MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

INTERACTIVE DATABASE OF EDUCATIONAL MATERIALS ON ACADEMIC INTEGRITY

Bachelor’s Thesis

JAN MARTINEK

Brno, Spring 2023

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

INTERACTIVE DATABASE OF EDUCATIONAL MATERIALS ON ACADEMIC INTEGRITY

Bachelor’s Thesis

JAN MARTINEK

Advisor: Mgr. Tomáš Foltýnek, Ph.D.

Department of Computer Science

Brno, Spring 2023

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Jan Martinek

Advisor: Mgr. Tomáš Foltýnek, Ph.D.

Acknowledgements

I want to give my thanks mainly to my supervisor, Mgr. Tomáš Foltýnek, Ph.D., for leading me through the topic and making time for regular consultations. I'd also like to thank my advisor, Ing. Dita Henek Dlabolová, Ph.D. and all the personnel at ENAI for discussing the requirements with me and for giving me valuable feedback. Finally, I am grateful to my family, who have kept my morale up during my studies.

Abstract

The thesis aims to reimplement an existing WordPress plugin created by Leoš Lang into a new web application. The resulting application uses a PHP back-end supported by the CodeIgniter4 framework. The new application is customized directly for ENAI (European Network for Academic Integrity) in both design and functionality. It mostly matches the original functionality, modified by including changes requested by the ENAI members. Its primary focus is on improving the administration interface, file handling, and filtering. The application is available for browsing on the ENAI website.

`https://academicintegrity.eu/materials` [1]
`https://github.com/Razzoved/educational-database-bp` [2]

Keywords

PHP, framework, MVC, CodeIgniter, JavaScript, database, material, academic integrity

Contents

List of Tables

List of Figures

List of used tools

- Linux - operating system used for writing the text of the thesis.
- Grammarly - online tool for correcting texts in english.
- Texmaker - tool for writing LateX documents. It was used with fithesis4 template.
- Windows - operating system, used for writing the code and running the development enviroment.
- Insomnia - program used for testing the requests to the server.
- Visual Studio Code - code editor. All of the application's code was written in it.
- Laragon - program capable of running a local development server.
- HeidiSQL - database management tool, used alongside Laragon.
- Mozilla Firefox, Microsoft Edge, Google Chrome - web browsers. Application was tested on all of them.
- Git - versioning tool. GitLab FI (git management software hosted by the faculty) was used during development, while GitHub holds the final product for long-term unrestricted availability.
- WinSCP - tool for transferring files from one computer to another. The application was deployed to the server with it.
- Visual Paradigm - tool for creating UML diagrams.

1 Introduction

Previously, it was almost impossible for people to share information globally. However, this has changed drastically in recent decades with the rise of the Internet. Each and every year, the number of people using it grows. Over 63 % of people worldwide can utilize it to convey or acquire new information [3]. With the increased usage also comes an increase in available resources. While this is great for variety, it also sometimes creates an obstacle for anyone searching for valuable learning materials.

European Network for Academic Integrity (ENAI) is an organization comprising higher education institutions that strives to support its members and the public in the field of academic integrity. An interactive database of educational materials has been made available on the ENAI to aid in this effort and has seen considerable success since its conception.

The thesis bases itself on the previous version of this database application, originally created as a more general plugin for WordPress. The plugin offered a system supporting the management of educational materials to anyone employing WordPress. While still widely used, the ENAI personnel wanted to move from WordPress new solution. They were already considering updating the plugins user interface to a more accessible look, bet were restricted by the WordPresses capabilities. The final resolution came when the plugin stopped working after a WordPress update. Such a decision requires the reimplementation of the plugin. The rework is also used to introduce new features, refine the existing ones, and tailor the application specifically to the needs of ENAI.

The application uses PHP as its server-side language, supported by a CodeIgniter4 framework. Frameworks provide many benefits, such as speeding up development by providing many often-used functions or enhancing security and maintainability. The project is designed to be a standalone web application, a small content management system. It must be as portable as possible, with only a few steps needed to run on most servers that support PHP. The user side of the application, or front end, uses HTML, CSS, and pure JavaScript under the ECS6 standard.

The application can be divided into two almost separate areas. First is accessible by anyone and features a view of single or all materials. It allows users to download attached files and view related subjects and websites. The search supports suggestions and, along with filtering, should be easy to grasp and control. The second is a restricted management area. It allows easy and intuitive management of materials. The management includes the creation, updating, and deletion of materials, tags, accounts, and files. A dashboard is included to allow viewing of statistics of use.

1.1 Thesis structure

The second chapter takes a look at the overall context. The third chapter focuses on the analysis of requirements. The fourth one focuses on design decisions, showcasing select diagrams. The fifth chapter focuses on selected implementation details and challenges. The sixth chapter evaluates the work and offers additional insights and possible future improvements.

2 Prerequisites

The application is revolves around e-learning in the area of academic integrity. It is a content management system many people use to educate themselves in this area. Aside from personnal education, many of the available materials are intended as a tool to support the teachers in their lessons. With this being the case, I think it important to explain those terms.

2.1 European Network for Academic Integrity

"The European Network for Academic Integrity (ENAI) is an association gathering educational institutions and individuals interested in maintaining and promoting academic integrity" [4]. It aims to support its members and the public in the field of academic integrity, being the most extensive academic integrity network in Europe. Aside from organizing the European Conference on Ethics and Integrity in Academia, ENAI also provides several free services to the public, such as a glossary on academic integrity, monthly webinars, self-assessment tools, support for the victims of academic misconduct or a database of educational materials.

The mentioned database is one of the key players in supporting its efforts. The database runs on a WordPress plugin [5] made by Leoš Lang. Its caretakers wish to have it remade into a separate web application. Together with this request, they want to change or remove several old features. Those conditions are examined later in section 3.

2.2 E-learning

The first appearance of e-learning dates to the mid-1960s, but the real growth began much later with the growth of the Internet in the late 90s [6]. Since then, many forms of e-learning have come into existence, be it interactive forms like videos or more passive ones like e-books.

Most of the forms can be described by the following definition: "E-learning is the use of electronic technologies to create learning experiences" (7, page. 1). Learning is experienced with the usage of learning objects. “A learning object is a chunk of electronic content that can be accessed individually and that ultimately accomplishes a single learning objective and can prove it” (7, page. 47).

2.3 Content management systems

“A content management system (CMS) is primarily a repository for multimedia content objects” (8, page. 17). According to this definition, three main areas every CMS should provide can be identified:

- collection - acquiring information into the system, be it from existing or newly created sources.
- management - modifying or removing information that has already been collected and saved into the system.
- publishing - making the content available to the users through the web, e-mail, or any other means, such as printed copies.

For this thesis, it is important to mention two concrete kinds of CMS. First, an aim of the thesis. A system that handles educational materials directly and simplifies the collection and management operations for the user. The other one is more advanced but serves a different purpose. It allows any user to create and manage websites without prior programming knowledge. One of these is WordPress, mentioned in section (ref. 2.5).

2.4 PHP

Hypertext PreProcessor (PHP) is the currently most used server-side scripting language. Being an open-source project able to run on most operating systems ensures portability to almost any server. Its main advantages are platform and database independence or easy understandability with C-like syntax. It offers many library functions by default, such as array or hashing helpers, easing the burden on developers. PHP code can be embedded into HTML files. The server reads and alters the contents based on the embedded code before sending the page to the user (9, page. 30). PHP is very fast and scalable. It can also service virtually any content, whether in the form of small project or a large-scale website like Facebook. The updates to PHP are mostly backward compatible, so little or no changes are usually required. Though be warned that this is not a always the case.

2.5 WordPress

WordPress is the most popular content management system. It is an open-source platform running on PHP that allows anyone to contribute to its improvement. Due to the rapid development and wide range of downloadable plugins and themes, WordPress maintains a high following. It is used by 63.3 % of all websites whose content management system is known or 43.2 % in total [3].

WordPress has a large and active community of users who share knowledge and resources related to WordPress development. The use of WordPress plugins enabled developers to participate in this community and leverage existing WordPress features and knowledge.

The core of WordPress is designed to be more lightweight and flexible. A WordPress plugin is a collection of PHP scripts that can be accompanied by custom stylesheets and templates. Their purpose is to extend the core with additional functionalities. They can make changes to the database, show content to users, integrate into administration, automate tasks (such as comment filtering), or function as a separate application.

2.6 Existing solutions

This section was written with help of ChatGPT.

A manual and AI-assisted search for a freely available software that could be used as a replacement for the plugin was conducted. It resulted in a reveal of two distinct system design trends. The following text elaborates further on those trends, adds several examples, and mentions why none of them was selected for for the project.

The first of the trends if a category of systems known as Learning Management Systems (LMS). Those are a kind of CMS, albeit with a more specific focus. This focus is on delivering full-length courses. The terms ′course′ and ′educational material′ are often used interchangeably, but the two have a number of key differences.

A course is a structured educational experience that is designed to provide learners with a specific set of learning outcomes or objectives. A course typically has a defined start and end date and may be delivered in a variety of formats, such as in-person classes, online modules, or a combination of both. Courses usually have a set curriculum outlining their topics and learning activities. They may include assessments or evaluations to measure the learner's progress and understanding.

Educational material refers to any content or resource designed to support learning, education, both from the view of students and teachers. Such resources include textbooks, exercises, videos, tutorials, articles, and more. Unlike a course, educational materials are often self-paced, self-directed experiences. This means that learners and educators can access and use them on their own schedule and at their own pace.

While both courses and educational material aim to support learning and education, courses are generally more structured and have clear outcomes, and require more preparation work before release. On the other hand, educational materials are more flexible and have more uses, like teaching other than learning.

The second trend were generalized CMS. This category contains all systems that are similar to WordPress.

2.6.1 System list

- **Moodle** [10] is a very popular platform. It was created in 2002 by Martin Dougiamas, an educator from Australia. Nowadays, it has more than 160000 sites in no less than 240 countries. Moodle provides a wide range of features, including course management, assessments, collaboration tools, and more. It is extensible with plugins.

- **Open edX** [11] is an open-source platform for delivering massive open online courses created by edX, a non-profit organization founded by Harvard and MIT. One of the key features is its modular architecture, which allows for easy customization and extension. It has a large and active community of developers and users who contribute to the platform's development and support, enhancing its overall functionality.

- **Canvas** [12] is a platform that includes features such as course management, assignments, quizzes, and discussions, as well as integration with other tools such as Google Drive, Zoom, and Microsoft Office. Canvas is written in Ruby programming language. Ruby is unavailable on the current hosting [13].

- **Omeka** [14] is specifically designed for creating and managing digital collections and exhibits. It includes features such as customizable themes, metadata management, and access control to create a flexible and organized environment for digital content. It is a CMS.

2.6.2 Final verdict on the systems

All of the systems fullfil their given roles, and they do it very well. However, they all mostly share the same fault, aiming for a different objective. What is needed for the project is a system tha can manage different resources, but not in a form of whole courses. A course is difficult to prepare, and many of the materials that the ENAI wants to gather are not only used for learning, but for teaching too, and a teacher usually does not use an online course as a teaching tool during one of his classes.

The general CMS are denied too. One of those systems has already been tried on the server. The original plugin broke as a result of one of the WordPress updates, which caused many issues. As such, the overhead and possible incompatibilities those systems introduce when their plugins or applications are not under active maintenance solidified their denial.

2.7 MVC

The Model-View-Controller (MVC) pattern is a popular design pattern used in web development. Its main idea is to separate the application into three interconnected components: the Model, the View, and the Controller. The MVC pattern's primary benefit is promoting the division of responsibilities and modularity. By splitting the application into distinct components, each meant for a specific set of tasks, developers can focus on one component at a time. Changes to that component do not necessarily affect the other components. This makes it easier to maintain and update the application over time.

"The Model is the part of the system that manages all tasks related to data: validation, session state and control, data source structure. The Model greatly reduces the complexity of the code the developer needs to write." [15]. The Model represents the application's data and business logic, defining how data is stored, manipulated, and retrieved. It provides all methods for interacting with the data and is responsible for maintaining their consistency and integrity.

The View component represents the user interface and may be thought of as a front-end of the application. It is responsible for presenting data to the user in a visually appealing and understandable way and allowing interactions between the user and the application. "By separating the design of the application from the logic of the application we greatly reduce the risk of errors showing up when the designer decides to alter the interface of that application by changing a logo or a table" [15].

The Controller component acts as an intermediary between the Model and the View. "A controller accepts requests and prepares the data for a response. It is also responsible with establishing the format of that response. The Controller interacts with the Model in order to retrieve the needed data and generates the View" [15]. A controller may verify or transform the user inputs before passing them to the Model and generating a response.

3 Analysis and Design

The first part looks at the existing plugin. The second discusses the requirements for the new implementation. After requirements come the use cases, data modeling, and UX design.

3.1 Plugin

The plugin that has been mentioned several times already is an extension of the WordPress content management system. The extension serves as an interface between a user and the servers' filesystem and database. It provides teaching and learning support for various topics on academic integrity. It has seen widespread use since its release, containing over 250 different materials. Those materials must be preserved during the transfer to the new implementation. It is essential to determine the plugins' content and data structure to achieve this. The following analysis is to fulfill this purpose and, moreover, review the possibility of using the code of the plugin instead as a base instead of starting from scratch. To get more information about the plugin, see the thesis written by the plugin creator [5].

After the analysis, it was clear that putting the original code to use would prove more difficult than starting from zero (ref. 3.1.2). The application is still meant to be a replacement, so the existing user interface and data need to be taken into account while discovering the requirements.

3.1.1 User interface

The first step in analyzing the plugin is to familiarize ourselves with the user interface, making the next steps easier to grasp by understanding the context. Its graphical design is quite plain, which is in no way a bad thing. Many websites are suffering from overdesign, and the plainness makes the plugin look clean and easy to navigate in comparison.

The plugin inherits the same navigation page as its parent site. This feature cannot be a part of the new implementation as it aims to be a separate software. The plugin adds several viewable pages:

- **Material details** - shows all data of a single material. This includes connected data such as relations or files.
- **Multiple materials** - the materials displayed in a list. This list is paged and can be filtered by both the search bar and filters. Every material shows its title, thumbnail, publish date, rating, short description, and views.
- **Materials table** - under restricted access. It shows all materials in a paged table. The table supports bulk operations but lacks adequate support for searching, filtering, and sorting.
- **Material editor** - under restricted access. It is used for editing or the creation or a single material and contains input fields for materials title, sender, type, and content. Additionally, it allows the assignment of taxonomies, thumbnail, files, or a single link. Relations can be added manually. All of the additional features are hard to operate, requiring a high amount of scrolling and clicking. For example, the thumbnail is at the bottom right side of the editor and not easily accessible. The same goes for taxonomies. Files must be uploaded to the shared servers directory, and only then can they be added to the material, which is an arduous process.
- **Configuration** - under restricted access. Used for setting the theme and basic settings like the number of shown items per page.

Taxonomies can be managed anywhere in the plugins' administration area. It should be noted that the materials cannot have multiple external sites linked unless added directly to the content. Material cannot have both an external link and files, always only one of the two. The search bar includes suggestions, but they are not very responsive.

There are several things that could be improved with support for mobile devices. Most of the buttons appear often appear out of place. The filters use a different layout from the PC version, are broken, and cannot be closed after they were opened. Many of the filters often belong to a single category, and since there is no support for taxonomy hierarchies, it creates large lists that are hard to use.

3.1.2 Structure

At a glance, the plugin seems straightforward. It does not use the WordPress MVC framework, instead following a standard plugin folder structure:

- *css/* - contains compiled SCSS files
- *sass/* - contains uncompiled SCSS files
- *img/* - contains default image assets
- *inc/* - contains all serverside scripts (PHP files)
- *js/* - contains all javascript files
- *e-learning-file-manager.php* - plugin entry point
- *uninstall.php* - not implemented

The plugin uses a range of custom data types and implements their related functionalities by itself. The code is poorly documented outside of the mentioned thesis [5]. The whole implementation depends heavily on WordPress and its application interface. It supports older versions of PHP with guaranteed support for version 5. This directly translates to incompatibilities with newer versions. It is not actively maintained, causing additional incompatibilities with WordPress, too. Both the high dependency and incompatibilities pitched a decision to drop the plugin's code altogether, using only its analysis. With this, the database can finally be looked at.

Since the plugin works as an educational database, it must handle educational materials. The materials can be decomposed into the following properties:

- *Title* - the name of the material.
- *Sender* - the name of the person who sent or created the material.
- *Date* - of when the material was published.
- *Type* - indicates the content type of the material.
- *Tags* - categories and other metadata.

- *Comments* - provided by users to receive feedback on materials.
- *Rating* - provided by users to favor quality materials.
- *Content* - can be a description of provided attachments or a complete learning experience.
- *Attachments* - link to a website or files on the server.

Sender as an attribute has only ever been used during the materials' creation, never shown otherwise, and can be removed. While shown, the type has never seen much use and should be merged into tags.

Tags are a central part of the application. They are also known as taxonomies. While the data structure allows hierarchization, the front-end never implemented it. Due to that, they are ordered in a category-to-value relationship. Over the years, many tags accumulated, categories started overflowing, and their refinement is long overdue.

The rating was originally a part of the comment system, where each comment could contain a rating of one through five. Many malicious users took advantage of the comments to spread spam and advertisement messages, and the comments were disabled. The rating should be a separate function in the new version, with comments never returning.

The attachments could either be links or files. Files were saved into folders based on the year and month of their upload. This should be changed into a model where every attachment belongs directly to the material as per the administrator's request.

3.2 Requirements

3.2.1 Functional requirements

Based on the analysis of the plugin and discussions with the ENAI personnel, new functional requirements were stated. Those requirements represent a set of functionalities the application must provide its users.

- creation and management of materials, resources, tags
- assignment of resources and tags to a material
- viewing and filtering (by title and tags) of all materials
- supporting multiple language materials (references)
- rating of materials
- setting of the home page and about page
- setting of a default image
- gather and view statistics about material usage

3.2.2 Non-functional requirements

Non-functional requirements represent a set of conditions that the application needs to be aware of since they can limit its operation by disallowing the use of some technologies or imposing that specific metrics such as quality of delivery be met.

- application must be compatible with Forpsi server [13]
- data are saved in a MySQL database
- PHP 7+
- internet connection and web browser

The application should be mobile-centric. It should be designed for mobile users first and only after that for computers or other devices with larger screen sizes. There should be a high level of focus on the design of a user-friendly interface. Each action should be well understood and simple to do. This requirement is weighed more in the management section in this case.

3.3 Use case diagram

The use case diagram is a behavioral diagram from Unified Modeling Language (UML). It is a graphical representation of the relations between the users and their requirements on the system. A use case diagram highlights the system's functionality from the point of view of one or many users. The user is not necessarily a physical person and is generally called an actor. The diagram differentiates between the actors' roles, even if the actor is only one user in reality. Roles can be grouped into role hierarchies, in which the children inherit the use cases from their parents. A use case diagram is often used to aid in stating the systems' design goals and communicating those intentions to clients.

As seen on the diagram (see Fig. 3.1), the application requires a two-level design approach: the presentation and management layers.

The presentation layer, also called the public layer, is accessible to anyone. Its primary responsibility is the presentation of saved materials to anyone visiting the website. In this case, there are two different functionalities. The first is a view of multiple materials, and the second is a view of a single material in greater detail. In the first case, the materials should be searchable by their titles and, at the same time, filterable by their assigned tags. Every user should be able to view the top-performing materials. The details must contain full-length content with various attachments, be it links that can be visited or files that can be downloaded, and an interactive material rating.

The management layer is the core functionality of the application. It should be restricted by an authentication process that every user needs to pass before being granted access. This should protect the vulnerable database from malicious actors. This process is depicted by *Login* action of any user (see Fig. 3.1). Logged-in users can log

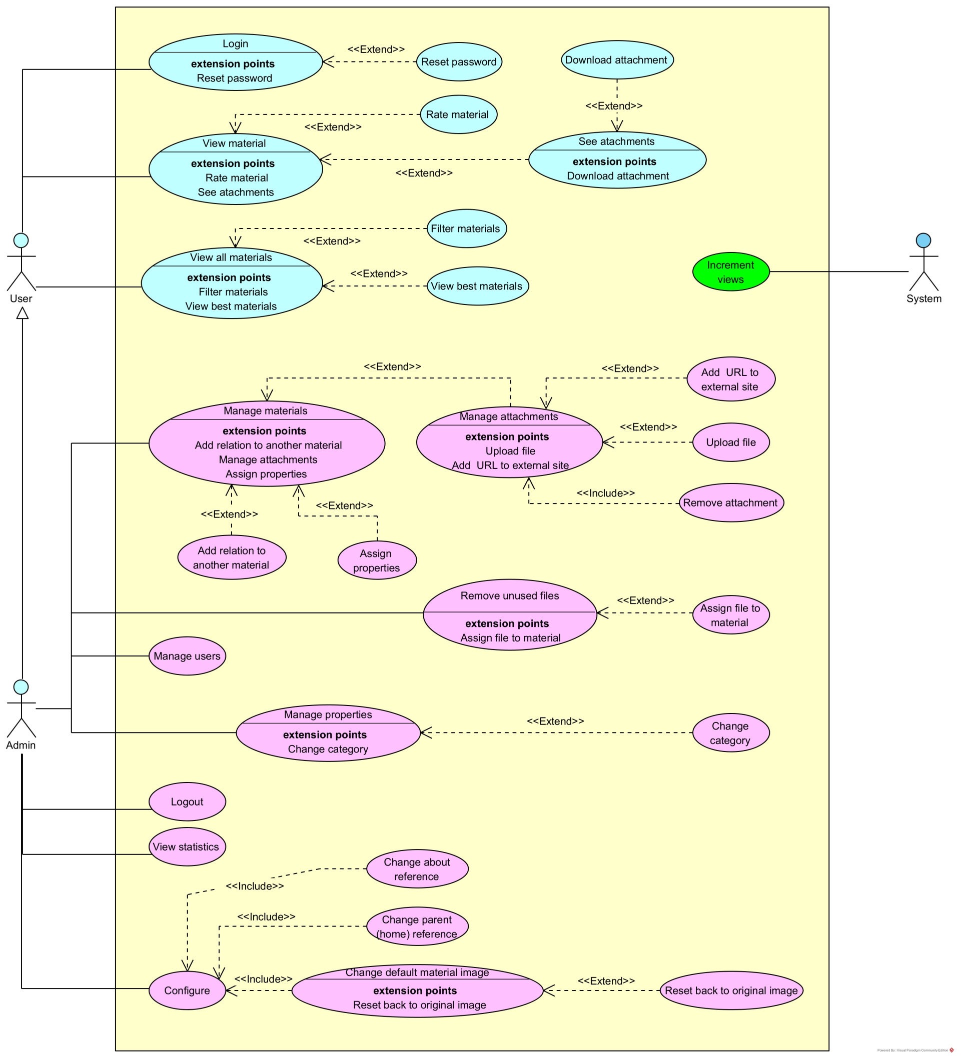


Figure 3.1: Use case diagram

out if they desire to do so. Any user should be able to reset his own password.

After logging in, the administrator must be able to see usage statistics and manage the database. Database management includes the creation, modification, and removal of all relevant data entities. Tags should be protected from unwanted removal if they are in use. Resources should be fully managed by the editor that handles the materials, with the only exception being unused files. Unused files are files that were assigned to some material, lived in it for some time, and then were either replaced or removed. All the data assignable to a material can be created during its creation, simplifying the work significantly by removing the need for repeated operations. Administrators should be able to create new administrators. Any administrator should be able to change the application's home reference and the reference to the About page. Each material uses a shared default image as a thumbnail in case of having none, and this default image should also be configurable. This configured image should always be reversible to the default one.

3.4 Data modelling with ERD

An Entity-Relationship Diagram (ERD) is a graphical representation of data entities and their relationships. It is commonly used to model the system's data structure when designing databases. An ERD consists of entities, entity attributes, and relationships. Every entity represents a single data object in the system. Sometimes there is a need to connect two entities in a many-to-many relationship, which is impossible in a typical database. An additional connector is required to separate this relation into a many-to-one one-to-many relation. ERD does not have a limit on the number of data entities. However, it is good practice to try and keep them at a minimum. Each entity can have multiple attributes. An attribute is a representation of a value of a column for a given instance of an entity. Each attribute can have constraints imposed upon it, such as a primary key or foreign key constraint.

The new implementation will be a standalone project and does not need to conform to WordPresses' guidelines anymore. The only

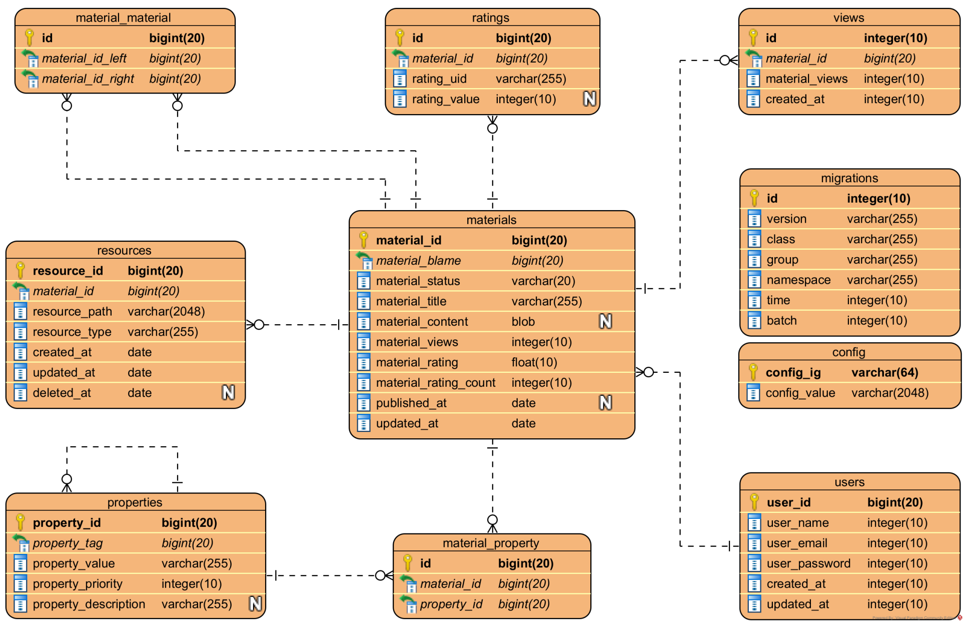


Figure 3.2: Entity relationship diagram

limiting factor is the requirement of compatibility with a MySQL database driver. The final schema will be different from the plugins' original. The ERD (see Fig. 3.2) displays ten separate entities.

Two of the entities serve as support for the application and do not have any relations. The *migrations* entity is added in case a database migration is needed. It holds information about successful migrations and enables their rollbacking whenever defined. The *config* handles the storage and distribution of settings configurable by administrators, like the replacement of the aforementioned default image.

The following entities store the application data.

Materials are saved into *materials*. Several redundant columns are added to them to reduce loading times and allow for easier sorting later in the implementation. The redundant columns are *material_views*, *material_rating* and *material_rating_count*. Each can be computed by aggregating their relative tables, but aggregation is costly and can mostly be done on update instead of select. We can presume that selection is a much more frequent operation.

Initially called taxonomies, tags are stored in a *properties*. Tags will also be called properties in the implementation. Properties can be part of property hierarchies, forming tree-like structures where *property_tag* acts as a parent of the property. An additional connector table is required to assign properties to materials without duplication.

Most of the original metadata is contained in *resources* and connected to materials by a connector entity. Resources include URLs, downloadable files, and thumbnails. Paths are stored relative to the root folder of the application. Links have the entire value saved.

Related materials are kept inside of *material_material* entity, where each row forms a bi directional relation. Material can be related to an unlimited amount of other materials, but not to self.

Each material can be rated, and the ratings are saved in *ratings*. Each row holds a rating of a material given by a single user, with the average computed and saved into the material.

The difference in the database proved hard to overcome when importing the original database to the new implementation. It was deemed to be about the same work as manual upload, which was done instead and served as the first real test of the system.

3.5 UX Design

"User Experience (UX) design is the act and art of crafting the interface and interactions for a website or application [16]." Wireframing is just one of its parts. It produces conceivable layouts of what the final product could look like. Wireframes are usually bare, composed only of boxes, triangles, and other simple shapes, but they still give the art direction. UX design is usually prefaced with in-depth research of the target audience along with the discovery of requirements, consideration of limitations of technologies, and, in the case of redesigning an application, the question of what is hampering the users.

In the case of the original, most of the complaints came from the administrators, but the publicly available view had its share of problems too. From the thesis [5] it can be seen that the design and wireframing were done with larger screens in mind. In the past few years, the use of mobile devices grew exponentially, with now over 58 % of worldwide internet traffic being mobile devices [17]. The specific issues are elaborated upon in the plugin analysis section (ref. 3.1).

3.5.1 Wireframes

The first and foremost decision was on the layout of the filters. Filters are one of the most used parts of the application, present in both the public and administration layers. The layout needs to support the nesting of properties. The implemented design (see Fig. 3.3) needs to be very flexible. Each *property* that has children can be closed. Every property can be selected, so checkboxes have been added. If the checkbox is checked and the item has children, the children should disappear. In some cases, one category contained many values, so a *show more/less* button is added to further reduce the displayed item count. A similar layout is used inside material details but with no checkboxes.

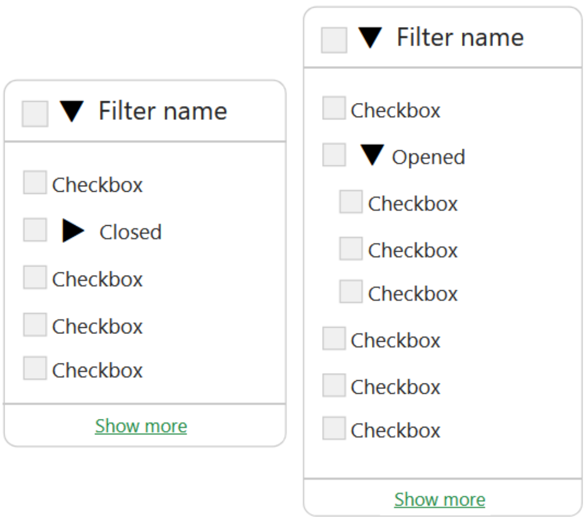


Figure 3.3: Wireframes of one group of filters

Another step was the creation of a materials' card (see Fig. 3.4). The card is an overview of the materials' information and should be used when displaying multiple materials on one page. It shows a thumbnail at the top, followed by the title of the material, and then other details, the most important being the rating. The whole card should indicate that it is clickable. On a larger screen, the card is more horizontal than vertical but shows the same data (see Fig. 3.5).

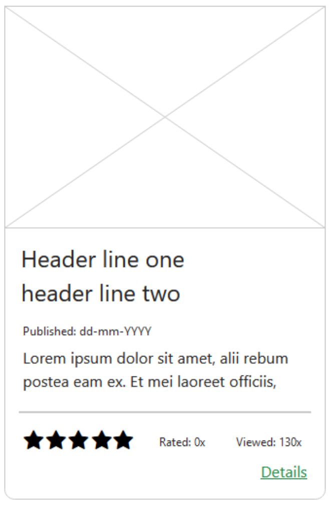


Figure 3.4: Material card for mobile devices

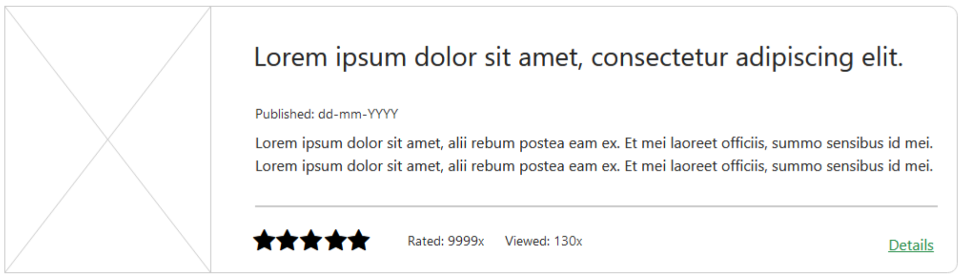

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Published: dd-mm-YYYY
Lorem ipsum dolor sit amet, alii rebum postea eam ex. Et mei laoreet officiis, summo sensibus id mei.
Lorem ipsum dolor sit amet, alii rebum postea eam ex. Et mei laoreet officiis, summo sensibus id mei.
★★★★★
Rated: 9999x
Viewed: 130x
Details

Figure 3.5: Material card

With those two components, the public page can be designed. The mobile version (see Fig. 3.6) is stacked in one column. On top of the page, a navigation bar is present. The navigation bar is toggleable if the links do not fit, showing them on separate rows (3). Under navigation, a page title is added. The filters can be toggled between hidden and shown, freeing scrolling space. A search bar is added to add the ability to search for materials. The only item left is the cards, which can be paged.

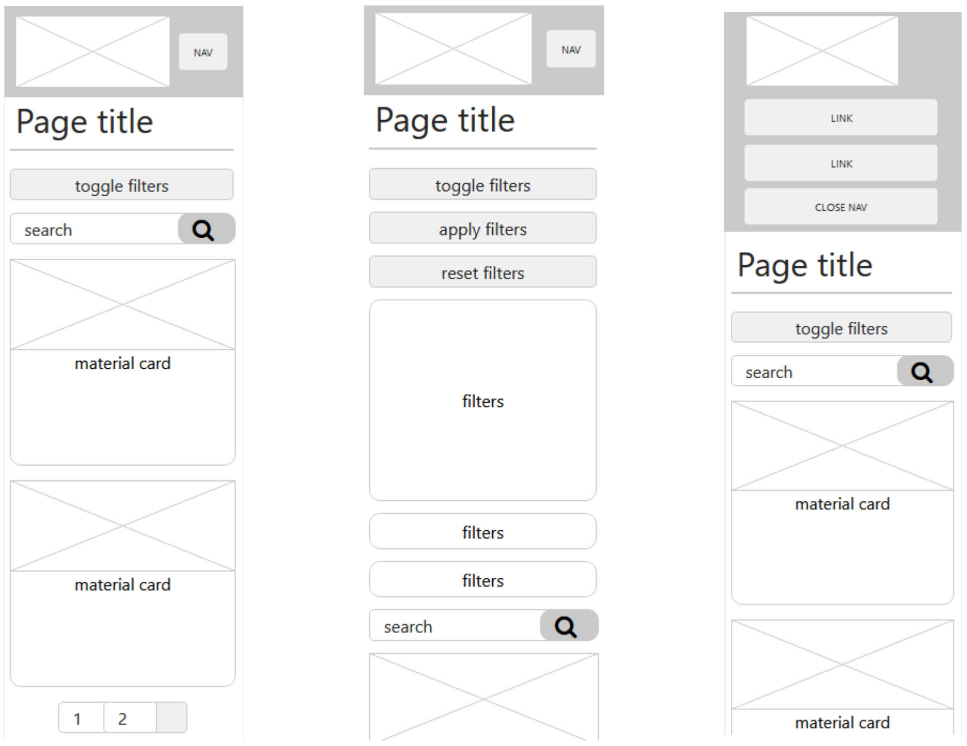


Figure 3.6: Public mobile page: (1) default, (2) opened filters, (3) opened navigation

The computer version uses a two-column layout (see Fig. 3.7), with one being a much smaller sidebar where the filters are placed. The rest is similar to the mobile version, only using the space more freely and missing the ability to hide all the filters completely. The navigation bar should fit all links without the need for toggling, too.

The material details use the same layout as the public page (see Fig. 3.7). The only significant difference is the header. The header contains the information seen on cards, except the description of the material. The thumbnail prefers being on the left side, switching to full width on mobile. The go back button should be present to allow

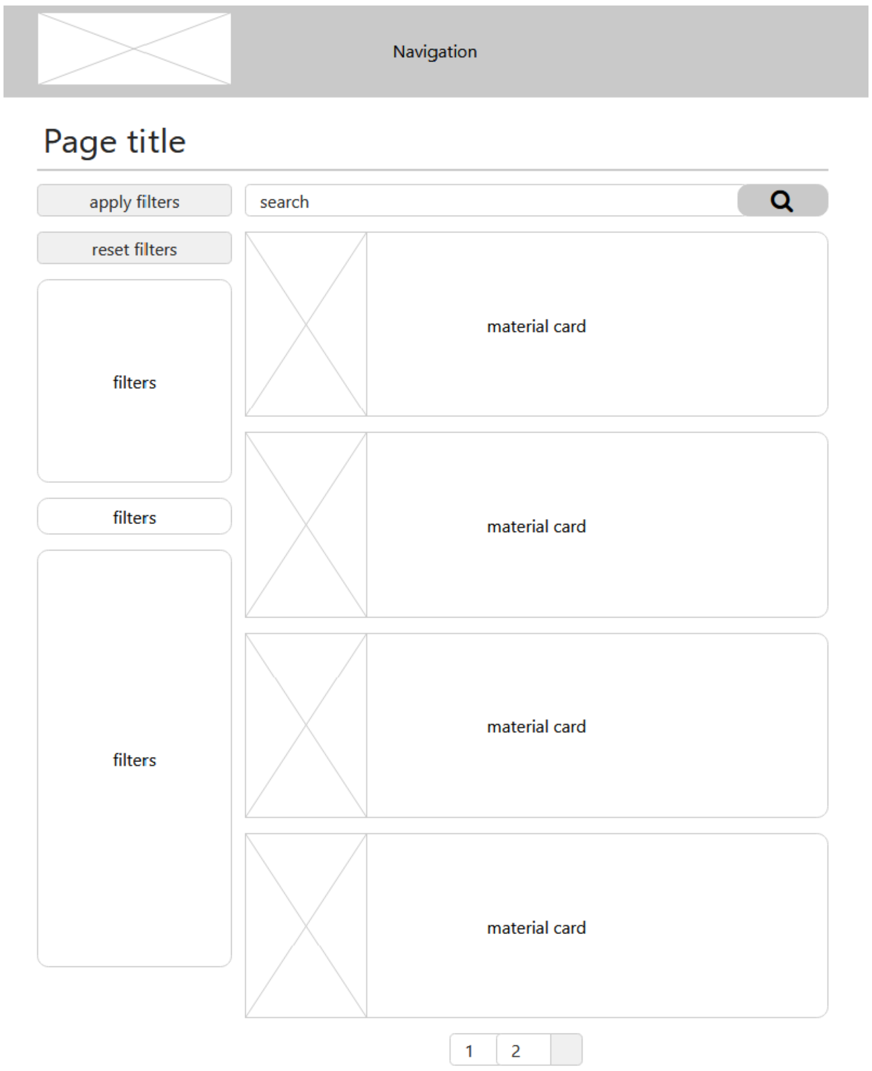


Figure 3.7: Public page

an intuitive return to the all materials page. Content, attachments, and other features are placed in a column below the header.

No additional wireframes were made because the administration follows the same design guidelines. The main differences are that administration cards contain additional information, such as ids of materials, date of last update, or current status. They also include clickable buttons enabling editing and deletion. Other design details were decided upon during the implementation.

4 Implementation

On the client's side, the application uses HTML to present the views and CSS to style them. The CSS was not written directly but compiled from SCSS. To add interactivity to the application, JavaScript was either embedded directly into the HTML files or loaded independently. The applications' server side is written entirely in PHP. To help with the implementation, a PHP framework CodeIgniter4 was used (ref. 4.1).

The application was developed on a local development server with the assistance of Laragon. Laragon is a Windows application that can run an Apache server with a MySQL database. The development server had been set up to match the deployment server as closely as possible.

A versioning tool, git, was used to improve the development workflow. A GitLab repository on the FI domain was used for development, but the final product resides on GitHub with revised commit history.

The editor of choice has been Visual Studio Code. Visual Studio Code is a powerful code editor with easy and quick extendability through extensions. It integrates the git directly into the editor. Added extensions were used to provide syntax highlights, code suggestions, SCSS compilation, and error lensing.

The implementation is described by way of looking at separate use cases. Every section that follows the authentication (ref. 4.7) belongs to the protected part of the application.

4.1 CodeIgniter4

CodeIgniter is an Application Development Framework - a toolkit - for people who build websites using PHP [18]. It is a small framework that claims to offer great performance. It is aimed at development under the MVC pattern but does not enforce it. The framework can have its core parts easily extended or completely replaced to make the system work as needed. In short, CodeIgniter is a malleable framework that provides the tools needed while staying out of the way. It does not require the use of the command line, making all the features available immediately with almost no additional setup. It is well documented [19], has a sizable community, and is actively maintained.

At the time of development, the latest release version and the version the application is implemented in was CodeIgniter4.

Several other frameworks were considered before deciding on CodeIgniter, more specifically Laravel and Symfony. Both of them are popular and excellent choices. Feature-wise, the most complete framework out of the three is Laravel. However, not by far, and as this was my first PHP project, the easiest-to-learn framework was selected. CodeIgniters' documentation was also the personally most readable.

Codeigniter4 is prepared for development beneath the MVC pattern. Due to this, the project's structure is separated into three distinct components (ref. 2.7). The majority of the code lives inside the *App* directory:

- Config - all files that configure the core application's behavior.
- Controllers - files serving as controllers from MVC.
- Database - contains database migrations and seeding. Seeding was not used.
- Entities - classes representing data from the database. They allow for better type safety and simpler workflow, providing magic methods for accessing their data along with their typecasting.
- Filters - files with logic that can be applied before or after visiting any of the application's routes.
- Libraries - shared logic.
- Models - files that serve as models from MVC.
- Validation - custom rules for verifying user inputs.
- Views - this folder contains all of the HTML files.

The project's entry point is a file named index.php. It is located inside the public directory. The public directory also contains assets, CSS and JavaScript files, and the temp and uploaded folders. Temp is where all the uploaded files that were not yet assigned live. After being assigned, those files are moved to uploads. Uploads are categorized into folders by their material ids, each material having one such directory.

The last directory worth mentioning is writable. Writable contains all of the logs, cache, and session files. More on the usage of the cache will be talked about later (ref. 4.5).

4.1.1 Setup and Updating

Before deploying CodeIgniter4, it is imperative to verify that the server can run PHP 7.4 or newer. The *intl*, *mbstring*, *json* extensions must be enabled. The application uses the provided Image class to manipulate thumbnails, so a *imagick* must be enabled too. The application must be configured using a *.env* file in the framework's root directory or directly inside the *App/Config* folder.

The first obstacle to the creation of the application came in the form of no way of configuring a new virtual host on the deployment machine. Virtual hosts are the preferred way of using CodeIgniter or any web-based application. To circumvent this, a new *.htaccess* file was created. This file reroutes all traffic that does not lead to a file from the root folder to the public folder's index, the application's runner. The new file also prevents unwanted access to the root directory. Be wary that it will not work whenever URL rewriting is disabled.

Running the application is as simple as renaming the applications' folder to be the same as the URL used for accessing it and modifying the env file with the desired database credentials and base applications URL. Before deploying the application, it is recommended to change the environment to production from development to hide debugging tools and error messages.

Sometimes it is desirable to update the framework to a newer version. Be warned that in doing so, incompatibilities may be introduced to the application. Updating can be done with either the composer or manually. In both cases, the changes to the *App* must be done by following the patch notes. New versions of this directory are not applied automatically and must be merged by hand or with specialized merge tools.

4.2 Models

The very first step in the application's implementation is the creation of entities and their models. This is an easy process due to the knowledge from the ERD model (see Fig. 3.2), mostly copying the structure but including the connectors directly in their intended entity.

Several custom methods are included in the resource and material entities. Those methods streamline their use in the controllers and views. The resource entity's main methods are getDefaultImage(), getURL(), and getRootPath(). The default image method retrieves the path to the default image from the database as a new entity. The other two methods work on the property's saved data, transforming them into the desired outcome with the URL being a valid web reference and the root path a relative path of the resource from the projects' root directory. The material entity then provides getter methods for its resources. Those resources are saved in a shared array, and the methods retrieve only the specified type.

Codeigniter provides a base model that can be extended to simplify the usage of the database. Each extension must specify the targeted table's name, its primary key, and the fields it can modify. Not all fields must be present, which disables their modification by the model. There is one model for each of the databases' tables.

4.3 Controllers and routing

Next came the creation of controllers. Each entity has its controller. Controllers are separated into two namespaces, public and administration. This is merely an organizational separation. Nonetheless, it is used to better discern their responsibilities when routing. Every public method has a route associated with it. The routes can either point to a viewable page or be an AJAX endpoint. All routes beginning with */admin/* are protected by an authentication filter (ref. 4.7).

The AJAX endpoints are targeted for use by the clients' javascript. It should be noted that the application supports Cross-site Request Forgery (CSRF) protection. CSRF is an attack that forces authenticated users to submit unwanted actions. Every submitted request must come with a CSRF token to be protected against this attack. The server verifies the token, and the request is approved if valid. Every submission changes the current token. Tokens can be saved as a unique input field in every form and loaded with the view. This is fine for standard forms, but when using AJAX, each submission invalidates the current token but does not replace it with a new one. This makes default AJAX unusable. All AJAX routes are grouped under the */ajax/* route to prevent this behavior. The route segment causes the application to append the current CSRF token to the AJAX response, allowing the replacement of all tokens currently present on the page.

Each controller provides an index method. The index, in most cases, presents the user with a view containing all accessible data of the related entity.

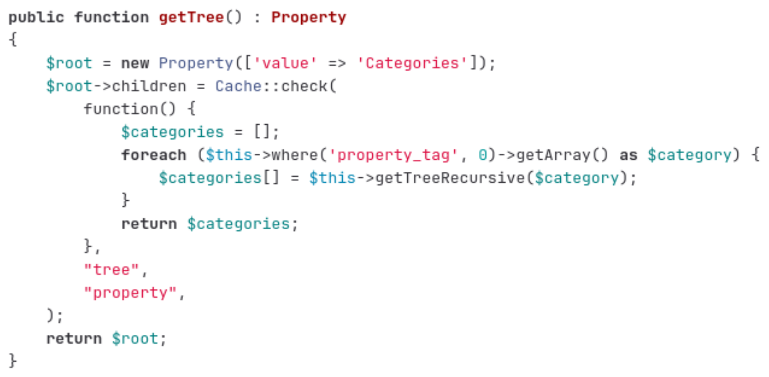
4.4 Views

The views component is the last part of the application, developed alongside the controllers. Views contain all of the applications' HTML code and some of the JavaScript. The HTML is saved in .php files and interlaced with PHP code. The framework then executes the embedded code before handing the result to the user. Due to that, it is possible to work with the entities directly inside the HTML. This is both more type-safe and pleasant than passing only basic data types, like strings and integers, across the views.

The views follow the same division into the public and protected sections as seen in controllers. They are further separated by their entities. Again, this is purely organizational. Most of the views were created as includable components; the rest are either layout or extend some layout.

4.5 Properties as filters

From the very start the properties were concieved as filters. To allow their easy handling inside the views, it is necessary to transform them from an array into a tree. This can be achieved by taking those properties whose tag is 0. For each such property, we find all of their children. Children are properties whose tag equals the current id. This process can be repeated until no such properties exist, meaning we arrived at a leaf.

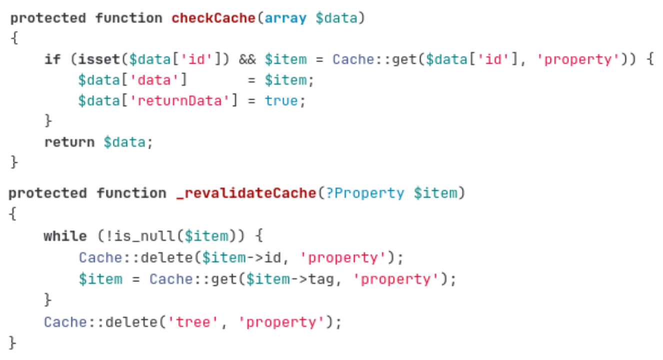


```php
public function getTree() : Property
{
    $root = new Property(['value' => 'Categories']);
    $root->children = Cache::check(
        function() {
            $categories = [];
            foreach ($this->where('property_tag', 0)->getArray() as $category) {
                $categories[] = $this->getTreeRecursive($category);
            }
            return $categories;
        },
        "tree",
        "property",
    );
    return $root;
}
```

Figure 4.1: Function to get all properties as a tree

The whole process requires an large amount of queries. To reduce the strain on the database, we can cache the properties and the resulting tree. Cache can be checked before each query asking for a single material. To ask for the tree, a separate cache call is made when getting the tree itself. There are, however, complications associated with caching, mainly if the cache is saved for longer periods of time. The cache has priority over the database, meaning that changes will not be projected until the cache is either revalidated or it expires. Every update insert, update and delete call must be accompanied with a cache revalidation. It is not necessary to clear the whole cache. Only the changed property and its chain of parents along with the tree file need to be cleared.

To fulfill the request to include a possibility of adding mouseover descriptions to the properties, a native HTML tooltips solution was used. It works by adding a *title* attribute to any element, which properties can use to display their description.



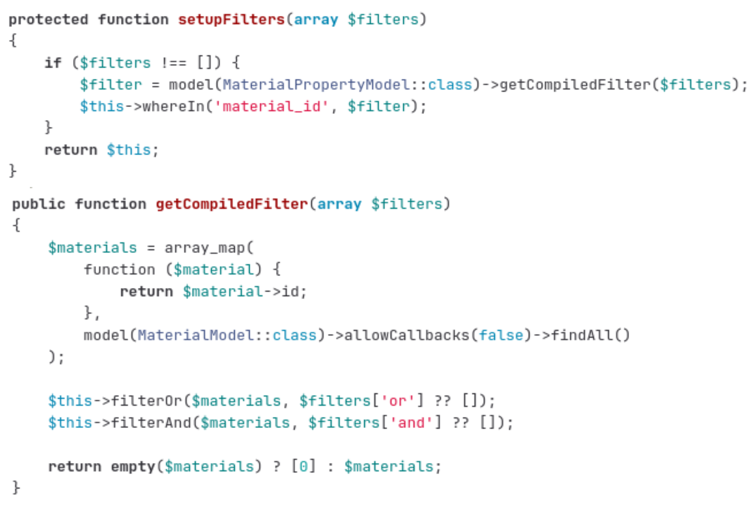
```
protected function checkCache(array $data)
{
    if (isset($data['id']) && $item = Cache::get($data['id'], 'property')) {
        $data['data']       = $item;
        $data['returnData'] = true;
    }
    return $data;
}

protected function _revalidateCache(?Property $item)
{
    while (!is_null($item)) {
        Cache::delete($item->id, 'property');
        $item = Cache::get($item->tag, 'property');
    }
    Cache::delete('tree', 'property');
}
```

Figure 4.2: Loading and revalidation of cache in property model

4.6 Filtering materials

With property tree is available, it is very easy to recurse over its contents and steadily build a filter caret. The filtering module is prepared with ease of use in mind (see Fig. 3.3). Each property can be checked. If the property is a category, this action collapses its subtree and internally selects all of the values as a valid value for matching. In other words it groups the properties in a *where in* part of the query. Properties that are not under a selected tree must all be matched. Aside from selecting the property, its subtree can be collapsed manually. This functionality is added to reduce the verticality of the filters.

In addition to using properties, the materials can be filtered by their titles. The search is case insensitive, but not accent-insensitive. To improve upon it this, an accent-insensitive suggestion system was added. The system uses a list of all available materials that is loaded with page. Search suggestions are often done by fetching the results directly from the server. However it was observed that such implementation seems unresponsive on the deployment server, sometimes taking seconds to display any results, which is why this alternative was chosen. If it proves a problem in the future, it can be changed fairly easily by replacing result fetching function with a server query.

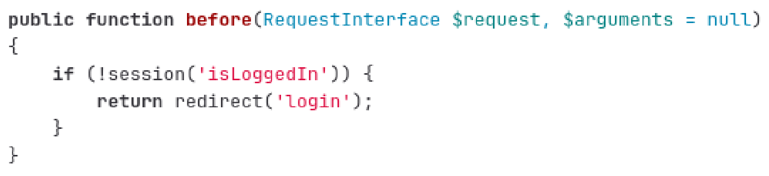


```
protected function setupFilters(array $filters)
{
    if ($filters !== []) {
        $filter = model(MaterialPropertyModel::class)->getCompiledFilter($filters);
        $this->whereIn('material_id', $filter);
    }
    return $this;
}

public function getCompiledFilter(array $filters)
{
    $materials = array_map(
        function ($material) {
            return $material->id;
        },
        model(MaterialModel::class)->allowCallbacks(false)->findAll()
    );

    $this->filterOr($materials, $filters['or'] ?? []);
    $this->filterAnd($materials, $filters['and'] ?? []);

    return empty($materials) ? [0] : $materials;
}
```

Figure 4.3: Filtering of materials: (1) call in model (2) call in connector

4.7 Authentication

The administration is protected by a password based authentication. This authentication uses CodeIgniters' sessions and filters to keep users logged in. All of the protected routes begin with */admin*. When visiting such a route, a filter is applied to verify if the user is logged in by consulting the currently active session. This means no authentication is required until the user logged out or the session was invalidated.



```
public function before(RequestInterface $request, $arguments = null)
{
    if (!session('isLoggedIn')) {
        return redirect('login');
    }
}
```

Figure 4.4: Administration protection

To prevent users from returning to login page when they are already logged in, another filter is applied. This filter automatically redirects them to the landing page of the administration section upon encountering a logged in user.

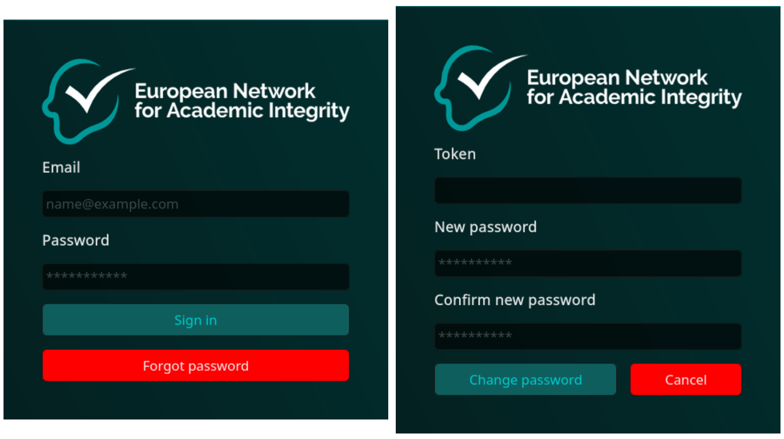


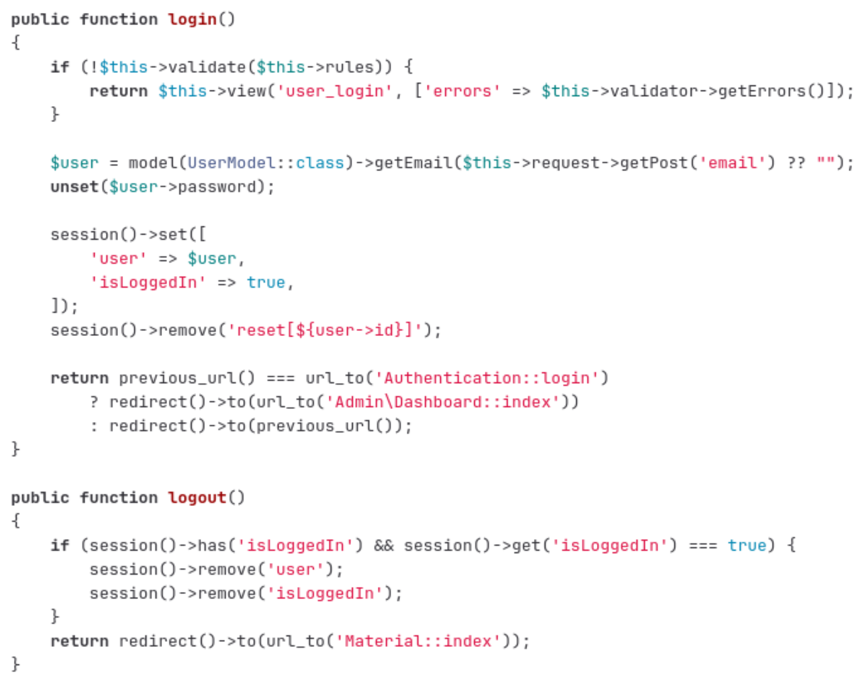
Figure 4.5: Login design and password reset design

The login process starts with the validation of the inputted email and password. The email is first used to find a single user. If found then tries to verify the saved hashed password with the inputted one. If the user is not found or the password not matched, returns to the login page and displays a message for the failed step. If the validation was successfull, saves the user inside the session and marks him as logged in, continuing to administration. When logging out, the saved session data are destroyed.

The application allows users to reset their password. The process consists of sending the token to the entered email, redirecting user to the reset password page and waiting for user to enter the token and new password. The token is checked with the token saved into the session, and if valid, sets the password to the new one.

4.8 Statistics

The application saves usage statistics for each material. Upon visiting a material by a non-logged in user, the materials views are incremented and control passed to the Views model. The model then verifies if the material was visited in current server time day. If it was, then increments the count. If not, then inserts a new record with count set to one. This enables the application to keep track of daily views.



```php
public function login()
{
    if (!$this->validate($this->rules)) {
        return $this->view('user_login', ['errors' => $this->validator->getErrors()]);
    }

    $user = model(UserModel::class)->getEmail($this->request->getPost('email') ?? "");
    unset($user->password);

    session()->set([
        'user' => $user,
        'isLoggedIn' => true,
    ]);
    session()->remove('reset[${user->id}]');

    return previous_url() === url_to('Authentication::login')
        ? redirect()->to(url_to('Admin\Dashboard::index'))
        : redirect()->to(previous_url());
}

public function logout()
{
    if (session()->has('isLoggedIn') && session()->get('isLoggedIn') === true) {
        session()->remove('user');
        session()->remove('isLoggedIn');
    }
    return redirect()->to(url_to('Material::index'));
}
```

Figure 4.6: Authentication controller login and logout

Another useful statistic is served by the Ratings model. Each user can have up to one rating for each material. The model can be used to retrieve the recent most favorite materials by grouping the ratings by update date and id of the material.

The administration uses those statistics on the dashboard page. Dashboard is used to render daily and total visits or most viewed materials, both recent and all-time. The statistics include the blame counter, keeping track of all last updates on materials, and their relevant users. The dashboard also provides shortcuts to the last updated and last published materials.

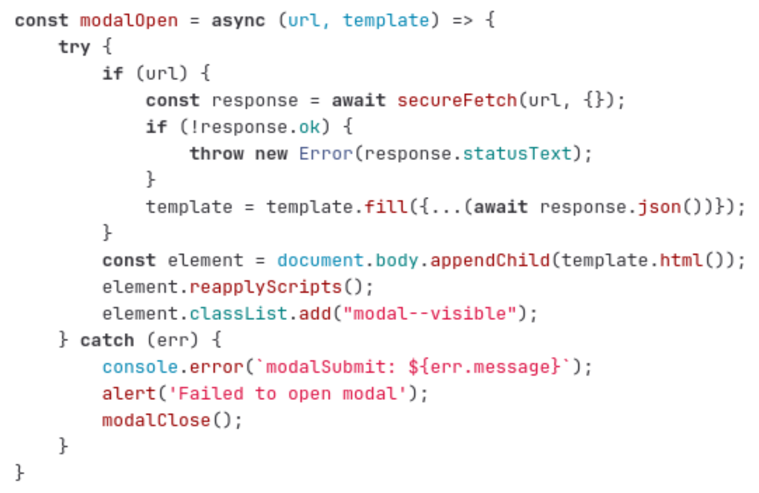
4.9 Creating or updating data

The administration was designed to be as resuable as possible. To that end, all of the manageable data use the same tabular layout. This layout is a redesign of the public one for viewing all materials. To allow easier management, each of the displayed entities can be created, edited or deleted directly in the view. The only exception to the creation and updating is the material entity, more elaborated upon in its own section (ref. 4.10).

For other entities, the creation and updating is done by way of modals. The modal is an HTML element that is displayed over the current content. Each has a title specifying the current action and a content that represents that action. This approach allows easy integration into other views.

The application's modals all use the same JavaScript code contained in the *modal.js* file. Every modal is opened by using the modalOpen (see Fig. 4.7) function. The function transforms the modal template from string to a HTMLElement. If the string contains scripts, it is necessary to reapply them, since the conversion marks them as disabled and does not run them. The funtion also fetches the data and fills the template if requested. The filling is done to allow the use of the function for both creation and editing.

Submission of the modal is handled by modalSubmit, a function that is an abstraction of sending the form data as a asynchronous request to the servers' controllers. The controllers then receives the request, verifies its contents and processes it. Then it sends either an



```
const modalOpen = async (url, template) => {
    try {
        if (url) {
            const response = await secureFetch(url, {});
            if (!response.ok) {
                throw new Error(response.statusText);
            }
            template = template.fill({...(await response.json())});
        }
        const element = document.body.appendChild(template.html());
        element.reapplyScripts();
        element.classList.add("modal--visible");
    } catch (err) {
        console.error(`modalSubmit: ${err.message}`);
        alert('Failed to open modal');
        modalClose();
    }
}
```

Figure 4.7: JavaScript function for opening modals

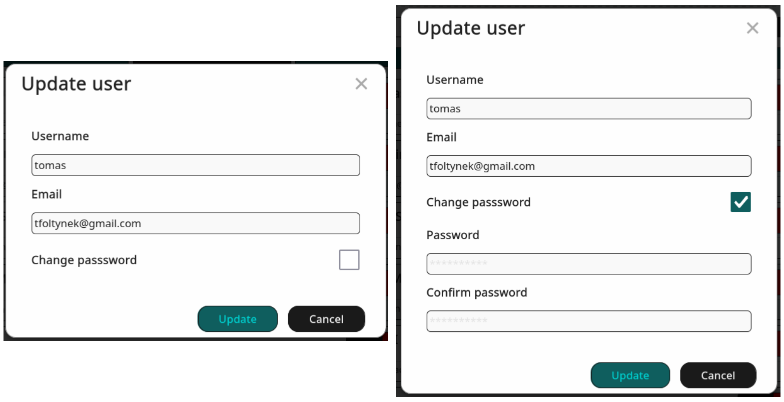
error or the result of the operation back to the client. In case of failure the error message is displayed in the modal, otherwise the modal is closed and the view updated with the response.

Every administrator is allowed to modify, create, and delete other users (see Fig. 4.8). An administrator cannot delete his own account. To prevent the unwanted modification of password while editing an account, a checkbox is added. This checkbox enables password format verification and adds the password to the database call. The password inputs are always present in case of new account.

A property can belong to a category. To achiveve a user friendly management, a list of categories is loaded when opening the property editor (see Fig. 4.9). The category input field can then either be empty or one of the listed categories, which is verified on each change of the field. The form also supports the addition of description or priority.

4.10 Creating or updating material

Material management is the most important feature of the application, but also the most complex. As such, it does not use a modal as its front-end view, it is a separate page. This page contains editable title, status, properties, content, thumbnail and files, links, and relations.

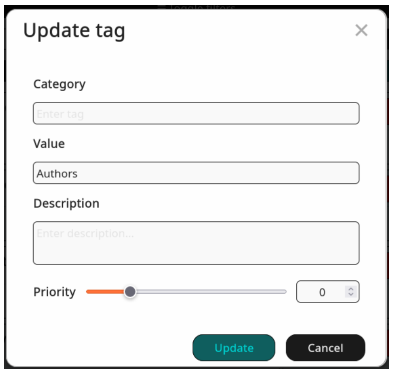

Update user

Username
tomas
Email
tfoltynek@gmail.com
Change passsword ☐

Update user

Username
tomas
Email
tfoltynek@gmail.com
Change passsword ✓
Password
Confirm password

Figure 4.8: User editor modal: (1) password hidden, (2) password shown


Update tag

Category
Enter tag
Value
Authors
Description
Enter description...
Priority 0

Figure 4.9: Property editor modal

It uses a third-party text editor [20] for the editing of content. The relations can be added by selecting an existing material from a loaded list of materials. Relations are implemented bi-directionally. Adding a relation will display it in both materials, and removing it will remove it from both, too.

Properties can be assigned to the material using a property selector (see Fig. 4.10). Property selector by default shows all currently assigned properties, but it can be unlocked to allow modifications. When the user unlocks it, the selector shows all properties of a single category, which can be toggled on and off. It shows a single category to prevent information overload, other categories are hidden. To change to a different category, a previous and next controls are added.

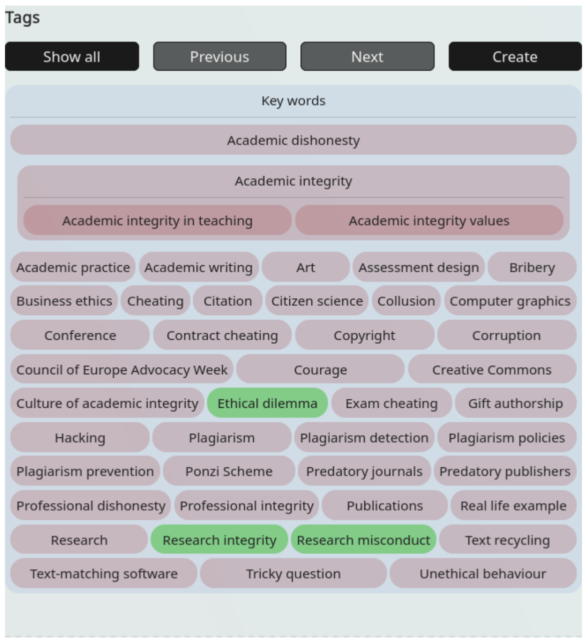
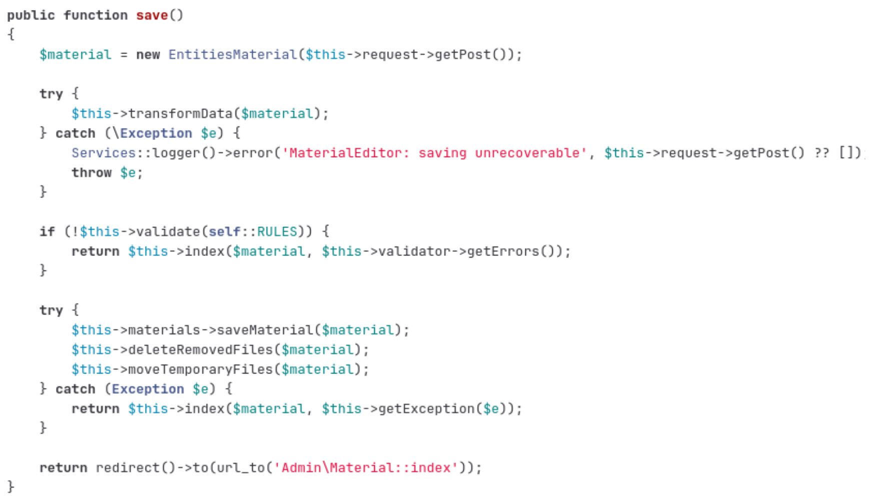


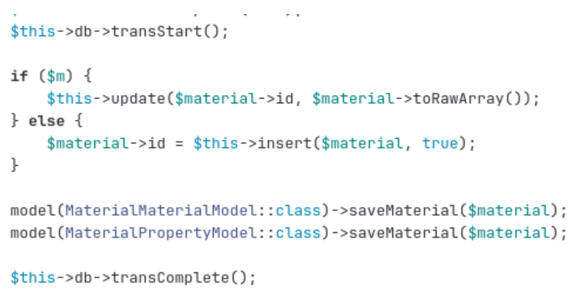
Figure 4.10: Unlocked property selector

The editor is controlled by a specialized controller. This controller handles the verification of inputs, their transformation from post data into material entity and the follow up saving. The saving is an extensive operation. The first step is the saving of material into the database. This is enclosed in a transaction, and rollbacked should any part fail. During the transaction, the materials properties and relations are saved too. After the transaction is complete, the newly unused resources, given by a special post array, are unassigned. All operations are done by a specialized library that handles all resource specific operations. In case of the resource being a file or thumbnail the unassignement is different for temporary and assigned resources. The temporary ones are deleted completely, while the assigned ones are removed from database and moved into a separate folder. In case of failure an error message is returned and added to the editor view.



```
public function save()
{
    $material = new EntitiesMaterial($this->request->getPost());

    try {
        $this->transformData($material);
    } catch (\Exception $e) {
        Services::logger()->error('MaterialEditor: saving unrecoverable', $this->request->getPost() ?? [])
        throw $e;
    }

    if (!$this->validate(self::RULES)) {
        return $this->index($material, $this->validator->getErrors());
    }

    try {
        $this->materials->saveMaterial($material);
        $this->deleteRemovedFiles($material);
        $this->moveTemporaryFiles($material);
    } catch (Exception $e) {
        return $this->index($material, $this->getException($e));
    }

    return redirect()->to(url_to('Admin\Material::index'));
}
```

Figure 4.11: Function for saving materials from controller



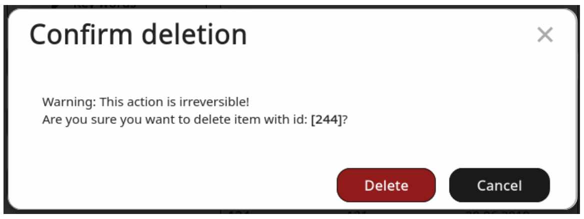
```
$this->db->transStart();

if ($m) {
    $this->update($material->id, $material->toRawArray());
} else {
    $material->id = $this->insert($material, true);
}

model(MaterialMaterialModel::class)->saveMaterial($material);
model(MaterialPropertyModel::class)->saveMaterial($material);

$this->db->transComplete();
```

Figure 4.12: Transaction for saving material

From the previous description, it can be seen that the material saving does not handle the uploading of files, but operates only on already uploaded ones. Files, including thumbnails, are uploaded directly on the material editor page by using asynchronous JavaScript. The uploads happen everytime a user opens the file selector and submits it, after which the file is passed to the controller, verified and the library stores it in the temporary directory. The client then receives the response in form of a new resource. The resource is then added to the page, either replacing thumbnail or appended to the list of attached files.

4.11 Deletion of data

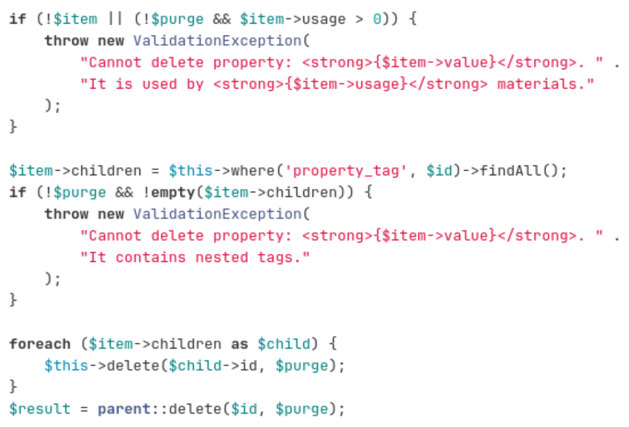
The deletion interface is shared between all entities. Each deletable element contains a deletion button. The button shows a confirmation modal upon interaction (see Fig. 4.13). This modal is shared for all items in the view. Due to that the id of the item to be deleted is loaded into the modal after each such interaction. The deletion action can be set to the model during this time too, however in the application's case, it is always hardcoded into the view. Each entity has its own deletion method inside its own controller. Those endpoints usually call the delete method of the model, however there are two special cases.


Confirm deletion

Warning: This action is irreversible!
Are you sure you want to delete item with id: **[244]**?

Figure 4.13: Modal used for deletion

First is the deletion of properties. There are two condition imposed. If the property is a category, meaning it has at least one subproperty, or if it is used in any material, it cannot be deleted and the user is notified with corresponding error message. The verification is done by way of overriding the property models default deletion method (see Fig. 4.14).



```
if (!$item || (!$purge && $item->usage > 0)) {
    throw new ValidationException(
        "Cannot delete property: <strong>{$item->value}</strong>. " .
        "It is used by <strong>{$item->usage}</strong> materials."
    );
}

$item->children = $this->where('property_tag', $id)->findAll();
if (!$purge && !empty($item->children)) {
    throw new ValidationException(
        "Cannot delete property: <strong>{$item->value}</strong>. " .
        "It contains nested tags."
    );
}

foreach ($item->children as $child) {
    $this->delete($child->id, $purge);
}
$result = parent::delete($id, $purge);
```

Figure 4.14: Deletion process of properties

Second is the deletion of materials. Each material is connected to a multitude of data, however the database uses foreign keys to service its deletion, handling all of the connections inside the database itself. Even with that being the case, there is a need to remove the attached files and thumbnail manually. This step is done before the removal of the material from the database.

4.12 Testing

The application's testing was done manually. No unit or other tests were written. While it is usually preferred to automatize the testing, it was not prioritized as the creation of tests is a time-consuming process, and their task could be mostly covered by the follow-up user testing. Extensive user, both public and administration, testing was done on both the pre-release and the release versions.

A randomized bot user test was considered for a while but not used. It can be presumed that administrators will not act maliciously in the administration section. If the application breaks due to random user behavior in the public section, it will not cause any real issues, being recoverable by a simple refresh of the page.

5 Conclusion

The thesis aimed to analyze, design and reimplement an educational materials database WordPress plugin into a standalone web application explicitly tailored to the needs of ENAI.

The introduction first elaborated on the context of the project, helping readers get into the problem. The next part looked at the existence of premade systems that would be possible to use in place of the new implementation. However, they were either too focused on the management of complete courses, were general content management systems, or handled Wikipedias, none of which fulfilled the application's intended functionality.

With the knowledge from the introduction, an analysis of the plugin was made. The analysis focused on finding the exact functionality of the plugin, and its interface and data flow much more than on its code. This way of research produced formal requirements for the application, which were then visualized in a use case diagram and used in a data model. With both the use and data model known, an abstract UI layout was constructed for most of the necessary components. This design was then implemented.

The implementation uses a PHP framework [19] as its back-end, greatly simplifying and speeding up the development process. The framework operates on an MVC design pattern, which significantly increases the maintainability and modularity of the application.

The product of the work is a new web application that provides easy-to-use management of an educational database. While the application is thematically customized for ENAI, it is freely available for use and modifications [2] under the MIT license.

5.1 Contributions of the work

The work produced an open-source web application designed specifically for the presentation and management of educational materials.

Its primary and intended contribution is its deployment to the ENAI web server, which will run until it once again becomes obsolete. This server is already quite known, and the new application may help it grow further.

Systematic support of views and ratings should, in theory, help with improvements to the material's quality. The application also supports a virtually unlimited level of tag hierarchies, only limited by the complexity of the presentation to the user. This is a direct upgrade from the old implementation, which was impossible. This upgrade should allow easier management of tags by grouping them together and introduce an easier way of searching for whole categories. Another improvement is the quicker and easier management of files, reducing the complexity of several steps into one.

The application is deployable with minimal setup. It also offers a setup guide. It being open-source enables anyone to download it, possibly customize and then deploy it to their own server, all free of charge. The introduction spoke about a shortage of applications of this kind, which means that their number might increase when no entry costs are required.

5.2 Possible future improvements

The application meets the criteria stated at the start of the development. Nevertheless, there is always a possibility for improvements.

For example, the support for multi-lingual materials is done by bi-directional relations, which could be massively improved by implementing material collections. Those collections could be the parents of each material with properties attached to them instead. Those collections could then share their tags with their materials, allowing for easier management.

Another possible improvement would be to add support for more languages on the level of the whole application. Some parts already, like validation, already support such a notion. However, only some do. If a translation were to be provided, minor changes would be required across the whole application, but a framework-supported localization could then be used.

As a last example, an algorithm for automatically finding similar relations could be implemented. This operation is currently done by hand and is mixed with language support.

5.3 Evaluation

The application was created in a span of seven months with over five hundred commits recorded on git. The main bulk of the work came in the later part of development. While I do not have a precise measurement of the time spent on the project, it is at the very least three hundred hours. The costs would undoubtedly accumulate was the project a paid one. The project is my first web application, having learned all of the languages and principles during its development. The application has come out well in most aspects.

Bibliography

1. *Interactive database of educational materials* [online]. European Network for Academic Integrity, 2023 [visited on 2023-04-01]. Available from: `https://www.academicintegrity.eu/materials`.
2. *Interactive database of educational materials* [online]. Brno: Jan Martinek, 2023 [visited on 2023-05-17]. Available from: `https://github.com/Razzoved/educational-database-bp`.
3. *Usage statistics and market share of WordPress* [online]. Q-Success, 2009/2023 [visited on 2023-03-26]. Available from: `https://w3techs.com/technologies/details/cm-wordpress`.
4. *About ENAI* [online]. European Network for Academic Integrity, 2023 [visited on 2023-04-01]. Available from: `https://www.academicintegrity.eu/wp/about-enai/`.
5. LANG, Leoš. *Implementace e-learningového modulu pro systém Wordpress* [online]. Brno, 2018 [visited on 2023-03-10]. Available from: `https://theses.cz/id/eybogi/`. Mendelova univerzita v Brně.
6. IMED BOUCHRIKA, Phd. *What Is eLearning? Types, Advantages, and Drawbacks* [online]. Research.com, 2022/2023 [visited on 2023-03-10]. Available from: `https://research.com/education/what-is-elearning`.
7. HORTON, William. *e-Learning by Design*. 2nd ed. John Wiley & Sons, Inc., 2011. ISBN 978-0-470-90002-4.
8. MAUTHE, A.; THOMAS, P. *Professional Content Management Systems: Handling Digital Media Assets*. 1st ed. Chichester: John Wiley & Sons, Inc., 2004. ISBN 978-0-470-85542-3.
9. WELLING, L.; THOMSON, L. *Mistrovství PHP a MySQL*. 1. vyd. Přel. BAŠE, Ondřej. Brno: Computer Press, 2017. ISBN 978-80-251-4892-1.
10. *Moodle* [online]. Moodle HQ, 2023 [visited on 2023-05-12]. Available from: `https://moodle.org/`.
11. *Open edX* [online]. Axim Collaborative, 2023 [visited on 2023-05-12]. Available from: `https://openedx.org/`.

12. *Canvas LMS* [online]. Instructure Inc., 2023 [visited on 2023-05-12]. Available from: `https://github.com/instructure/canvas-lms`.

13. *Overview of web technologies used by Forpsi.com* [online]. Q-Success, 2023 [visited on 2023-05-15]. Available from: `https://w3techs.com/sites/info/forpsi.com`.

14. *Omeka* [online]. Digital Scholar, 2023 [visited on 2023-05-12]. Available from: `https://omeka.org/`.

15. GILMORE, W. Jason. *Easy PHP Websites with the Zend Framework*. 1st ed. Columbus: W.J. Gilmore, LLC, 2009. ISBN 978-0615303888.

16. HAMM, Matthew J. *Wireframing Essentials: An introduction to user experience design*. 1st ed. Birmingham: Packt Publishing, 2014. ISBN 978-1-84969-854-2.

17. *Global mobile traffic worldwide 2022 1stQ* [online]. Hamburg: Statista GmbH, 2022 [visited on 2023-05-12]. Available from: `http://web.archive.org/web/20230512021013/https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/`.

18. *CodeIgniter* [online]. CodeIgniter Foundation, 2019/2023 [visited on 2023-04-01]. Available from: `https://www.codeigniter.com`.

19. *CodeIgniter4 User Guide* [online]. CodeIgniter Foundation, 2019/2023 [visited on 2023-04-01]. Available from: `https://www.codeigniter.com/user_guide/`.

20. *TinyMCE* [online]. Tiny Technologies Inc, 2023 [visited on 2023-05-11]. Available from: `https://www.tiny.cloud/`.

A Source Code

Full source code of the application can be found in the attachments or in the GitHub repository:

`https://github.com/Razzoved/educational-database-bp`.