthorizedAdminOnly* and *redirectUnauthorizedTeacherAdminOnly*. Routes that are restricted to admins or teachers call these methods before navigation. If the user does not have the required role, they are redirected to a 404 page.

A similar check is performed on the backend: the role of the user is verified when a request is made, and if the user is not authorized, an error is returned.

### 3.5.2.2 Course shell

Another architectural decision we faced was how to show the course environment. Every page inside a course (home, pages, highscores, etc), needs to display both the sidebar and the navbar with the course title. Initially, we considered copying the sidebar component into every page that required it, but we quickly realized there had to be a better approach.

Next, we tried using 2 separate routers, one for routes with the sidebar and one without. In *router.js*, the routes with the sidebar would import the sidebar and display it. However, this approach was not ideal, as we had to import the sidebar individually for each route.

When we decided to also show the course title on every course page, we restricted the sidebar logic entirely. We created a new component *CourseShell*, which includes both the sidebar and the course title. Instead of importing the sidebar in each route, we defined *CourseShell* as a single parent route, with the other routes being nested routes inside *CourseShell*. This way, all nested routes are automatically rendered inside the shell, without having to specify this per route.

## 3.6 System Overview

Now that requirements have been defined and the overall system structure established, this section details the various pages within the system, along with the functionalities each page will offer.

### 3.6.1 Courses Dashboard Page

The Courses Dashboard Page displays all the courses a user is a part of. Courses are listed in a grid layout, making it easy for users to browse and select the course they wish to access. Each course is represented by a course card that includes details, such as the course title and a background image.

Additionally, the page supports a search functionality that allows users to quickly find a specific course by name. Depending on a user's role, additional functionalities become available: teachers can delete and edit a course they are part of, while administrators have the ability to create a new course.

### 3.6.2 Course Homepage

After selecting a specific course from the course dashboard, users are redirected to the course's homepage. This page provides an overview of the selected course, displaying information such as the course's title and any additional descriptions provided by the teacher. The main components of the course are accessible via a sidebar, which includes sections such as pages, people, and highscores. This allows users to navigate the course components with ease. Teachers and administrators have the additional functionality of course editing.

### 3.6.3 Course Pages Overview Page

This page is found within a course and displays all the pages that have been added to it. Users may use it to navigate to any page in the course. Teachers and administrators additionally have the possibility to create, edit, or delete a page.

### 3.6.4 Course Page Page

After selecting a specific page from either the *Pages Overview* or the sidebar within a course, users are redirected to the selected page. A page may contain both textual information and challenges. Users can interact with and solve these challenges directly on the page. Teachers and administrators additionally have the right to edit or delete the page.

### 3.6.5 Course Users Page

The Course Users Page displays a list of all users enrolled in a specific course. Users are presented in a table format, which supports both searching and filtering functionality, making it easy to locate any specific user or view users based on their roles. Additionally, teachers and administrators are able to add and remove users from the course. The user addition form also supports searching and filtering functionality. Further, it allows the addition of multiple users at once, removing the need to add them individually.

### 3.6.6 Course Highscores Page

The Course Highscores Page displays the highscores of all users enrolled in a specific course. While similar in structure to the previously implemented global highscores page, this version is limited to the users enrolled within the course itself. Clicking on a user's name in the high scores list redirects to a user's course specific profile. This page is also a course-specific version of a



user's global profile and only displays data and statistics related to challenges completed within the course.

### 3.6.7 Challenges Page

The Challenges Page was previously implemented and has been slightly modified. Depending on the user's role, the challenges listed on the page will differ: students will only see challenges from courses they are a part of, teachers will see challenges from courses they are a part of or are authors of and administrators will see all challenges in the system. A "Create challenge" button has been added, allowing teachers or administrators to easily add new challenges to the system. Challenges can now be edited or deleted by their author or by an administrator through this page. Additionally, challenges that are not marked as visible are still displayed to their authors and to administrators.

### 3.6.8 Challenge Creation and Editing Page

This page allows for teachers or administrators to create or edit a challenge. It includes a form that enables the users to input all necessary details for a challenge. When editing a challenge, the form is prefilled with the current data and validation is implemented to ensure all required fields are completed before submission.

### 3.6.9 Users Page

The Users Page is an administrator-only page that provides an overview of all users registered in the system. Users are displayed in a list format, which supports search and filtering functionalities, making it easy to locate a specific user or view users based on their roles. Additionally, administrators are able to change the role of a user.

## 3.7 Activity Diagrams

Activity diagrams have been developed to offer a visual description on how the teacher should interact with the Twente Hacking School interface to perform some specific tasks. These diagrams could be used for training purposes to teach how the system works.

### 3.7.1 Creating a New Challenge

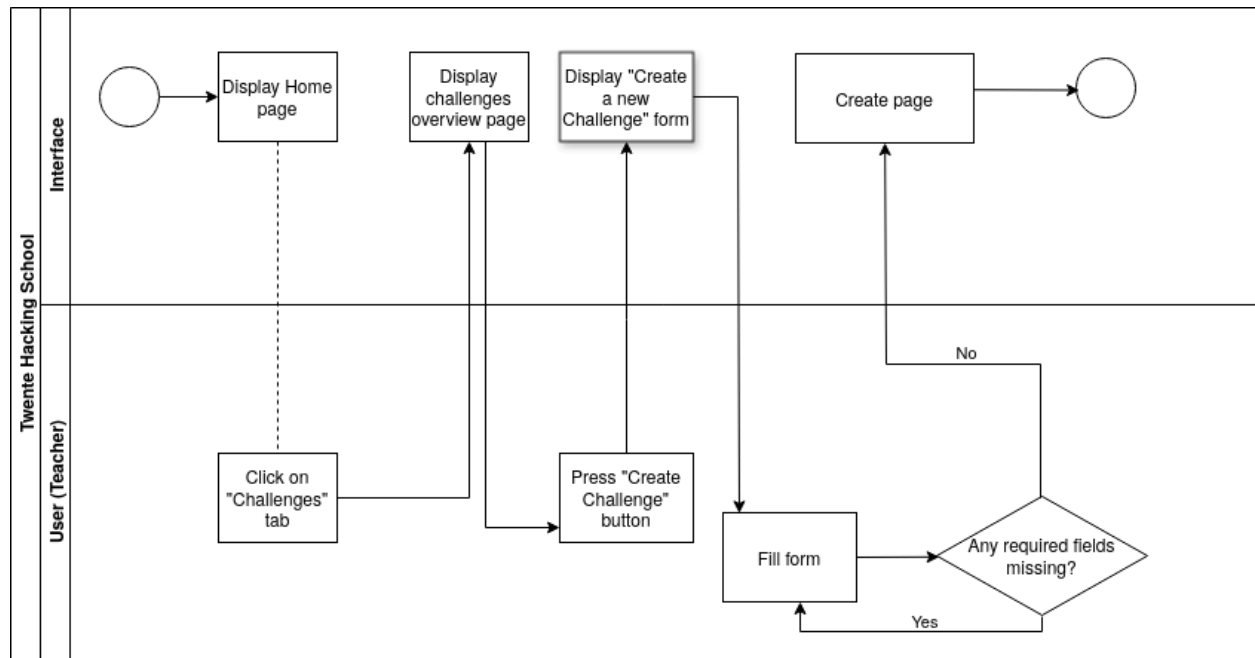


Figure 3.2: Challenge creation activity diagram

From the home page, users click on the "Challenges" tab to view existing challenges. By pressing the "Create Challenge" button, they access a form where they can input challenge details. Similar to other creation processes, the system validates that all required fields are completed before allowing the challenge to be created.

### 3.7.2 Creating a New Course

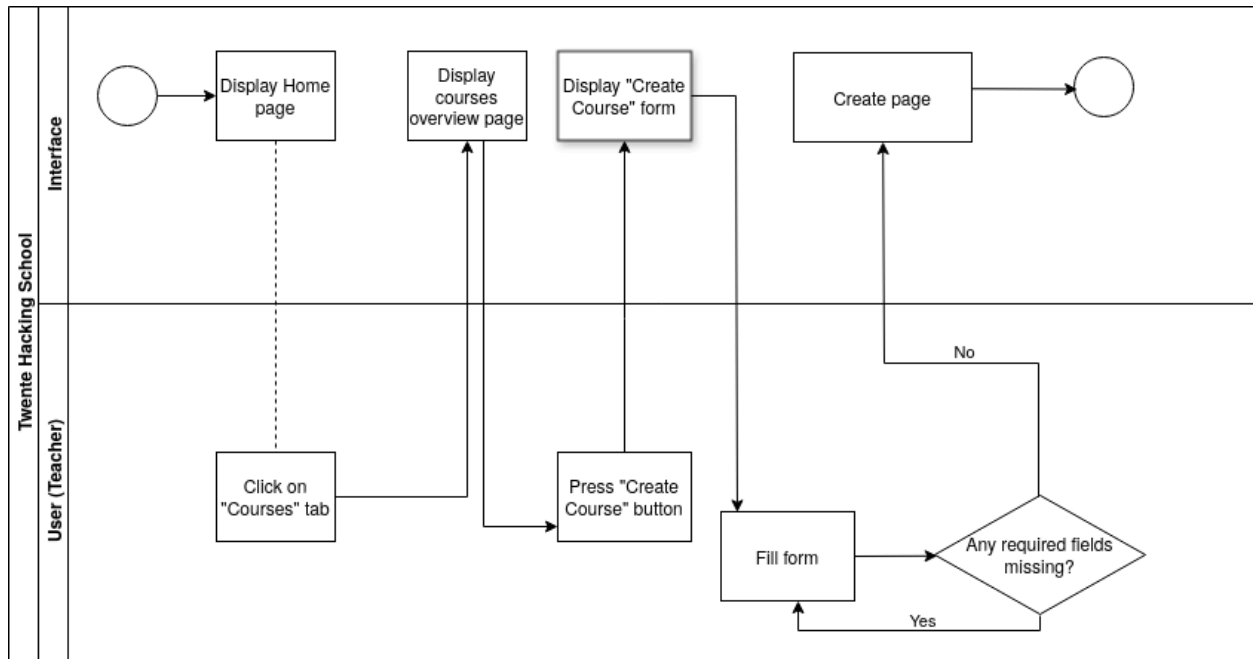


Figure 3.3: Course creation activity diagram

Starting at the home page, users click on the "Courses" tab to view all courses. By pressing the "Create Course" button, they access a form to input course details. The system validates that all required fields are completed before allowing course creation. If fields are missing, users return to the form to complete it; otherwise, the new course is created.

### 3.7.3 Adding Pages to a Course

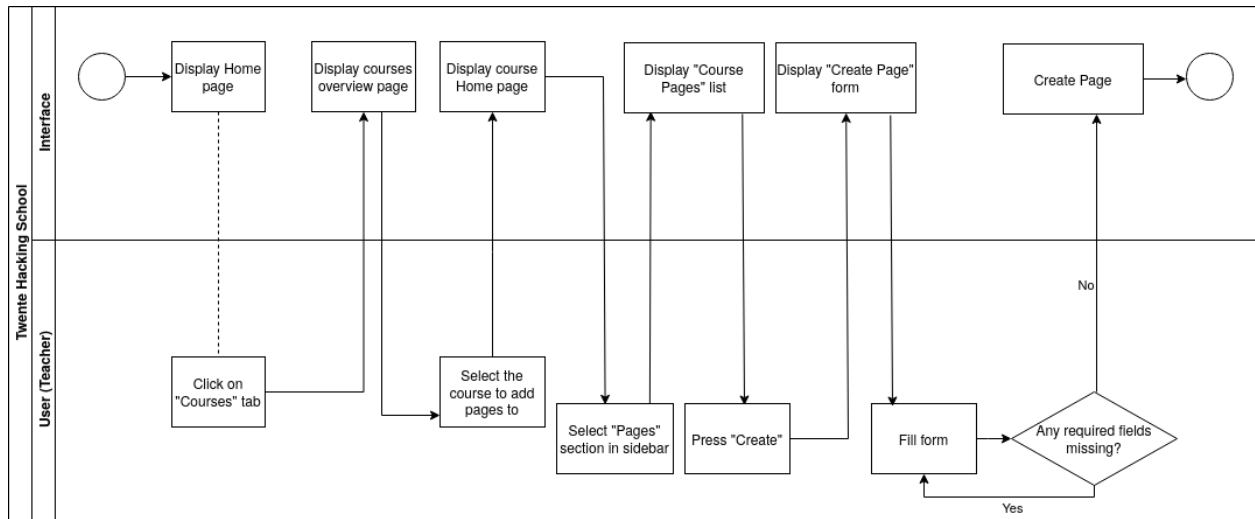


Figure 3.3: Page creation activity diagram

Users begin at the home page and click the "Courses" tab to see available courses. After selecting a specific course, they navigate to the "Pages" section in the sidebar. From there, they press "Create" to access the page creation form. The system checks for any missing required fields, allowing creation only when all necessary information is provided.

### 3.7.4 Adding Users to a Course

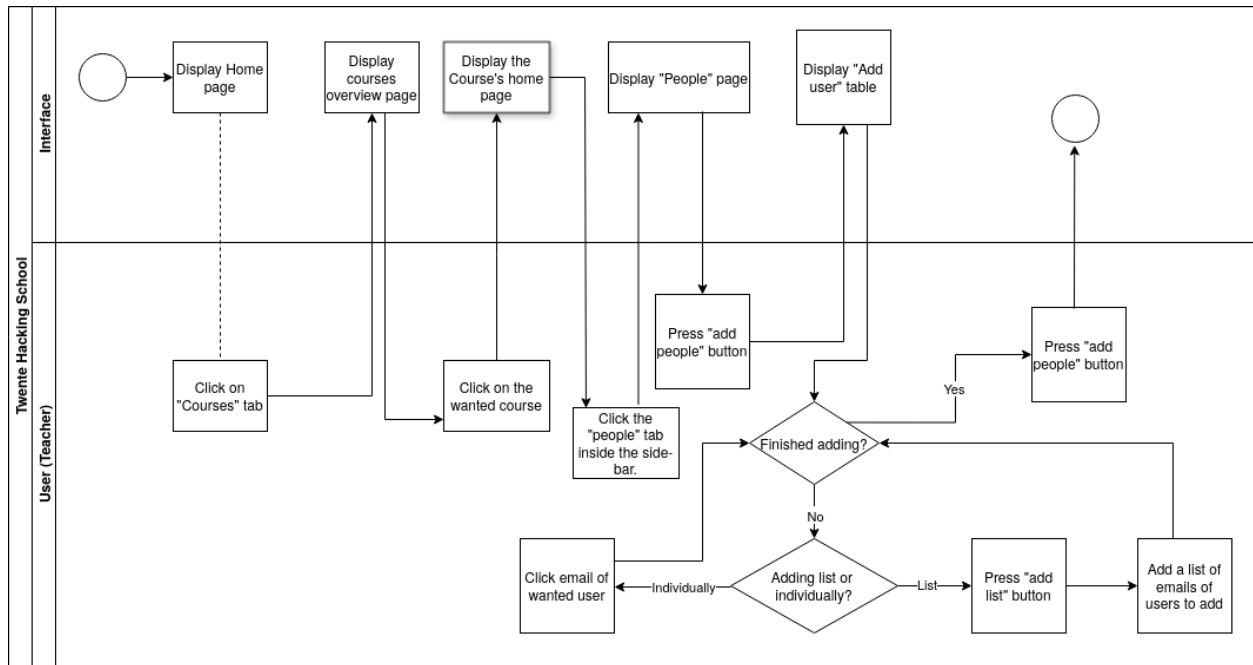


Figure 3.4: Course users management activity diagram

From the home page, users navigate to the courses overview by clicking the "Courses" tab. After selecting the desired course, they access the people management section by clicking the "People" tab in the sidebar. Users can add people either individually (by clicking on specific emails) or in bulk (by pressing the "add list" button and uploading a list of emails). The interface provides a decision point to continue adding users or finish the process.

## 4. Implementation & Development

In this section, we detail how we transitioned from our global design to a functioning system. We will provide and explain diagrams such as the Entity-Relationship Diagram (ERD) and front-end schematics that provide an overview of the system's structure. Additionally, we discuss the design challenges encountered during development.

### 4.1 System Description

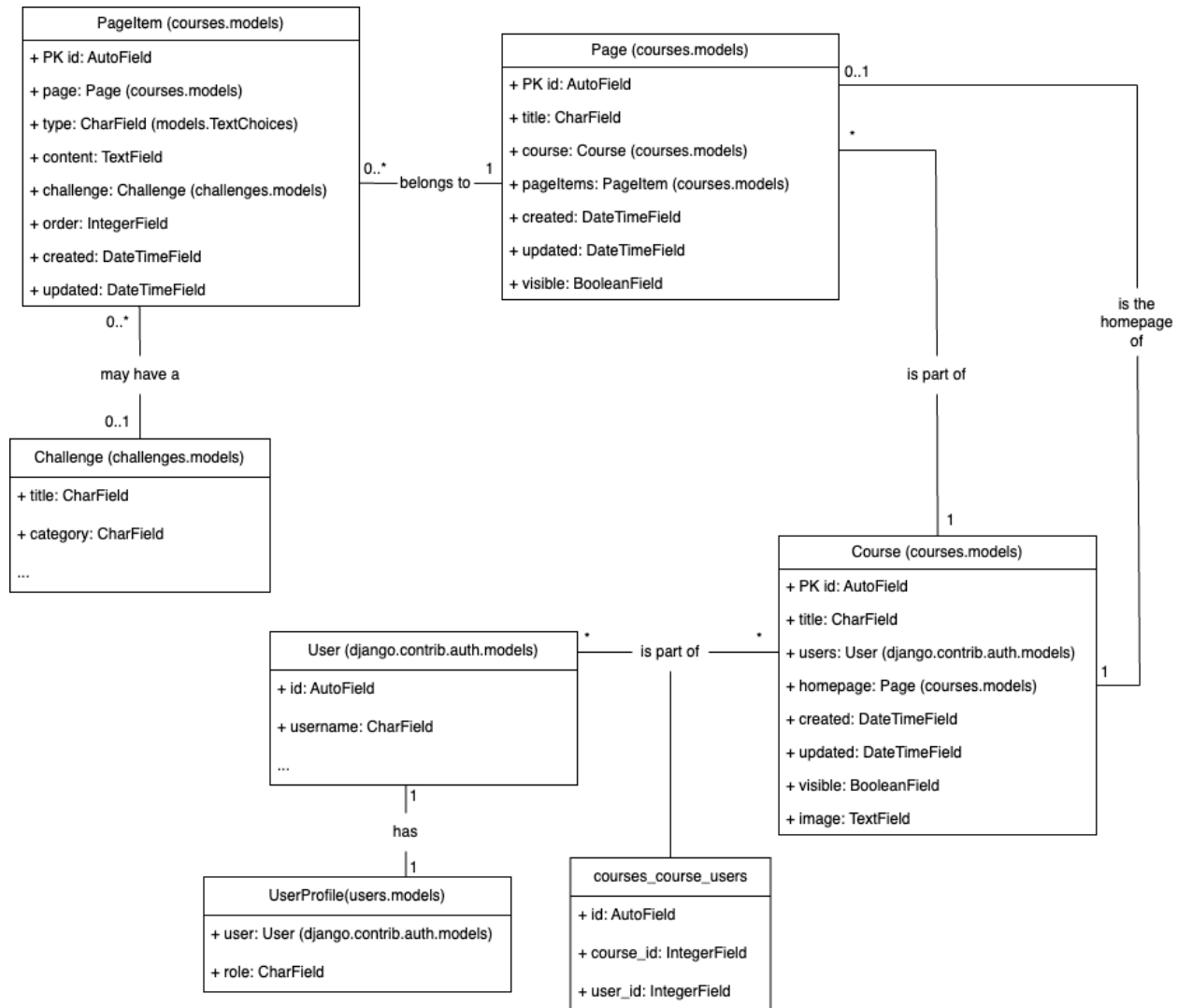


Figure 4.1: ERD representing models made for this project.

The *Course* model represents a course in the Twente Hacking School. This table includes several attributes and relationships: a course has a title, users, a homepage, timestamps (date & time) to indicate when it was created and last updated, a “visible” flag to represent whether the course is

visible to the students who are part of this course and an associated image (displayed on the course card in the dashboard). The `users` field defines a many-to-many relationship with Django's native `User` model, allowing multiple users to enroll in multiple courses. Additionally, the `homepage` field is a foreign key to the *Page* model, assigning one specific page as the course's homepage.

The *User* model, provided by Django's native built-in authentication system, was not developed as part of this project and it remains unmodified in our implementation. The model *UserProfile* represents a profile for each user, it stores additional information of each user, namely the role of a user. In our implementation, a user can be either an "admin", a "teacher", or a "student". Depending on their role, a user may be limited in the actions they are able to perform. A teacher, for instance, may only create pages within a course they are part of, modify course information, and enroll students into a course. A student may only see courses they are enrolled in and solve challenges within those courses, while an admin is able to perform all actions and view all information.

Although the model doesn't explicitly include a "pages" field, the one-to-many relationship from *Page* to *Course* lets you access all pages associated with a course.

The *Page* model represents a page that is created within a course. This model includes several attributes and relationships: each page has a title and a course field which is linked via a foreign key to a specific *Course* model. The page model includes `pageItems` which references the *pageItem* model, and the model includes timestamps to indicate when the page was created and last updated. Lastly, each page has a "visible" flag which represents whether the page is visible to the students who are part of this course.

The *PageItem* model represents an item within a Page. This model includes:

- **Page:** Linked via a foreign key to the page that this item belongs to. This established a many-to-one relationship: a single page can have multiple `pageItems`.
- **Type:** Indicating the type of content the item holds (e.g, text, challenge).
- **Content:** A text field used for storing the content of a `pageItem` of type 'text'.
- **Challenge:** Represented by the *Challenge* model, which was previously implemented and remains unmodified in this project. The foreign key to the *Challenge* model is only used if the `pageItem` is referencing a challenge instead of plain text. This means multiple `pageItems` can reference the same Challenge (one-to-many relationship from Challenge to `PageItem`)
- **Order:** This determines the item's position on the page, allowing change to the order in which the items appear to the user.
- **Created and Updated:** Timestamps to indicate when the `pageItem` was created and last updated.

## 4.2 Front-End Diagram

Given that the front-end is built using Vue.js, which is component-based, we created a diagram that contributes to both efficient development and the long-term maintainability of the project. It serves as a useful tool for organizing components logically.. From the outset, we iterated through several versions of the front-end design, refining and improving it over time.

All design variations can be found in the Appendix C, where the evolution of the front-end diagram is clearly illustrated. Notably, there is a significant difference between the initial and final version, reflecting the improvements made throughout the project. One of the main challenges we encountered was determining the best way to implement the Navigation Bar within the course view. This issue is evident in the progression of the designs. Additionally, we focused on maintaining a clear separation of components by structuring them to be as modular and reusable as possible, for both clarity and maintainability.

## 4.3 Design Challenges and Solutions

This section will briefly mention some challenges we had during the design of the system, and how we solved these challenges.

### 4.3.1 Role permissions

A challenge we had throughout the whole development was which roles should have access to what. For example, whether a teacher should be able to create a course or not. However, these were always minor challenges and were easily resolved by mentioning them during the next meeting with the client.

### 4.3.2 Challenges in a course

Another challenge we encountered was whether a teacher could use challenges created by another user in their own courses. If the original creator would edit their challenge, it would also be changed in the course containing that challenge, is that the behaviour we want?

In coordination with the client we had the idea to make it possible to “clone” challenges. If you want to use a challenge in your own course, you can just duplicate it, and edit the challenge as desired. Another problem this solves is if a challenge would get deleted, courses with this challenge would have the challenge deleted as well, which can lead to unintended consequences.

However, some complications arose when we thought a bit more about it. Highscores would no longer be fair: if a challenge would be duplicated multiple times, you could solve the original challenge and its clones, and get more points than you originally would have.

When the original challenge gets edited for a valid reason, like a typo or mistake, the cloned challenges wouldn't have these updates, leading to outdated versions of the same challenges.



It would also clutter the challenges list, which would make it harder to add challenges to a page, and make the global challenges view container duplicates.

Because of these complications, we decided to stick to our original implementation. Teachers can use any challenge that is set to visible, no matter who the owner is. The challenge does not get duplicated, meaning if the original owner edits their challenge, it would also be changed in all courses containing this challenge.

#### 4.3.3 Highscores in a course

We also encountered some problems figuring out how to do the highscores of a course. As challenge attempts just include the user id and challenge id, it was difficult to calculate the highscore of a user from only the challenges *inside* a course.

We thought about including the course id in the challenge attempts as well, then we could just filter by course id and get the highscores like that. However, if you have a challenge which is used in multiple courses, solving the challenge in one course should also mark it as solved in the others. This complicated calculating the highscores in a course a little bit.

We solved this by getting a list of all challenges which appear in a page of the course, and calculating the highscores from this list. This way, challenge attempts are still global, and doing a challenge inside a course would complete it for all courses.

## 5. Testing

This section outlines our testing plan and the testing results. It details the various functionalities to be tested and explains our testing approach. We also provide the results from the unit testing, integration testing, system testing and usability testing. Additionally, any design choices or bugs that were found and solved based on the results of the tests are also discussed.

### 5.1 Testing Strategy

#### 5.1.1 Approach

We focus on writing comprehensive tests to ensure maximum coverage for the newly added API endpoints. Pre-existing functionalities and endpoints have not been covered by us, as they were already tested by the previous developers. We have four test strategies: Unit Testing, System Testing, User testing and Penetration Testing.

#### 5.1.2 Unit Testing

Unit tests will be conducted as automated tests which use the Django unit test framework. We are testing all the API endpoints and ensuring the role based system is in place, ensuring that each user role has access only to the appropriate functionalities.

#### 5.1.3 System Testing

Each member of the team will perform system tests to check all the newly implemented functionalities, to ensure the work flow. We will also test edge cases and unusual user behaviour to ensure that the system responds correctly under unexpected conditions.

#### 5.1.4 User Testing (Usability Testing)

We will conduct a round of User Testing to evaluate how users react to the platform and how easily they complete common tasks such as: creating a challenge, editing a course, creating pages, adding people to a course. We will gather feedback to identify any usability issues, misleading interfaces or features that could be implemented. This is to ensure that the system is user friendly and intuitive.

#### 5.1.5 Penetration Testing

Penetration testing will be conducted to identify potential security vulnerabilities within the platform. We will be performing real-world attack scenarios to evaluate how well the system can withstand unauthorized access and other threats.

## 5.2 Software risk issues

There are multiple areas in our project that could represent a risk and that have to be thoroughly tested. Since we have created a role-based system, it is very important to test whether the features that should only be accessible by a user with a certain role can be accessed by users with different roles. For example, a student should not be able to manage a course, add people, pages or change challenges.

To address this, because this is one of the main risks of the system, we decided to write unit tests for testing the API endpoints on the back-end. We performed system tests to ensure that users would not have access to unauthorized pages. For mitigating this risk, we made sure to enforce the role-based authorization by performing checks on both the frontend and the backend.

We also identified a risk that currently does not pose a problem to the system, but it might in the future, so we decided to mention it here. Because of how the challenges are displayed in the original code we received, html code that teachers might write in two fields of the challenges does get rendered in the frontend (the Description and the Hint fields). This is not a problem, as scripts do get removed and variables are not available in the html, but a user can still change the appearance of the frontend if they wanted, the future developers should keep this in mind when further working on the system and possibly implementing features such as enabling students to create their own challenges.

## 5.3 Functionalities to be tested

In this section, we will list the main newly implemented functionalities that need to be tested, as can be derived from the list of extracted requirements. To ensure the platform functions as intended, the following functionalities will be tested using a combination of unit tests, system tests and user tests. We will categorize the features based on the level of risk, using High (H) or Low (L). The level of risk indicates how critical testing a certain functionality is.

Functionality to be tested (should only be done by)	Level of Risk
Create a course (admin)	H
Edit a course (teacher/admin)	H
Delete a course (admin)	H
See details of a course (teacher/admin/student)	H
Create a challenge (teacher/admin)	H
Edit a challenge (teacher/admin)	H

Delete a challenge (teacher/admin)	H
Change the role of a user (admin)	H
Add a user to a course (admin/teacher)	H
Remove a user from a course (admin/teacher)	H
Create a page in a course (admin/teacher)	H
Edit a page in a course (admin/teacher)	H
Delete a page in a course (admin/teacher)	H
See all the existing users (admin)	L
See all users in a course (teacher/admin/student)	L
See highscores in a course (teacher/admin/student)	L

For the functionalities that relate to a course, we assume that the teacher/student is part of that course.

## 5.4 Pass/Fail Criteria

For all high risk tests, all the tests must pass without any issues. For checking the right API endpoints, a passed test is considered when the expected status code is returned, and the outputs in the json response are correct. Failed tests will result in the feature not being included in the final product.

## 5.5 Schedule

We used an agile technique when working on the project, which led to the requirements being added and changed during the development phase. The unit tests were made after we created all the necessary API endpoints. The system tests were thoroughly conducted throughout the implementation of each feature. The user testing session took place in the 8th week of the module, when the majority of features were in place, but still having some time to make necessary adjustments. The penetration tests were conducted towards the end of the project.

## 5.6 Test Results

### 5.6.1.1 Unit tests

All API endpoints have been tested to assert the correct HTTP status response for all the features that were mentioned in the table above. We discovered a bug, when editing a challenge and the role was admin, the access was not authorized because they were not the author of the challenge. That had to be changed since the admin should be able to edit any challenge.

### 5.6.1.2 System Tests

All the functionalities of the frontend/backend and interaction of the system were thoroughly tested, with rounds of bug fixing each time an issue was discovered.

### 5.6.1.3 User Tests

The user tests have provided a great amount of useful feedback which we then used to make the system more user-friendly and fix a few bugs. To conduct the user tests, we worked with some of the university team that might be using the system that we have created. We asked them to go through a couple of realistic scenarios that they will have to go through when using the application. A complete list of the scenarios can be found in Appendix B. After the meeting we made a list of areas of improvement and transformed them into specific tasks that had to be implemented:

1. **Problem:** The users didn't know which fields were mandatory when creating a challenge.  
**Solution:** We added a star for the fields that were mandatory.
2. **Problem:** The add people button wasn't clear enough in the modal where you add people to a course.  
**Solution:** We made the Add People button in the popup filled instead of outlined.
3. **Problem:** Users couldn't remove the Download field from a challenge unless a file was uploaded first.  
**Solution:** We added a delete button when uploading a file, allowing users to remove the download field without any extra steps.
4. **Problem:** The Official checkbox was unnecessary, as there was no differentiation in the backend between an official and unofficial challenge.  
**Solution:** We removed the Official checkbox.

5. **Problem:** The Vagrant option was not supported, but there was a button to specify that the challenge was using Vagrant.  
**Solution:** We removed the Vagrant checkbox.
6. **Problem:** A challenge was not visible by default when creating it, and it was counter intuitive.  
**Solution:** We made the Visible checkbox true by default when creating a challenge.
7. **Problem:** When scrolling in the score field, the score could get negative.  
**Solution:** We fixed the score field to not make it possible for the score to go below 0.
8. **Problem:** There was no upper limit to the score field.  
**Solution:** We set an upper integer limit to the score field.
9. **Problem:** It was possible to use characters like “+”, “-”, “e” inside the number fields.  
**Solution:** We restricted the number fields to only allow numeric values.
10. **Problem:** The deletion of a course was too easy and mistakenly removing a course was a concern for the users.  
**Solution:** We made the users type the name of the course that they want to delete as an additional confirmation.
11. **Problem:** The users suggested not to make the emails visible to other students for security reasons.  
**Solution:** We hid the emails of users in course members.
12. **Problem:** The users did not find it intuitive that they did not have the option to delete and edit a page inside the page itself.  
**Solution:** We made these features also available inside the pages.
13. **Problem:** The users said that the challenge should not be able to be deleted/edited if it is inside a page.  
**Solution:** We hid the three-dot menu on the challenge object inside the page.
14. **Problem:** The users found it unintuitive that you can not edit a course from inside the course.  
**Solution:** We added the edit course function inside the course itself.
15. **Problem:** When adding a challenge to a page there was no search functionality making it difficult to find the challenge.

**Solution:** We added search functionality to the challenge selector.

**16. Problem:** The Add People button was not fixed at the bottom of the page and the users could not find it easily when adding people to a course.

**Solution:** We made the Add People button fixed at the bottom of the modal.

**17. Problem:** The Not Visible eye icon appeared before the page had fully loaded, which confused users.

**Solution:** We hid the Not Visible eye icon until the page finishes loading.

## 6. Future Planning

In this section, we discuss the future planning for the content management system. It includes features that were originally planned but not implemented due to time constraints, as well as additional functionality identified during development that would improve the system in the future. As we will no longer be developing the application, this section also outlines how the content management system can be extended, maintained, and improved upon by the current, and new developers.

### 6.1 Features to be implemented

There are a number of features we did not implement because we ran out of time, and a few features that were not on our list, but would improve the system for future iterations.

#### 6.1.1 Responsive design

As the system is meant for doing hacking challenges, it is mostly used on desktop devices. Because of this, we decided to focus on making more features instead of making the application responsive for smaller screens. Making the content management system mobile friendly should not be too difficult, it is mostly the sidebar that makes it difficult to use on smaller screens.

#### 6.1.2 More options inside a page

An item inside a page currently only supports text and challenges. Adding support for more options inside a page would allow for a lot more customization. Possible options could be:

- Image, video or audio. A simple item that would display an image, video or audio inside the page
- Question field. Having the ability to create a multiple choice question would make a page more interactive and can test the users knowledge.

#### 6.1.3 Course, page, challenge and hint reordering

Currently, drag and drop functionality only exists for text and challenge items *inside* a page. However, this functionality could be applied in many places, for example:

- Pages inside a course. Teachers of a course would be able to reorder the list of pages however they want. The list of pages is currently just sorted by their creation date, making it difficult to add a page to the middle of the course at a later time.
- Course dashboard. Users could be able to reorder their enlisted courses on the dashboard, similar to canvas, for more user customization.
- Challenge list. The challenge list outside of a course is sorted by categories, and then by creation date. Allowing the admin to reorder the challenge on this global list would be a nice to have feature.



- Challenge hints. The hints of a challenge already have an “Order” field, but when saving a challenge in the newly created challenge page, it just saves the hints in the order they were added.

#### 6.1.4 Email notifications

There is currently no way to know whether you have been added to a course, other than manually checking your dashboard. By adding email notifications, users could be notified when they are added/removed to a course, making it a lot more user friendly.

This functionality could be extended to multiple places, such as:

- Teachers receiving an email when someone else changes their course, page or challenge
- Users being notified of their role being changed by an admin
- Someone overtaking you in the leaderboard, encouraging them to take it back

#### 6.1.5 History of edits

One of our “won't” requirements was having a history of edits of a course. Getting insight into the changes to your course would be a nice quality of life for the teachers and admins. This functionality could also be extended to pages and challenges for even more insight.

#### 6.1.6 “Vuetifying” previous components/pages

The current developers have expressed they really like the look of the Vuetify components, and have already questioned if they should convert the old system to use Vuetify. This would be a great way to make the old and new system look more uniform, and improve the usability of the old system a lot, as it is currently difficult to navigate without a mouse, which could cause problems for impaired users, such as those using a screen reader.

### 6.2 Future development

Of course, for the future maintainers to be able to implement these suggested features, it must be clear how to do so in the newly developed system. Most code we have made should speak for itself and should not be difficult to maintain.

#### 6.2.1 User roles

One of the most important things we have added is the role system. Users can be either “Student”, “Teacher” or “Admin”.

It is not possible to make users an Admin inside the system itself, but this is done through django. To create a superuser you can use the CLI command `python manage.py createsuperuser`. This will create a django staff account, and create a corresponding

UserProfile class with the Admin role. Another option would be to go to the UserProfile table in the django admin panel, and change the role here.

Users that sign up will be the Student role by default, but admins can go to the “People” tab in the navbar and switch users between Student and Teacher roles.

The way roles are checked in frontend is very simple. Permission checking for navigation is done in *router.js*, the methods *redirectUnauthorizedAdminOnly* and *redirectUnauthorizedTeacherAdminOnly* check if a user has a certain role before navigating to a route, and show a 404 page if they are unauthorized.

To show and hide buttons for certain roles (like the “Create page” button), we use the getter *isStudent*, *isTeacher* and *isAdmin*, and simply use a v-if to show/hide buttons.

Role checking in the backend uses the methods *has\_view\_permission* and *has\_edit\_permission* inside *courses.py* and *pages.py*. These methods check whether a user has view or edit permissions in an endpoint, respectively. If the user does not have the required role, an error is returned.

Adding more roles can be done inside **users/models.py** by adding them to the `ROLE_CHOICES` object, and creating a new migration. These roles are just strings, so to check if a user has a certain role, just do `user.profile.role == 'role'`

## 6.2.2 Frontend Architecture

The architecture of the new system follows the same structure as the existing system. Routes are defined inside *router.js*. Components are defined in their feature specific folder, these are saved in the folder **components/..**

The structure of all the new components can be found in the frontend diagram in section 4.2

## 6.2.3 Backend Architecture & API

For the backend, we created a new django app called “courses” and “users”. Django automatically generated all the files inside these apps.

The app **courses** handles all the logic for the content management system:

- **models** folder contains the models for *Course*, *Page* and *PageItem*
- **api** folder contains all the endpoints for courses and pages
- **urls.py** defines the urls of the endpoints

Since the existing application used django’s default user class, we had to create a **user** app for the role system.

- **models.py** contains the *UserProfile* class
- **api** folder contains all the endpoints related to users
- **urls.py** defines the urls of the endpoints
- **signals.py** contains signals that ensure the roles automatically created and updated when the related user object changes

## 7. Conclusion & Evaluation

### 7.1 Conclusion

For this project we were tasked to expand the existing platform “Twente Hacking Squad” into a Canvas-like content management system. “Twente Hacking School” now supports a role based system with course management, and migrated challenge management functionality into the web interface. Thanks to our concrete planning we were able to stay ahead of schedule for the majority of the project duration and we managed to implement 87% of the functional requirements that were outlined in the planning.

### 7.2 Evaluation

#### 7.2.1 Must

We have successfully implemented all the "must have" requirements. Administrators can now assign roles and create new courses. Teachers are capable of managing pages, adding and editing challenges, and enrolling students. Students can access their enrolled courses, view content, interact with challenges, and see their peers and highscores. These updates ensure a robust, role-specific experience.

#### 7.2.2 Should

In addition to the core functionality, we have also implemented the key "should have" features to enhance usability to the teacher experience. Teachers can now add other teachers to the courses they are part of, enrich course content using markdown, however uploading media such as images and videos directly within pages is not supported. Lastly, a challenge creation form has been integrated to the system, to facilitate the process of creating new challenges.

#### 7.2.3 Could

For the "could have" features, we implemented the first two: teachers can now create pages with interleaved descriptions and challenges, similar to a Python Notebook, and they can reorder pages within their courses. However, the notification features related to challenge modifications and deletions have not been implemented at this stage.

#### 7.2.4 Won't

None of the "won't have" requirements were implemented in the system, as planned by the team during the design phase.

### 7.2.5 Non-functional requirements

While developing the platform, the non-functional requirements were taken into account and the end product meets all of the performance, security, usability, reliability, scalability, compatibility, availability and maintainability objectives. For portability, the responsiveness for all sized devices is yet to be implemented, as mentioned in section 6.1.1.

## 8. Bibliography

1. Vuetify. (n.d.). *Vuetify — A Vue Component Framework*. Retrieved April 16, 2025, from <https://vuetifyjs.com/en/>
2. Wikipedia contributors. (n.d.). *MoSCoW method*. Wikipedia. Retrieved April 16, 2025, from [https://en.wikipedia.org/wiki/MoSCoW\\_method](https://en.wikipedia.org/wiki/MoSCoW_method)

## 9. Appendices

### Appendix A: Mock-ups

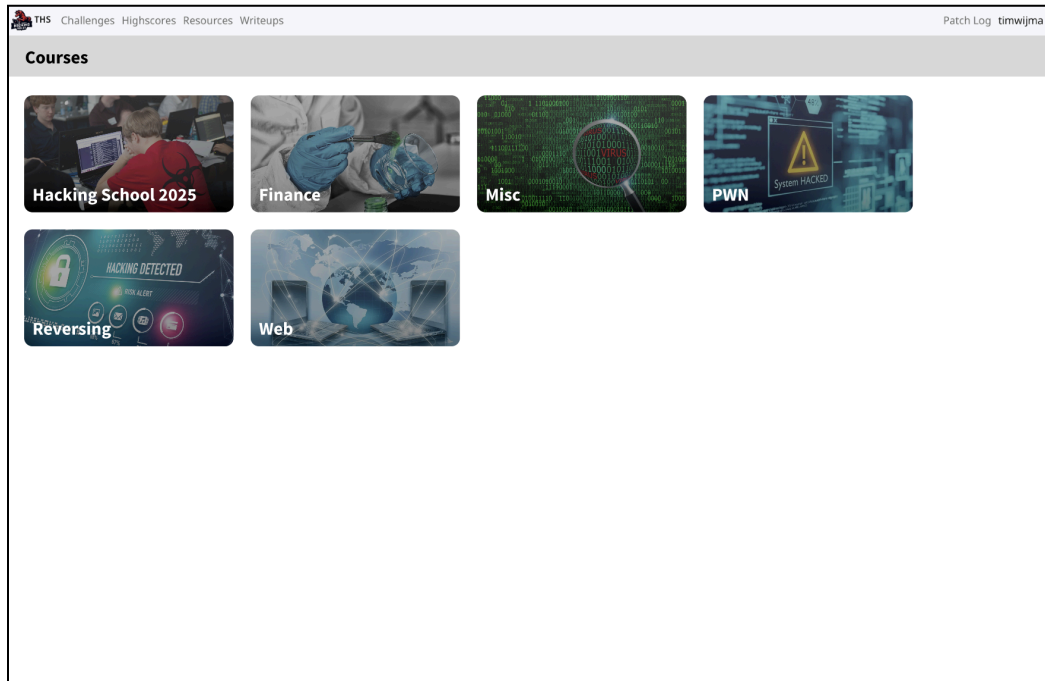


Figure A.1: Course Dashboard

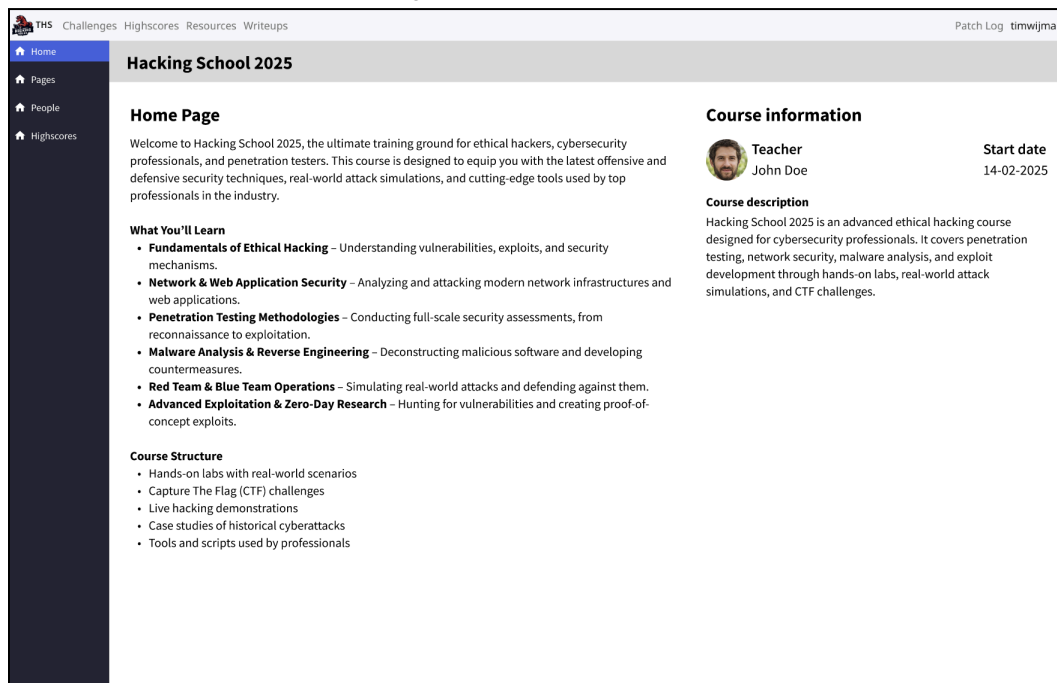


Figure A.1: Course Homepage

THS Challenges Highscores Resources Writeups Patch Log timwijma			
Home Pages People Highscores			
Hacking School 2025			
Pages			
Name	Completed		Mandatory
Your first challenge	No		Yes
Challenge number 2	Yes		Yes
The last challenge	No		No

Figure A.3: Course Pages Overview

THS
Challenges
Highscores
Resources
Writeups
Patch Log
timwijma

Home
Pages
People
Highscores

Hacking School 2025

### Your first challenge

Welcome to your first test as an ethical hacker. This challenge will assess your ability to think like an attacker, analyze vulnerabilities, and exploit weaknesses. You'll need to use reconnaissance, critical thinking, and the right tools to break through the first layer of security. Stay sharp, work methodically, and remember—every system has a flaw. Your mission is to find it. Are you ready? Let the hacking begin.

[Start Challenge!](#)

Figure A.4: Course Page



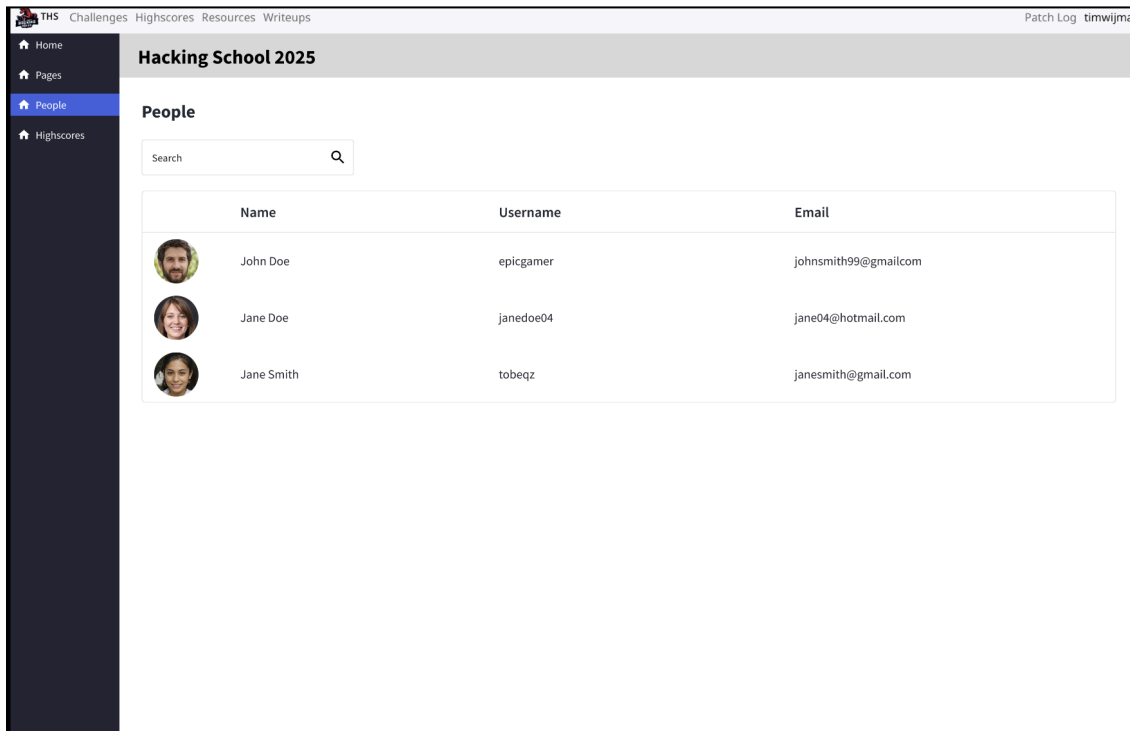


Figure A.5: Course Users Page

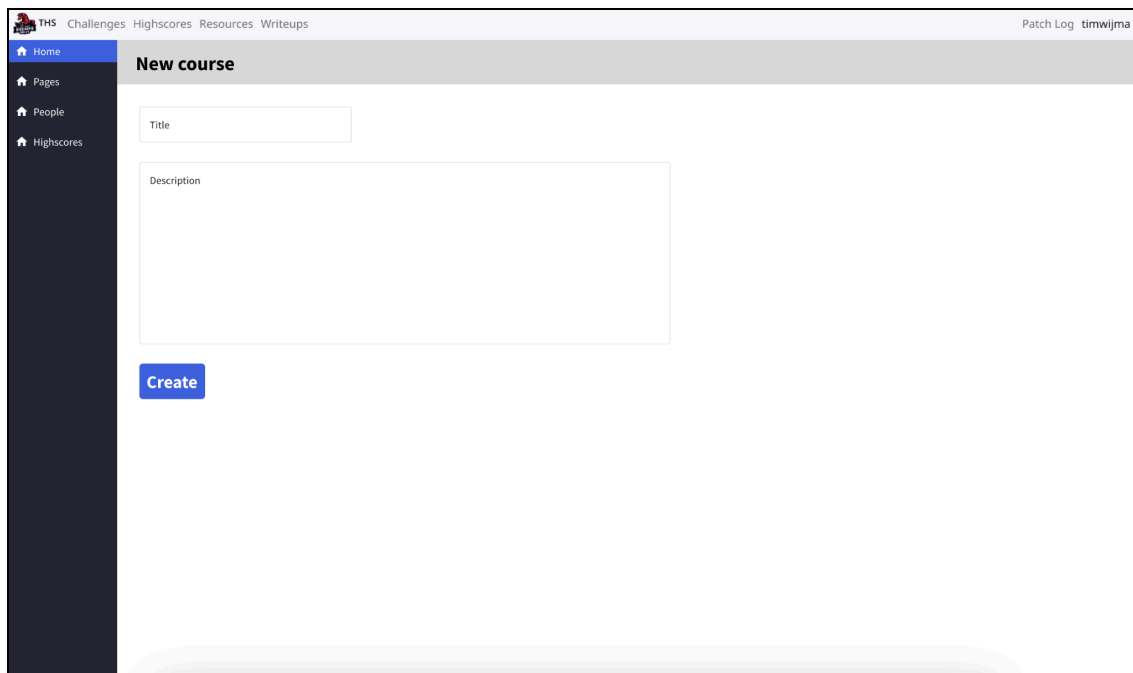




Figure A.6: Course Creation Page


[Challenges](#)
[Highscores](#)
[Resources](#)
[Writeups](#)

[Patch Log](#)
[timwijnma](#)

## People






	Name	Username	Email	Role	
	John Doe	epicgamer	johnsmith99@gmail.com	Teacher	...
	Jane Doe	janedoe04	jane04@hotmail.com	Teacher	...
	Jane Smith	tobeqz	jan smith@gmail.com	Student	...

Figure A.7: Users Overview Page (Admin)

## Appendix B: User Test Scenarios

### Scenario 1: Creating a challenge(Teacher)

**Context:** Logged in as a teacher, the teacher is able to create a new challenge.

- New challenge: **“SQL Injection - Demo”** (make sure the challenge is visible).
- The challenge should have the text flag: **“1234”**

**Goal:** The user successfully creates the new challenge and is able to see it in the list of challenges.

### Scenario 2: Editing a challenge(Teacher)

**Context:** Logged in as teacher who is the author of an already existing challenge:

- Challenge to be edited: **“SQL Injection - Demo”**
- Change description of the challenge.

**Goal:** The user successfully finds the challenge he is supposed to edit. The user edits it accordingly and saves the changes.

### Scenario 3: Edit a course (Teacher)

**Context:** Logged in as a teacher, the teacher is able to edit a course that is already created:

- Course to be edited: **“Cryptography 101 2024-2025”**
- Change title and the image of the course.

**Goal:** The user successfully finds the course that they are supposed to edit. The user edits and saves the changes made to the course.

### Scenario 4: Delete a course (Teacher)

**Context:** Logged in as a teacher, the teacher is able to delete a course that is already created:

- Course to be deleted: **“Cryptography 101 2023-2024”**

**Goal:** The user successfully finds the course that they are supposed to delete. The user deletes the course.

### Scenario 5: Create pages in a course (Teacher)

**Context:** Logged in as a teacher, the teacher is able to add pages to a course that is already created:

- Add a page in course: **“Cryptography 101 2024-2025”** called “page without challenge”  
with a title and description

**Goal:** The user successfully adds a page to the course. The user is able to see the page in the list of pages inside the course.

#### Scenario 6: Add a challenge to a course (Teacher)

**Context:** Logged in as a teacher, the teacher is able to add a challenge to a course that is already created.

- Add a new page to the course: “**Cryptography 101 2024-2025**” called “page with challenge”  
With a title, description and select challenge” SQL Injection - Demo”

**Goal:** Add a challenge to a course through a page. The user should be able to see the page in the pages list of the course and the challenge when inside the created page.

#### Scenario 7: Delete a page from a course (Teacher)

**Context:** Logged in as a teacher, the teacher is able to delete a page to a course that already exists.

- Delete the page created without the challenge.

**Goal:** Delete a page from the course. The user should not see the page on the pages list of the course.

#### Scenario 8: Add a student to a course.(Teacher)

**Context:** Logged in as a teacher, the teacher is able to add a student to a course that is already created:

- Course to add the students to: “Cryptography 101 2024-2025”
- Add student John by clicking the plus sign.
- Add students [student1@gmail.com](mailto:student1@gmail.com) and [student2@gmail.com](mailto:student2@gmail.com) with the email list.
- Search for student Jane and then add her to the course using one of the methods

above.

**Goal:** The user successfully finds the course that they are supposed to add the students to. The user adds all of the students to the course.

#### Scenario 9: View a course and all of the people that are part of that course.(Student)

**Context:** Logged in as a student, the student is able to view a course that is already created and that he is part of, as well as, view all the people that are part of that course:

- Course to be viewed: “**Cryptography 101 2024-2025**”
- View all the people of the course.

**Goal:** The user successfully finds the course that they are supposed to view. The user views the main page of the course and is able to view all of the people that are part of that course.

Scenario 10: Solve a challenge inside of a course & see rank in course highscores.(Student)

**Context:** Logged in as a student, the user should complete a challenge inside of course “Cryptography 101 2024-2025” and then see their rank in the leaderboard.

- Challenge to be solved: “**SQL Injection - Demo**” inside course “Cryptography 101 2024-2025”

**Goal:** after completing a challenge, the student sees their rank on the highscores course page.

## Appendix C: Front-End Diagrams

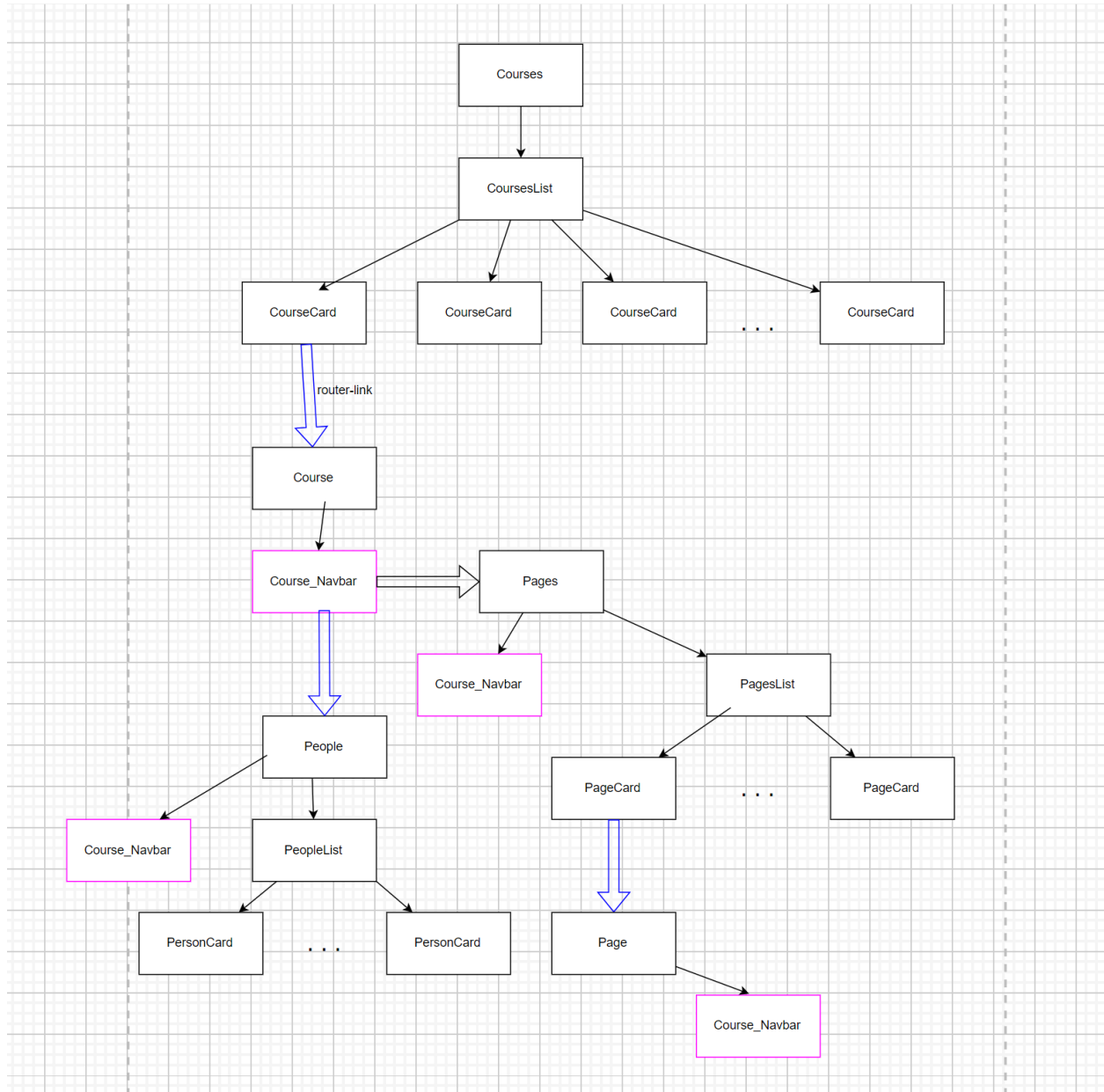


Figure C.7: Version 1

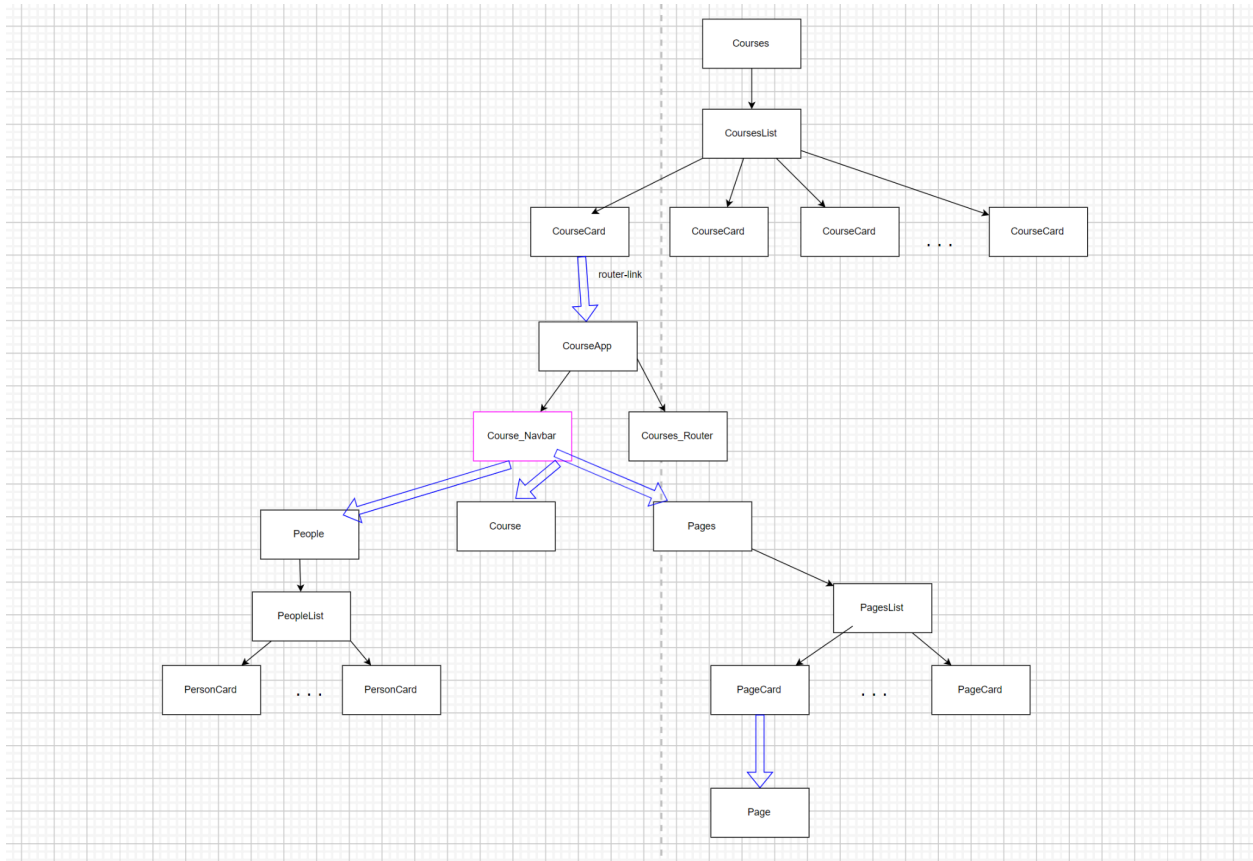


Figure C.2: Version 2

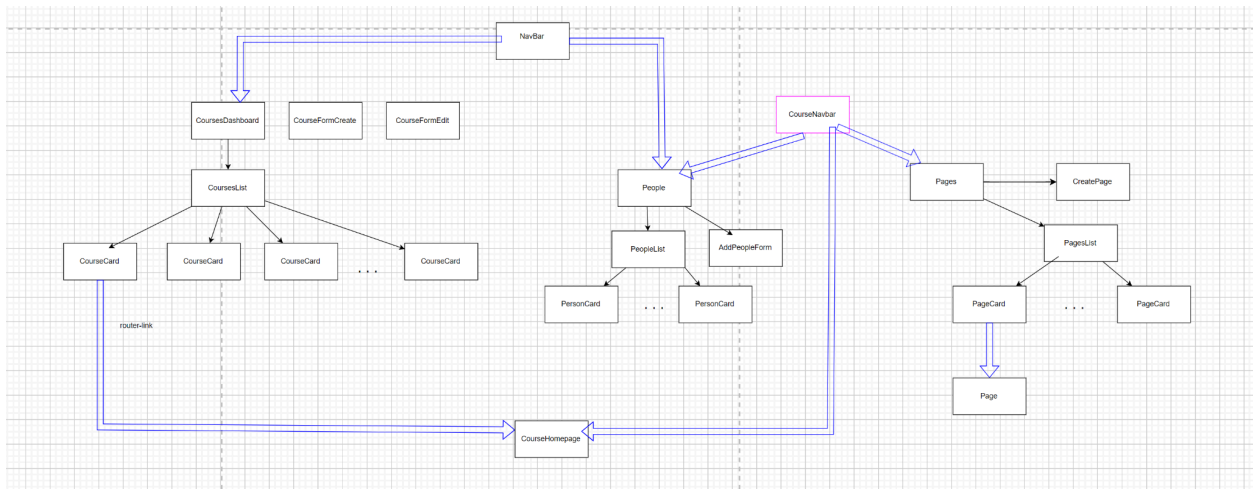


Figure C.3: Version 3

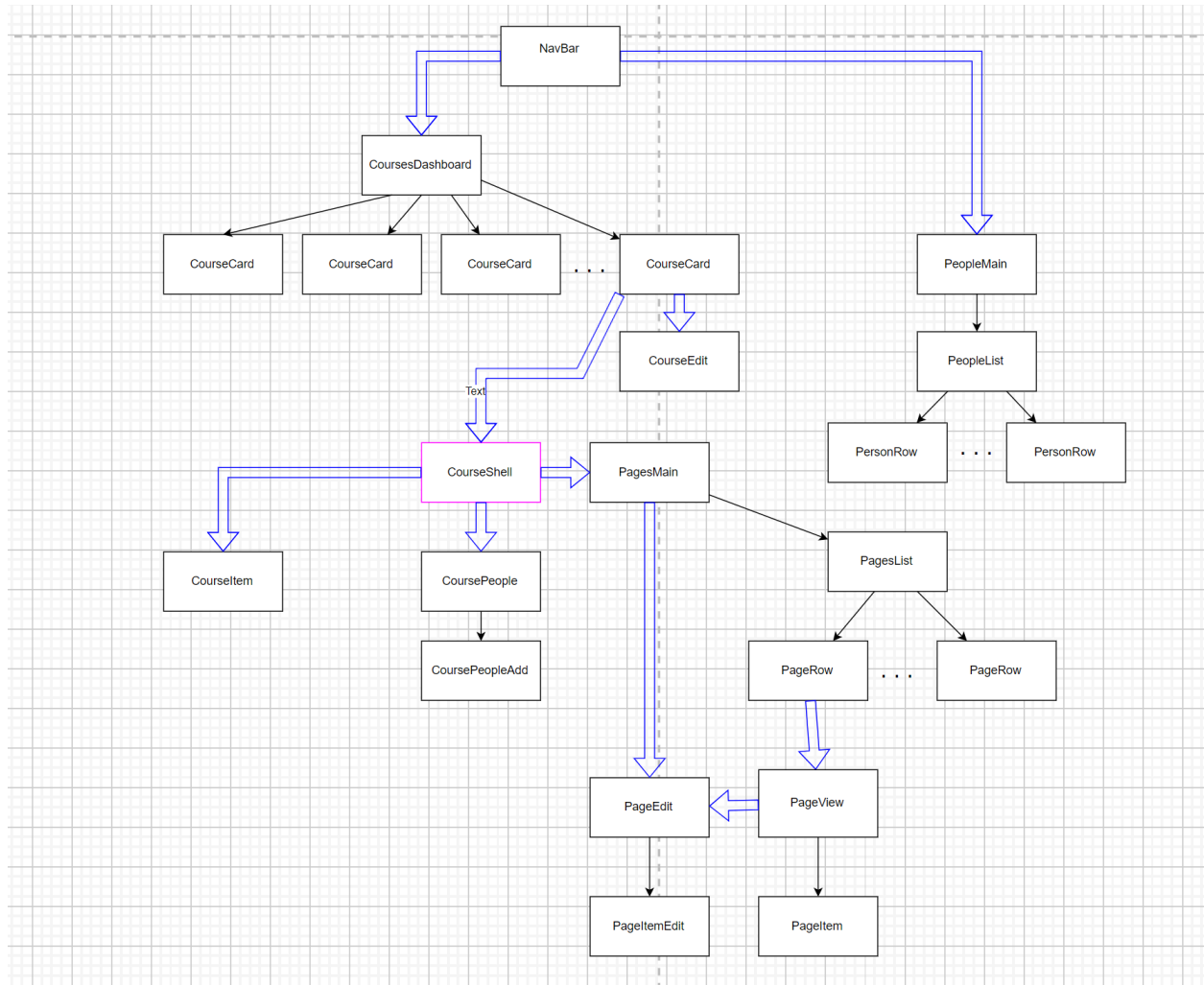


Figure C.4: Version 4 (Final Version)



## Appendix D: Meetings with Client

### D.1 Meeting 1

This was our first meeting with the client/supervisor regarding the project. The main goal of this meeting was to understand the project requirements. During the meeting, we discussed several key points and had a list of questions to clarify the goals with the client. Finally, it was also noted that having a lo-fi prototype would be a nice-to-have addition to the design process.

### D.2 Meeting 2

This meeting was to follow up on some questions from the first meeting and refine the requirements further. We brought up a list of questions, and the supervisor provided further clarification on several aspects of the system and expectations of the project. A discussion was held on what the supervisor prioritized in their evaluation, which focused primarily on the final working product, with an emphasis on stability over feature count. The project proposal was discussed briefly, for which feedback was received.

### D.3 Meeting 3

The meeting focused on refining and confirming specific requirements for the project. Several requirements were confirmed and modified according to our discussion with the supervisor. The supervisor suggested creating some interaction diagrams, to show how users interact with the system.

### D.4 Meeting 4

This meeting was mainly focused on the front-end implementation, user experience, and role based permissions. Our supervisor also emphasized the importance of testing every part of the system. Functionalities should be checked at every step to prevent any bugs propagating later in development.

### D.5 Meeting 5

This meeting focused on improving the user experience and refining the UI. We also discussed documentation and testing expectations for the project.

### D.6 Meeting 6

This meeting continued to refine project functionality, permissions, UI improvements, and backend considerations, particularly around challenge editing, user access, and teacher roles. For testing, at minimum, user testing should be conducted to verify intuitiveness and functionality and unit testing is very important for our supervisor.

## D.7 Meeting 7

This meeting was dedicated to a user testing session with several participants. The goal was to gather feedback on the platform's usability, and user experience. Participants interacted with various parts of the platform, such as challenge creation, course management, page editing, and user enrollment workflows. Most of the feedback focused on intuitive behavior, field validation, and form interactions. Overall, the feedback was highly useful and helped improve the user experience significantly. The majority of the changes are UI/UX-focused, aiming to make the platform more intuitive and user-friendly. This session provided clear direction on which areas need polishing before the final stages of development.

## D.8 Meeting 8

The meeting was focused on wrapping up development and shifting our attention toward documentation, testing, the final report, and the final presentation. The supervisor emphasized critical areas to prioritize during the closing phase of the project.