Extending Sapphire: Developing and Documenting an Open-Source Online Course Management and Grading System

Duaa Nakshbandi

Extending Sapphire: Developing and Documenting an Open-Source Online Course Management and Grading System

Duaa Nakshbandi

Bachelor's Thesis

to achieve the university degree of

Bachelor of Science (BSc)

Bachelor's Degree Programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 07 Aug 2024

Erweitern von Sapphire: Weiterentwicklung und Dokumentation eines Open-Source Online-Kursverwaltungs- und Benotungssystems

Duaa Nakshbandi

Bachelorarbeit

für den akademischen Grad

Bachelor of Science (BSc)

Bachelorstudium: Software Engineering and Management

an der

Technischen Universität Graz

Begutachter

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 07 Aug 2024

Diese Arbeit ist in englischer Sprache verfasst.

Abstract

This thesis describes extensions and improvements made to Sapphire, an open-source, web-based platform for course management and grading. The system uses web technologies such as HTML, CSS, JavaScript, Ruby on Rails, and PostgreSQL to provide a scalable and maintainable system architecture.

Two major technical contributions made to the system are the implementation of file and folder renaming, and the implementation of drag-and-drop functionality for moving files and folders. Another important contribution is the provision of comprehensive technical documentation and tutorials to assist future developers and administrators in maintaining and extending the Sapphire system.

The code contributions made in the scope of this thesis significantly enhance the usability, functionality, and efficiency of the platform for all its users, including students, tutors, lecturers, and administrators. The detailed tutorials serve as a resource for future Sapphire developers and administrators.

Kurzfassung

Diese Arbeit beschreibt die Erweiterung und Verbesserung von Sapphire, einer open-source, webbasierten Plattform für Kursmanagement und Notengebung. Das System nutzt Webtechnologien wie HTML, CSS, JavaScript, Ruby on Rails und PostgreSQL, um eine skalierbare und wartbare Systemarchitektur zu schaffen.

Zwei wichtige technische Beiträge sind die Implementierung der Datei- und Ordnerumbenennung und die Implementierung der Drag-and-Drop-Funktionalität zum Verschieben von Dateien und Ordnern. Ein weiterer wichtiger Beitrag ist die Schaffung einer umfassenden technischen Dokumentation und Tutorials, die zukünftige Entwicklerinnen und Entwickler sowie Administratorinnen und Administratoren bei der Wartung und Erweiterung des Sapphire-Systems unterstützen.

Die Code-Beiträge im Rahmen dieser Arbeit verbessern die Benutzerfreundlichkeit, die Funktionalität und die Effizienz der Plattform für alle Benutzerinnen und Benutzer, einschließlich Studentinnen und Studenten, Tutorinnen und Tutoren, Lehrkräfte und Administratorinnen und Administratoren, erheblich. Die detaillierten Tutorials dienen als Ressource für künftige Sapphire-Entwicklerinnen und Entwickler sowie Administratorinnen und Administratoren.

Contents

List of Figures

List of Listings

Acknowledgements

I am deeply grateful to Professor Keith Andrews for tirelessly reviewing my thesis, giving me an abundance of knowledge, opening my eyes to academic best practices, and greatly helping me refine my academic writing techniques. His guidance went beyond this thesis's scope in providing me with the opportunity to work as a teaching assistant in his Human-Computer Interaction (HCI) course. This experience greatly strengthened my understanding of HCI, which proved invaluable while working on this thesis and has been a great complement to my academic pursuits.

I would also like to extend my sincere thanks to: Amel Smajic, who introduced me to Sapphire and gave me training on its functionality, Matthias Link for his thorough review of my code contributions, and my family for their unwavering support during my studies. Finally, I am very grateful to all my kind colleagues at the Graz University of Technology for their support.

Duaa Nakshbandi

Graz, Austria, 07 Aug 2024

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using Keith Andrews' skeleton thesis [Andrews 2021].
- Figure 4.1, which shows Sapphire's software architecture, and Figure 4.2, which shows Sapphire's data model, were extracted from Link [2021] and are used under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) license.

Chapter 1

Introduction

"Begin at the beginning and go on till you come to the end; then stop."

[Lewis Carroll, Alice in Wonderland]

This thesis describes the development work and code contributions made to Sapphire, an open-source, web-based platform for course management and grading [Link and Kriechbaumer 2024a] during the period from Aug 2022 to Jul 2024. It provides reference material for future developers and Sapphire administrators tasked with maintaining Sapphire, capturing the knowledge and insights gained during the thesis period. The objective of this thesis project was to enhance usability and efficiency for all users of the system.

The first part of the thesis sets the context for the work. Chapter 2 describes the various web technologies relevant to Sapphire. In Chapter 3, Moodle, a widely used open-source course management system, is presented and briefly compared with Sapphire in terms of grading. Chapter 4 outlines Sapphire's story, including the motivations behind it, its development history, and its architecture.

The second part of the thesis details the implementation and technical documentation performed during this work. Chapter 5 discusses the technical implementation of extensions and improvements made to Sapphire, including file and folder renaming and drag-and-drop functionality for moving files and folders. Chapter 6 provides a number of in-depth tutorials helpful for the maintenance and further development of Sapphire. Finally, Chapter 7 outlines some ideas for future development of the system.

Chapter 2

Web Technology

"Technology like art is a soaring exercise of the human imagination."

[Daniel Bell]

The World Wide Web (or simply web) is one of the most significant services that run on the Internet. It consists of millions of web servers, accessed by web clients or browsers, and indexed and made searchable by search engines. The web builds upon several basic concepts:

- *The Internet*: According to the discussion in Tanenbaum et al. [2020], the Internet is a worldwide network of heterogeneous computer networks used for data exchange. It is based on the TCP/IP network protocol [Cerf and Kahn 1974]. Through this technical infrastructure, various communication and information services can be offered, including email, ssh, file transfer, peer-to-peer messaging, telephony, and the web. For general information about the Internet, the Internet Society's website [ISOC 2023] can be consulted.

- *URL*: URL stands for Uniform Resource Locator. It is a unique identifier for a web page which makes it addressable. The URL first specifies the access method, which for web URLs is `http` or `https`. This is followed by the domain name of the web server and a domain-specific path to the web page. The syntax of Uniform Resource Identifiers (URIs), which includes URLs, was defined by the IETF [Berners-Lee et al. 2005].

- *HTTP*: HTTP stands for Hypertext Transfer Protocol. It defines the communication between the web browser and web server and is a layer on top of TCP/IP. The web browser requests a web page from a web server using an HTTP request containing a URL. If the request is accepted, the web server responds with an HTTP response containing the requested web page in HTML. The official documentation of HTTP, published by the Internet Engineering Task Force (IETF), defines the HTTP/1.1 protocol, which is used in the client-server model for web applications [Fielding et al. 1999].

  HTTP/2 [Thomson and Benfield 2022] and HTTP/3 [Bishop 2022] are successive versions of the HTTP protocol designed to improve the performance of web communications. HTTP/2 improves speed through field compression and multiplexing, while HTTP/3 uses the QUIC transport protocol [Iyengar and Thomson 2022] for even greater performance gains.

- *The World Wide Web*: The World Wide Web is a network of millions of web servers spread all over the world. Web servers are accessed by web clients, also called browsers, speaking HTTP. Search engines help users locate information held on particular web servers.

- *Client-Server*: The term client-server describes the relationship between two computer programs. One program, the client, requests a service from the other program, the server. The server then fulfills the request. In a network setting, the server typically runs continuously on one computer and waits for requests from clients running on other computers.

- *Web Server*: A web server runs continuously and responds to requests from web clients by sending the corresponding web page back to the web browser as a response.
- *Web Browser*: A web browser is a web client used to access web pages on a web server over HTTP. The browser then parses the web page's HTML, CSS, and JavaScript and displays the resulting page to the user.
- *IP Address*: IP stands for Internet Protocol. An IP address is a unique numerical label assigned to each device connected to a computer network using the Internet Protocol for communication. It serves two main functions: identifying the host or network interface, and providing the location of the host in the network.

  IPv4 [USC 1981], the fourth version of the Internet protocol, is the most widely deployed protocol. IPv4 addresses are 32 bits in length, for example `93.184.216.34`. Since almost all IPv4 addresses have been allocated, IPv6 [Hinden and Deering 1998] was introduced, whose addresses are 128 bits in length, for example `2606:2800:220:1:248:1893:25c8:1946`. IPv6 offers a significantly larger address space, enabling many more devices to connect to the Internet.
- *Domain Name*: A domain name is a human-readable and memorable label for an IP address, for example, `example.com`. It serves as an address for a website on the Internet and is used to identify the web server on which a website is hosted. The domain name is translated into an IP address through the process of Domain Name System (DNS) resolution [Ellingwood 2020].
- *Web Page*: A web page is an individual page of information stored on a web server, typically written in HTML. A web page can be accessed by a web browser over the HTTP protocol by specifying a unique handle known as a URL. Modern web pages use CSS to style the page, and JavaScript to add behavior. Links within a page often point to other pages, either on the same web server or on a different web server.
- *Website*: A website refers to a collection of web pages hosted on a web server under the same domain name, for example, `tugraz.at`.
- *Home Page*: The homepage of a website typically serves as the main or starting page of the site and is the initial page visitors encounter when accessing the website.
- *Queries*: The domain-specific part of the URL can also contain a query string introduced by a question mark (?). The query string is interpreted by the web server, which then generates a response page.

The term *web development* covers all aspects of web design, web programming, database management, and deployment. Web development can be broken down into frontend and backend development, as shown in Figure 2.1. Developers who do both frontend and backend work are known as *full-stack* developers.

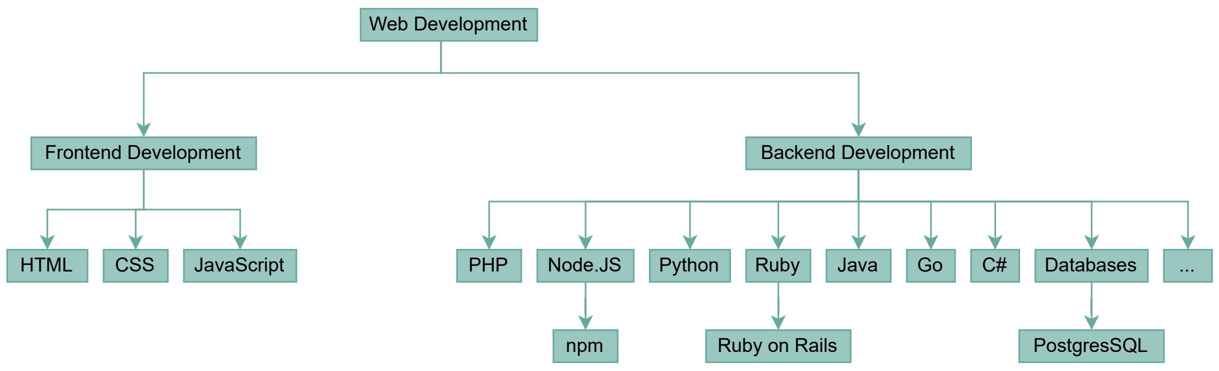


Figure 2.1: The branches of web development. [Diagram created by Duaa Nakshbandi using `app.diagrams.net`.]

2.1 Frontend Technology

An application's frontend is the part visible to the viewer, the graphical user interface of a website which is rendered in the web browser. Through the frontend, data and processes of the underlying backend are simplified and abstracted to present to the user. In frontend development, four main languages are used: HyperText Markup Language (HTML), Scalable Vector Graphics (SVG), Cascading Style Sheets (CSS), and JavaScript (JS).

2.1.1 HyperText Markup Language (HTML)

The HyperText Markup Language (HTML) is a markup language used to structure and display text, images, and other media in web pages. It is also useful for linking to other web pages. The HTML code of a web page consists of markup elements, each with an opening tag, inner content, and closing tag [WHATWG 2022, Section 3.2]. The fundamental structure of a web page can be seen in Listing 2.1. Web browsers, such as Chrome, Firefox, or Safari, interpret and graphically display HTML files. The Web Hypertext Application Technology Working Group (WHATWG) is responsible for developing and standardizing HTML and maintaining a Living Standard specification which is constantly evolving [WHATWG 2023].

An HTML document has two main parts:

- `<header>`: Primarily containing technical or documentary information, which is typically not displayed in the browser's display area.
- `<body>`: Containing the information that is typically visible in the browser's display area.

2.1.2 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) images are vector graphics stored as objects and paths instead of pixels. They are highly flexible, allowing for animation, and manipulation via HTML/CSS, and retain sharp edges even when enlarged. SVG files typically have the file extension `.svg`. Modern web browsers also support the use of SVG elements inside web pages, just like using HTML elements. Currently, SVG 1.1 [W3C 2011] and some parts of SVG 2.0 [W3C 2018] can be used in web pages. JavaScript can be used to create and manipulate SVG elements inside the browser's DOM (Document Object Model). When used appropriately, SVG graphics are of moderate size and can be easily resized without losing quality. They also support interactivity and can be manipulated with filters. SVG is developed and maintained by the W3C (World Wide Web Consortium), the main international standards body for the Internet.

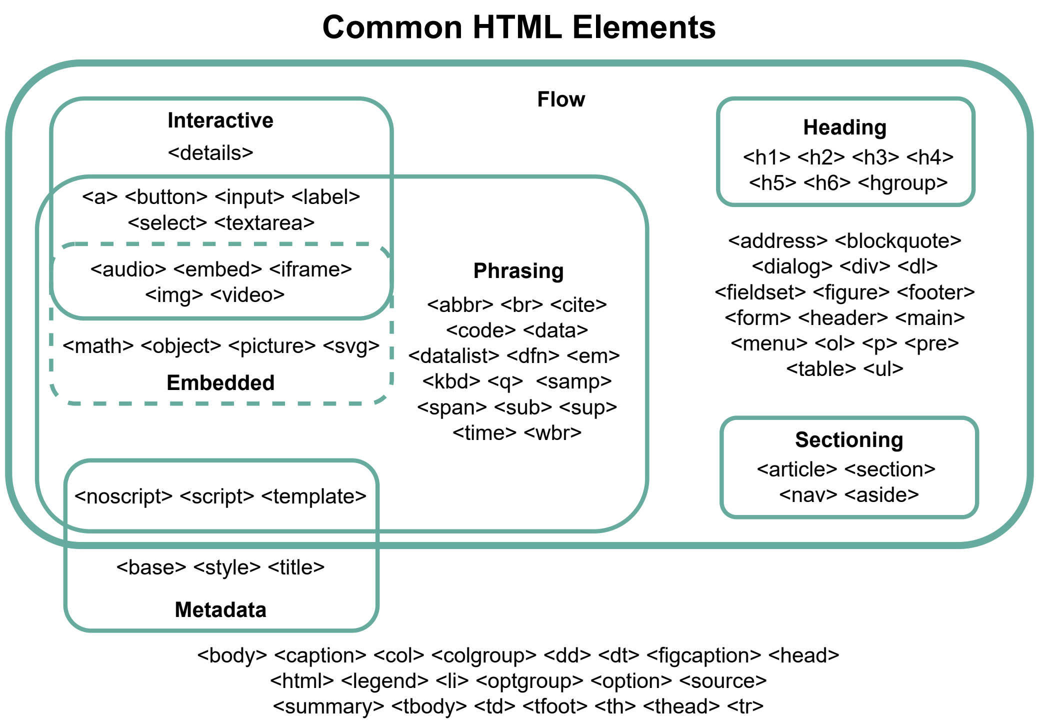
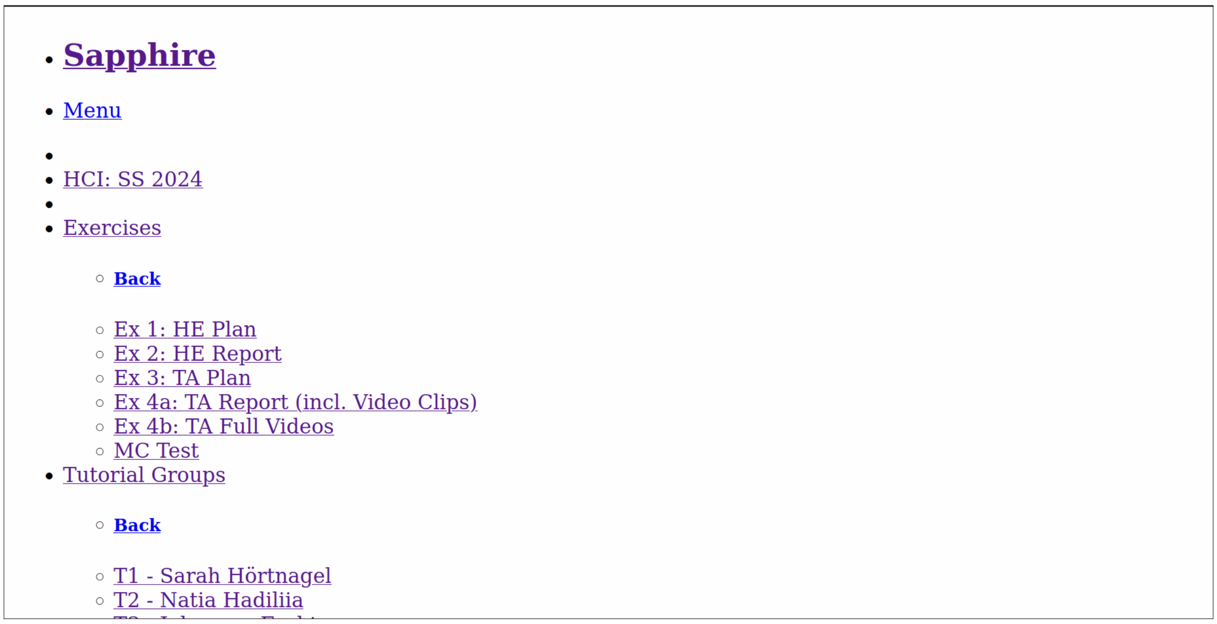


Figure 2.2: An overview of the most common HTML5 elements. [Redrawn from the figure 'HTML element content categories' of Wikipedia [2023a].]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title of website</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <!-- additional header details -->
    <!-- The browser does not display comments -->
  </head>
  <body>
    <h1>A Heading</h1>
    <p>A paragraph of text</p>
  </body>
</html>
```

Listing 2.1: The basic structure of an HTML web page.



- Sapphire
- Menu
-
- HCI: SS 2024
-
- Exercises
  - Back
  - Ex 1: HE Plan
  - Ex 2: HE Report
  - Ex 3: TA Plan
  - Ex 4a: TA Report (incl. Video Clips)
  - Ex 4b: TA Full Videos
  - MC Test
- Tutorial Groups
  - Back
  - T1 - Sarah Hörtnagel
  - T2 - Natia Hadiliia

Figure 2.3: Sapphire’s main page without CSS styling. The page is displayed using the web browser’s default styling for HTML elements. [Screenshot taken by the author.]

2.1.3 Cascading Style Sheets (CSS)

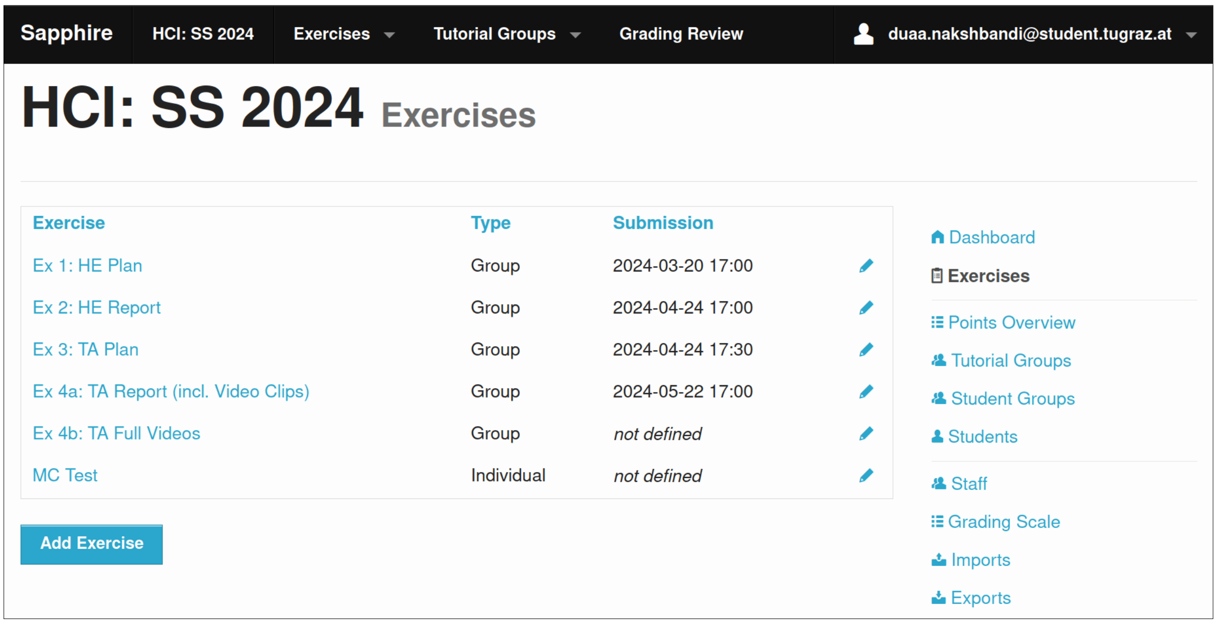
Cascading Style Sheets (CSS) is the current standard for styling and laying out web pages [W3C 2023]. CSS controls how an HTML document is presented, by adjusting the positioning, sizing, and styling of HTML (and to some extent SVG) elements. It can be used to define fonts, colors, lines, heights, and widths, among other things. The original goal of CSS was to separate content and appearance, simplifying both the programming code of a website and the work of the web designer. CSS was first defined in 1997 and has since been continuously enhanced. It is defined and standardized by the World Wide Web Consortium (W3C) [W3C 2023]. The effect of CSS is demonstrated in Figures 2.3 and 2.4, which show a Sapphire website with and without CSS styling.

Once a design has been created using CSS, it can be quickly and easily transferred to another project. CSS Modules provide a way to locally scope CSS classes and animation names. CSS frameworks are pre-built CSS stylesheets with defined color settings, layouts, and other elements. They recently have become increasingly popular among web developers, due to their ability to simplify and speed up the development process. Popular CSS frameworks include UIkit [UIkit 2023], Bootstrap [Bootstrap 2023], Bulma [Bulma 2023], Tailwind [Tailwind 2024].

2.1.4 JavaScript (JS)

JavaScript (JS) is a scripting language widely used in web development to create interactive web pages, along with HTML and CSS. Web developers use JavaScript for various applications, including the creation of automatically updated news feeds, forms, search functions, and other interactive features. Almost all interactive features found on web pages are created using JavaScript. The JavaScript language is maintained by Ecma International [Ecma 2024].

In real-world applications, there are often numerous validations and dynamic checks that can make JavaScript code difficult to test, particularly due to the lack of type checking. This is where TypeScript [Microsoft 2023b] can be beneficial, since it is a statically typed superset of JavaScript, with a compiler that detects and warns about type mismatches. TypeScript can execute all JavaScript code and supports all of its libraries, while also providing additional features and capabilities. When TypeScript code is



HCI: SS 2024 Exercises

Exercise	Type	Submission
Ex 1: HE Plan	Group	2024-03-20 17:00
Ex 2: HE Report	Group	2024-04-24 17:00
Ex 3: TA Plan	Group	2024-04-24 17:30
Ex 4a: TA Report (incl. Video Clips)	Group	2024-05-22 17:00
Ex 4b: TA Full Videos	Group	*not defined*
MC Test	Individual	*not defined*

Figure 2.4: Sapphire's main page styled with CSS. [Screenshot taken by the author.]

compiled, it is transpiled into a specific version of JavaScript, making it compatible with all modern web browsers.

Initially, JavaScript was intended to be used for frontend development, running inside a web browser. With the availability of JavaScript runtime environments like Node.js (discussed in Section 2.2.4), it can also be used outside the browser for a wide variety of applications.

2.2 Backend Technology

Backend development deals with the design and programming of server-side areas of a web application. This includes databases and server-side logic. The backend typically makes use of database servers or file access systems. The backend is close to the server hardware and is considered the counterpart to the frontend. The frontend and backend are closely connected and work together.

2.2.1 Ruby

Ruby is an object-oriented, high-level, interpreted, general-purpose programming language [Ruby 2023a]. It was created by Japanese programmer Yukihiro "Matz" Matsumoto. The official implementation is available under an open-source license. The language employs garbage collection and is dynamically typed. Ruby supports both the procedural and functional programming paradigms. Its primary use case is as a general-purpose scripting language with complete support for various web server applications, standard libraries, servers, and other system utilities. Drawing inspiration from Python and Perl, the syntax of Ruby features object-oriented programming features similar to those of Smalltalk. Ruby is an interpreted scripting language, most instructions are executed directly without the use of a compiler to convert the code into machine language instructions. Ruby programmers can leverage the powerful features of RubyGems, a package manager that provides tools to manage "gems", a standard format for the self-contained distribution of Ruby programs and libraries [Ruby 2023b].

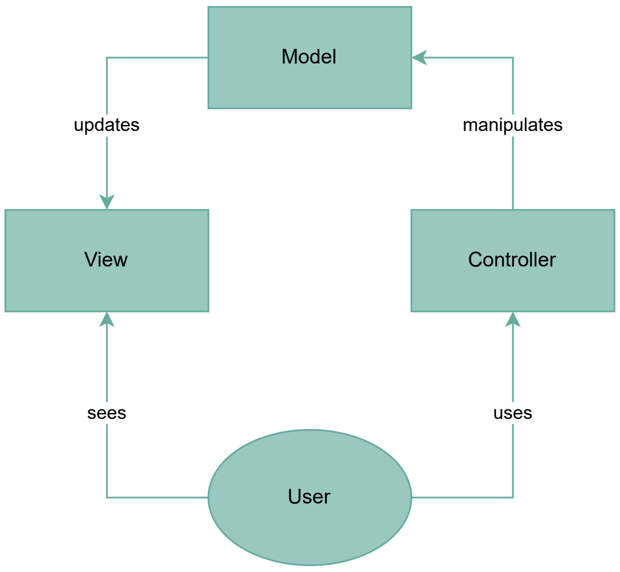


Figure 2.5: The Model-View-Controller (MVC) pattern. [Redrawn from Figure “Diagram of interactions within one possible take on the MVC pattern” from Wikipedia [2023b].]

2.2.2 Ruby on Rails

Ruby on Rails, commonly known as RoR or just Rails [Rails 2023], is a platform-independent framework built on top of the Ruby programming language. Rails is well-suited for developing dynamic websites that access databases, since it provides an efficient way to encapsulate database access and requests, leading to faster development times compared to other programming languages.

According to the discussion in Notodikromo [2020], the Rails runtime environment is a self-contained system that utilizes various scripts, libraries, and the Ruby interpreter. Each Rails application is organized into a directory tree which includes views, configuration files, database models, and controllers, among other components. One of the unique features of Rails is the separation of the database, data preparation, and data display areas, which makes maintaining Rails applications straightforward.

Rails, like many other frameworks, is based on the Model-View-Controller (MVC) pattern, illustrated in Figure 2.5. The MVC pattern is a widely documented concept, with resources available from notable sources such as Microsoft [Microsoft 2022], Oracle [Oracle 2023], and Apple [Apple 2018]. It comprises three components: Model, View, and Controller. The Model represents the logical structure of data in the application and the high-level class associated with it. The View is a set of classes that represent the elements in the user interface. These elements consist of all the visible objects on the screen, including buttons, panels, dialogs, and other interactive objects. The controller is responsible for handling user requests, processing data from models, and returning appropriate responses to views.

A goal of the MVC pattern is a flexible program design, which facilitates later change or extension and makes the reusability of the individual components possible. Following the discussion in Sunyaev [2020], one possible approach is to develop an application that leverages a shared model and adapts it for different platforms, such as Windows, Mac, Linux, or the Internet. In this case, the implementations share the same underlying model, but require new Controller and View implementations for each platform.

In the context of Rails, the MVC pattern is utilized in the following manner:

- *Model*: A database schema describing relationships between entities, distinguishing between classes, properties, and methods. Each class usually corresponds to a database table. Each line in this table is in turn assigned an object, which in turn has properties in the form of fields.

- *View*: Used to visualize data. The so-called “Action View” class can be rendered in HTML, XML, or JavaScript on a whole website or in parts.
- *Controller*: A Ruby file that is executed when a request is made via a URL. If, for example, a page such as the shopping cart is called in an online store, a parameter contained in the URL triggers a database query. This data is then processed via the View and an HTML page is filled with content.

2.2.3 PostgreSQL

As described by Ferrari and Pirozzi [2020], PostgreSQL is a prime example of an Object-Relational Database Management System (ORDBMS) that can store complex data items relationally. One of the key features of Postgres is its high adaptability, extensibility, and modifiability, allowing users to independently add new data types, operators, or functions, as well as extend indexing methods, triggers, and rules for flexible access to data objects. Postgres [PGDG 2023] operates using a client-server model over TCP/IP, where the server manages the databases and client requests. The central component of the server is the Postmaster. Both the server and clients can operate on various systems. Many Linux distributions provide a graphical client in addition to the command-line client. Postgres is free and platform-independent and is licensed under Berkeley Software Distribution (BSD) [UC Berkeley 1999]. It was originally developed at the University of California at Berkeley in the 1980s and was released for free in 1996, with the original name of “Postgres” still used today as a synonym.

2.2.4 Node.js

Node.js is an open-source, cross-platform runtime environment for JavaScript [OJSF 2023b] with a single thread for building quick and extensible backend and networking services. It is efficient and well-suited for real-time applications thanks to its event-driven, non-blocking I/O design and use of the V8 JavaScript runtime engine [OJSF 2023a]. Node.js is not a programming language. It is a runtime environment which allows the execution of JavaScript outside of web browsers. It makes it possible to build backend services for mobile or web apps using JavaScript.

Chapter 3

Learning Management Systems (LMS)

"If you look at history, innovation doesn't come just from giving people incentives; it comes from creating environments where their ideas can connect."

[Steven Johnson]

As outlined by Foreman [2017], a learning management system (LMS) is a multiuser software application that enables academic and commercial organizations to manage and track learning and training events and courses, which are accessed through a web browser. Both commercial and open-source LMS systems are available. While most LMS products are stand-alone systems, some are offered as apps or plug-ins for popular website frameworks, such as WordPress [WP 2023] and SharePoint [Microsoft 2023a].

LMS systems can be broadly categorized as corporate and academic. Academic LMS typically provide a virtual classroom environment, features for managing students' user accounts, managing course content, class administration, and grading. Corporate LMS focuses more on the delivery of training programs, without the need for fine-grained marking and feedback.

In an academic LMS, *students* (or participants, or pupils) upload their submissions for particular *exercises* (or assignments, or tasks). *Lecturers* (or teachers, instructors, or professors) provide course materials and prepare the exercises. *Tutors* (or teaching assistants) are often employed to grade submissions.

3.1 Features of Academic LMS

Academic LMS provides several features for lecturers and students to engage in effective online learning, as discussed by Foreman [2017, Chapter 3], including:

- *Course Content Management*: Course content management includes creating, managing, and storing lessons and exercises, course syllabus and schedules, quizzes and polls, and multimedia course materials created by the lecturer.
- *User Management*: User management includes creating and managing user accounts, user profiles, and permissions. Distinct user profiles help establish a virtual classroom and keep track of students' grades and performance.
- *Uploading Submissions*: Academic LMS often provide an online portal which supports file uploads in a variety of file formats, and may include drag-and-drop functionality. The submitted work can subsequently be easily located and reviewed by the evaluator in a single repository.

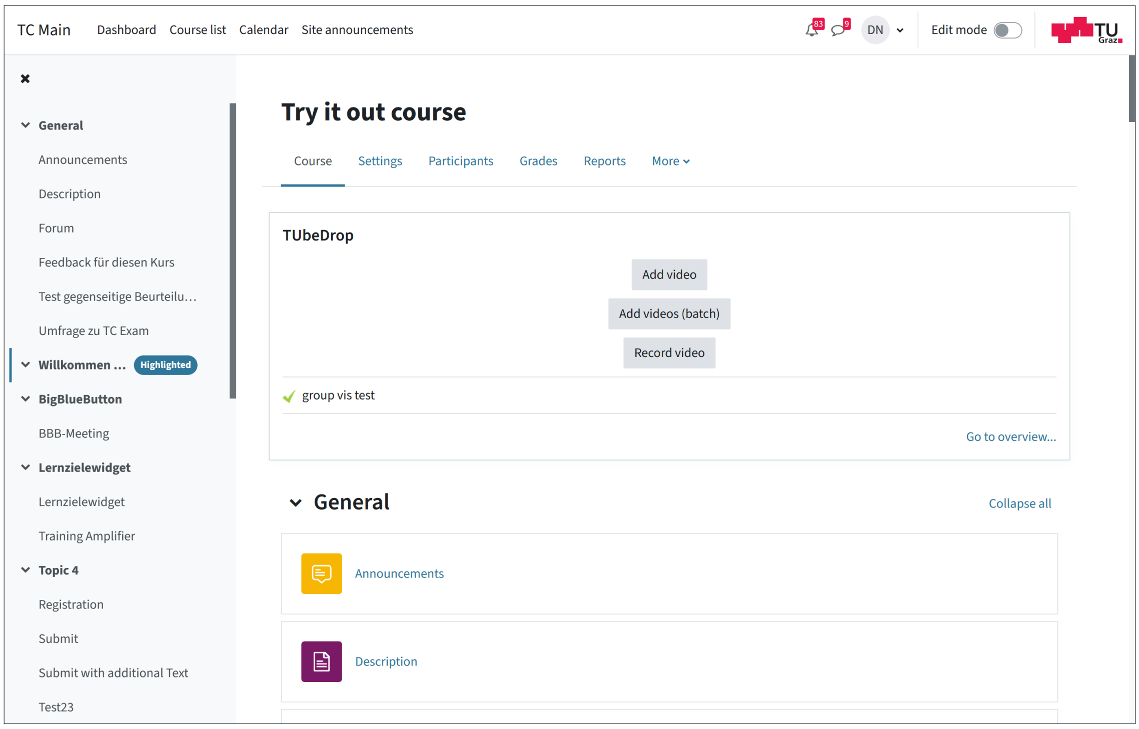


Figure 3.1: From a lecturer's perspective, managing course content within a demo course on the TU Graz TeachCenter Moodle instance. [Screenshot taken by the author.]

- *Defining a Grading System*: A system of ratings and points defining the criteria for evaluating submissions is essential when multiple tutors are employed in grading, to ensure consistency and transparency in the grading process.
- *Grading Interface*: Most LMSs provide an interface for a lecturer or tutor to enter an achieved number of points for each submission, as well as textual feedback on a per-exercise basis.

3.2 Moodle: Open-Source LMS

One prominent example of a learning management system is Moodle [Moodle 2023]. Moodle is an open-source learning management system created by Martin Dougiamas in 2002. It has gained significant traction in recent years and currently hosts 200 million educational resources on its sites. Moodle has been adopted by various institutions, including schools, colleges, and businesses. It enables different e-learning teaching methodologies, such as blended learning, distance learning, and flipped classrooms. Lecturers can use Moodle to create and organize online courses and deliver course materials and content (see Figure 3.1). Moodle provides a range of features, including quizzes, exercises (see Figure 3.2), grading, and feedback. The platform also offers various collaboration and communication tools between students and lecturers, such as forums (see Figure 3.3), messaging, and group work (see Figure 3.4).

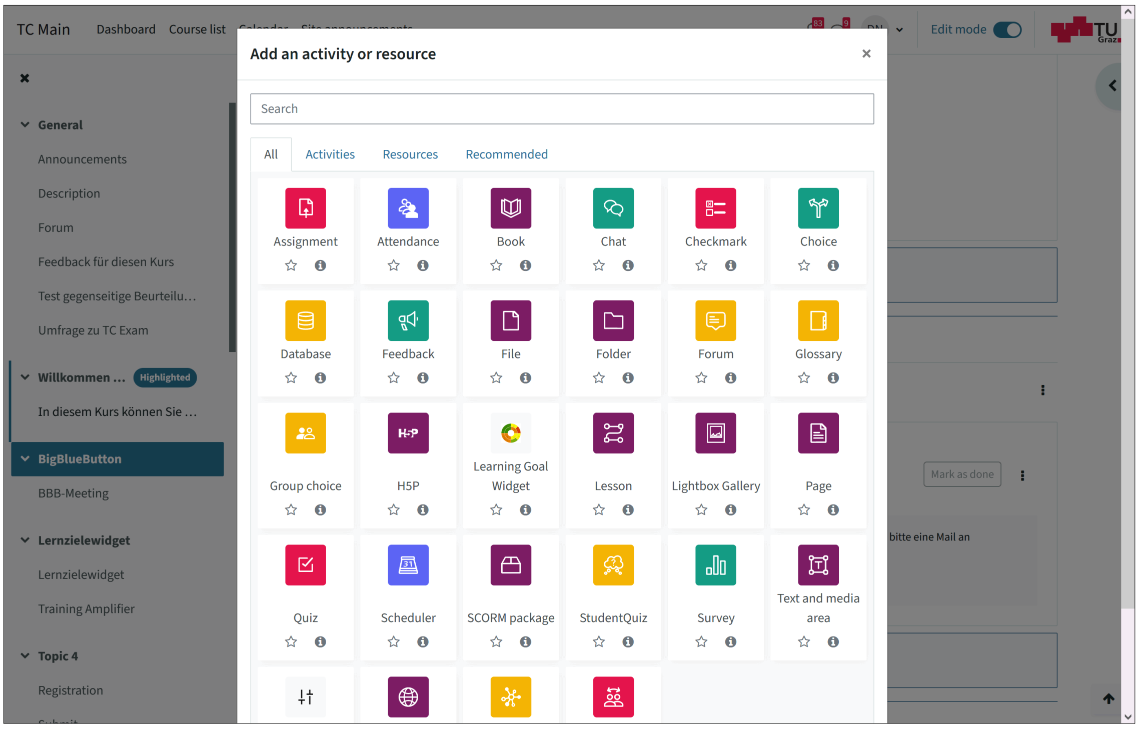
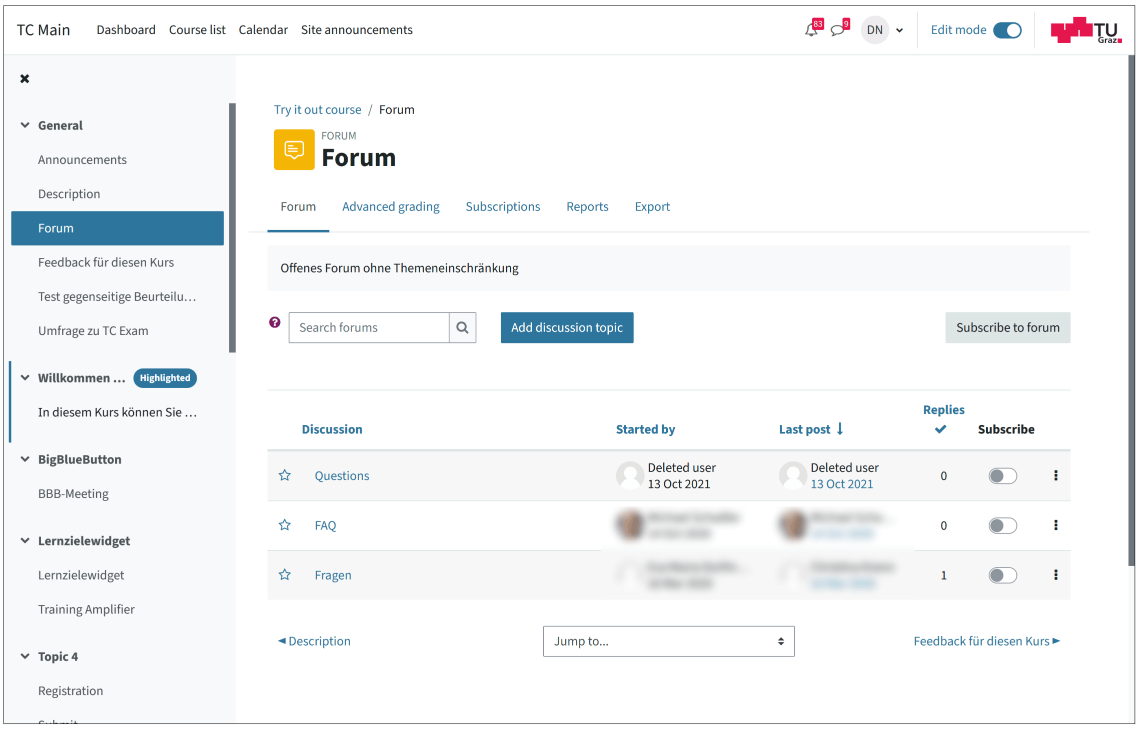


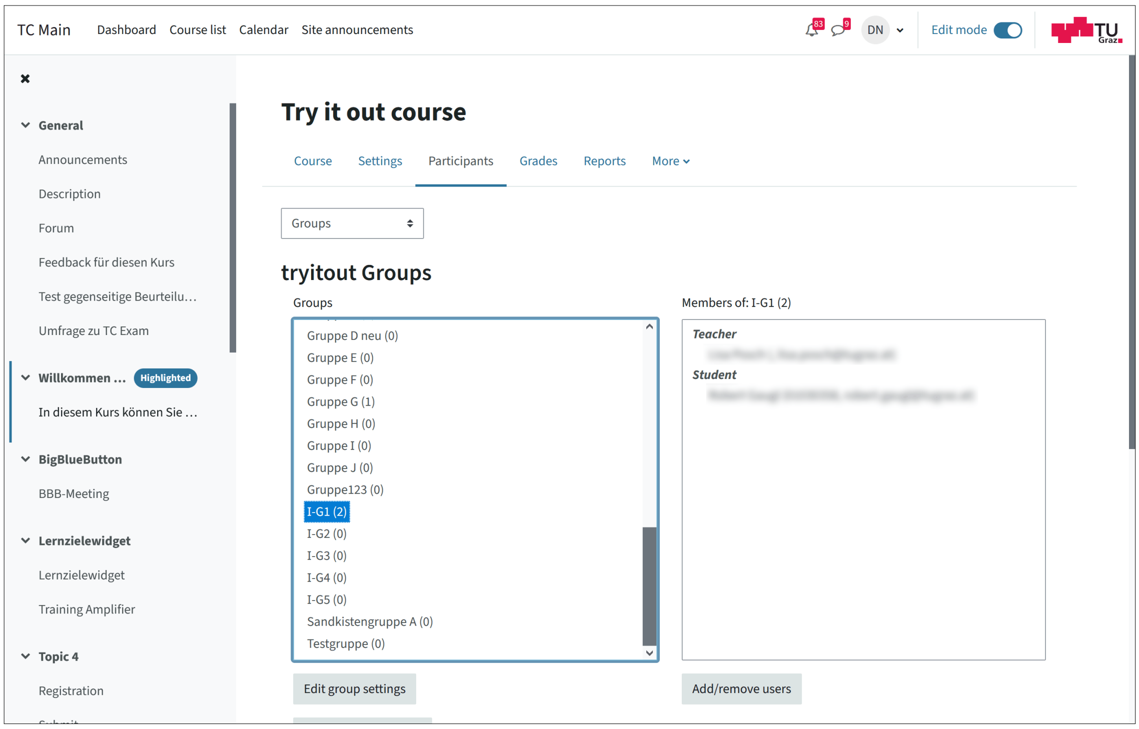
Figure 3.2: From a lecturer's perspective, managing course exercises within a demo course on the TU Graz TeachCenter Moodle instance. [Screenshot taken by the author.]



Offenes Forum ohne Themeneinschränkung

Discussion	Started by	Last post	Replies	Subscribe
Questions	Deleted user 13 Oct 2021	Deleted user 13 Oct 2021	0	
FAQ	[redacted]	[redacted]	0	
Fragen	[redacted]	[redacted]	1	

Figure 3.3: A course forum within a demo course on the TU Graz TeachCenter Moodle instance. [Screenshot taken by the author.]


TC Main Dashboard Course list Calendar Site announcements DN Edit mode

General
Announcements
Description
Forum
Feedback für diesen Kurs
Test gegenseitige Beurteilu…
Umfrage zu TC Exam
Willkommen … Highlighted
In diesem Kurs können Sie …
BigBlueButton
BBB-Meeting
Lernzielewidget
Lernzielewidget
Training Amplifier
Topic 4
Registration

Try it out course

Course Settings Participants Grades Reports More

Groups

tryitout Groups

Groups
Gruppe D neu (0)
Gruppe E (0)
Gruppe F (0)
Gruppe G (1)
Gruppe H (0)
Gruppe I (0)
Gruppe J (0)
Gruppe123 (0)
I-G1 (2)
I-G2 (0)
I-G3 (0)
I-G4 (0)
I-G5 (0)
Sandkistengruppe A (0)
Testgruppe (0)

Members of: I-G1 (2)
Teacher
[redacted]
Student
[redacted]

Edit group settings Add/remove users

Figure 3.4: From a lecturer's perspective, managing course groups within a demo course on the TU Graz TeachCenter Moodle instance. [Screenshot taken by the author.]

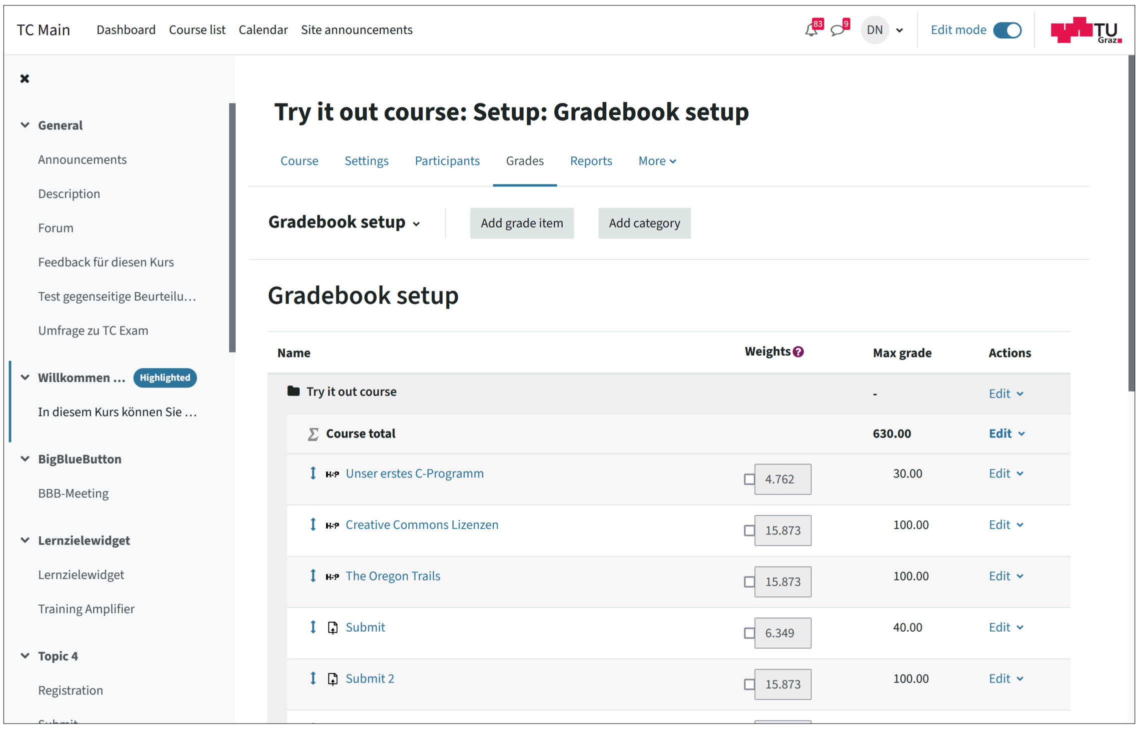
3.3 Grading in Moodle and Sapphire

In Moodle, lecturers configure a Grade book containing a (maximum) number of points per exercise and the total number of points for the course is calculated automatically, as illustrated in Figure 3.5. When grading, tutors look through each submission, determine the number of points to award, and enter the points into Moodle. Moodle does not provide support for a more granular approach to grading, such as defining ratings and deductions within an exercise. Such ratings and deductions on a per-exercise basis would have to be specified and maintained externally, say in a spreadsheet. Only the achieved points for the exercise as a whole are maintained by Moodle. The only way to communicate more granular feedback to a student in Moodle is to write or paste it into a text box, as shown in Figure 3.6.

In contrast, Sapphire offers the option to assign grading metrics (ratings) on a per-exercise basis, as shown in Figure 3.7. Lecturers in Sapphire can specify and enforce built-in grading metrics and points deductions for tutors to use for grading each exercise.

In many courses, a *grading scheme* determines how point totals are converted into grades by setting grade boundaries. Moodle has no explicit support for a grading scheme. This limitation often leads lecturers to have to manually communicate the course grading scheme, say by including it in one of the slides of their first lecture, as shown in Figure 3.8. Some lecturers communicate the grading scheme by using the Description tab in Moodle, as shown in Figure 3.9. The lack of a standard approach often results in students having difficulty locating the grading scheme.

In contrast, Sapphire includes built-in support for a grading scheme. Lecturers specify grade boundaries based on total points and assign them to a 1 (excellent) to 5 (fail) scale, as shown in Figure 3.10. The grade boundaries can be easily modified by dragging and dropping the boundaries in a visual overview, as shown in Figure 3.10. Currently, however, the grading scheme is not communicated to students, although this is logged as an issue for future development.

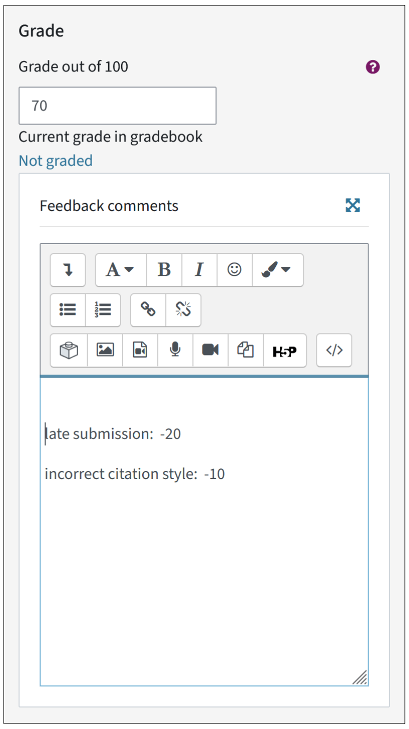


Try it out course: Setup: Gradebook setup

Gradebook setup

Name	Weights	Max grade	Actions
Try it out course		-	Edit
Course total		630.00	Edit
Unser erstes C-Programm	4.762	30.00	Edit
Creative Commons Lizenzen	15.873	100.00	Edit
The Oregon Trails	15.873	100.00	Edit
Submit	6.349	40.00	Edit
Submit 2	15.873	100.00	Edit

Figure 3.5: From a lecturer's perspective, managing the grading scheme in Moodle within a demo course on Graz University of Technology's TeachCenter Moodle instance. [Screenshot taken by the author.]



Grade

Grade out of 100

70

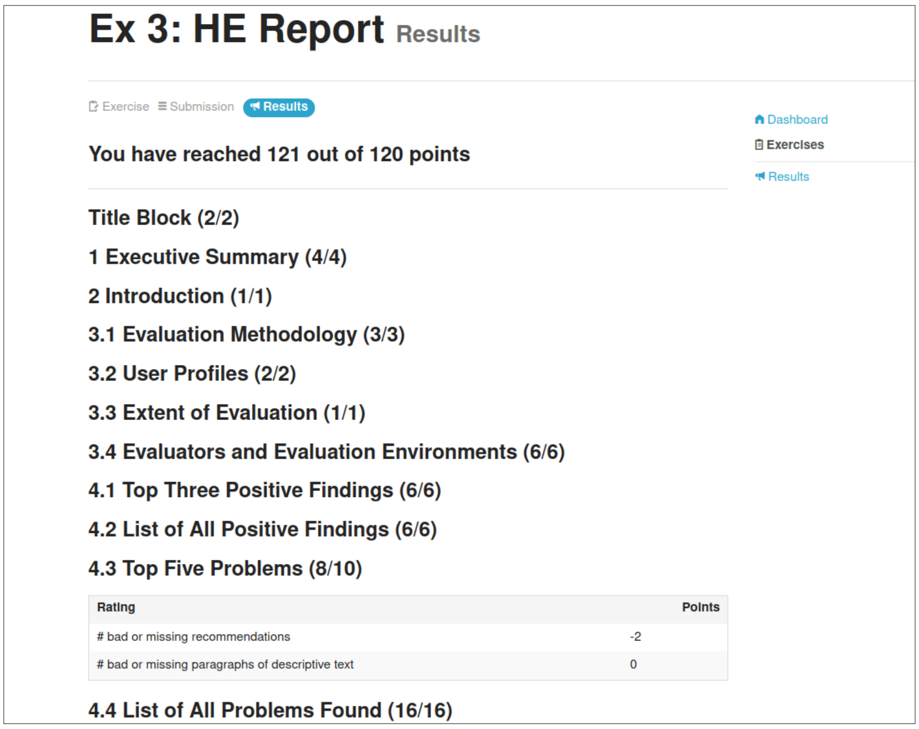
Current grade in gradebook

Not graded

late submission: -20

incorrect citation style: -10

Figure 3.6: Communicating fine-grained details about grading to a student can only be done in Moodle by writing or pasting textual feedback into a comment box. [Screenshot taken by the author.]



Ex 3: HE Report Results

Exercise ≡ Submission Results

You have reached 121 out of 120 points

Title Block (2/2)

1 Executive Summary (4/4)

2 Introduction (1/1)

3.1 Evaluation Methodology (3/3)

3.2 User Profiles (2/2)

3.3 Extent of Evaluation (1/1)

3.4 Evaluators and Evaluation Environments (6/6)

4.1 Top Three Positive Findings (6/6)

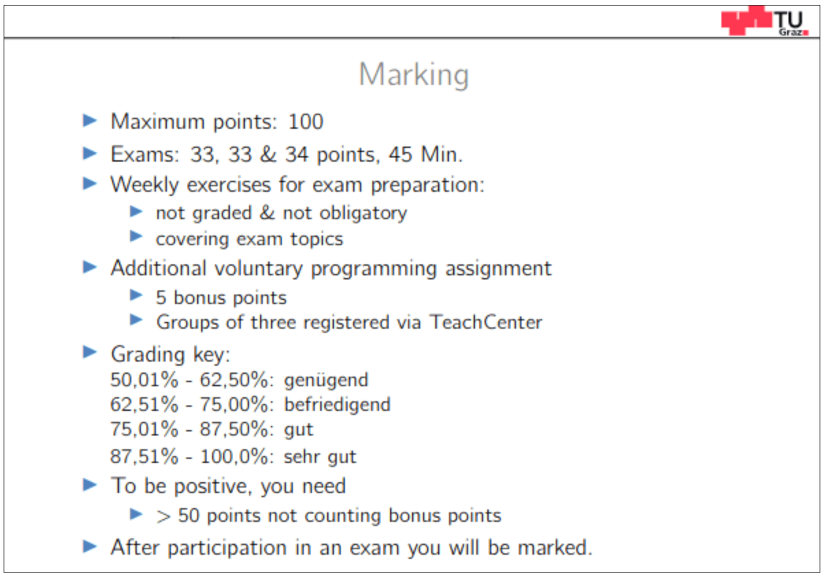
4.2 List of All Positive Findings (6/6)

4.3 Top Five Problems (8/10)

Rating	Points
# bad or missing recommendations	-2
# bad or missing paragraphs of descriptive text	0

4.4 List of All Problems Found (16/16)

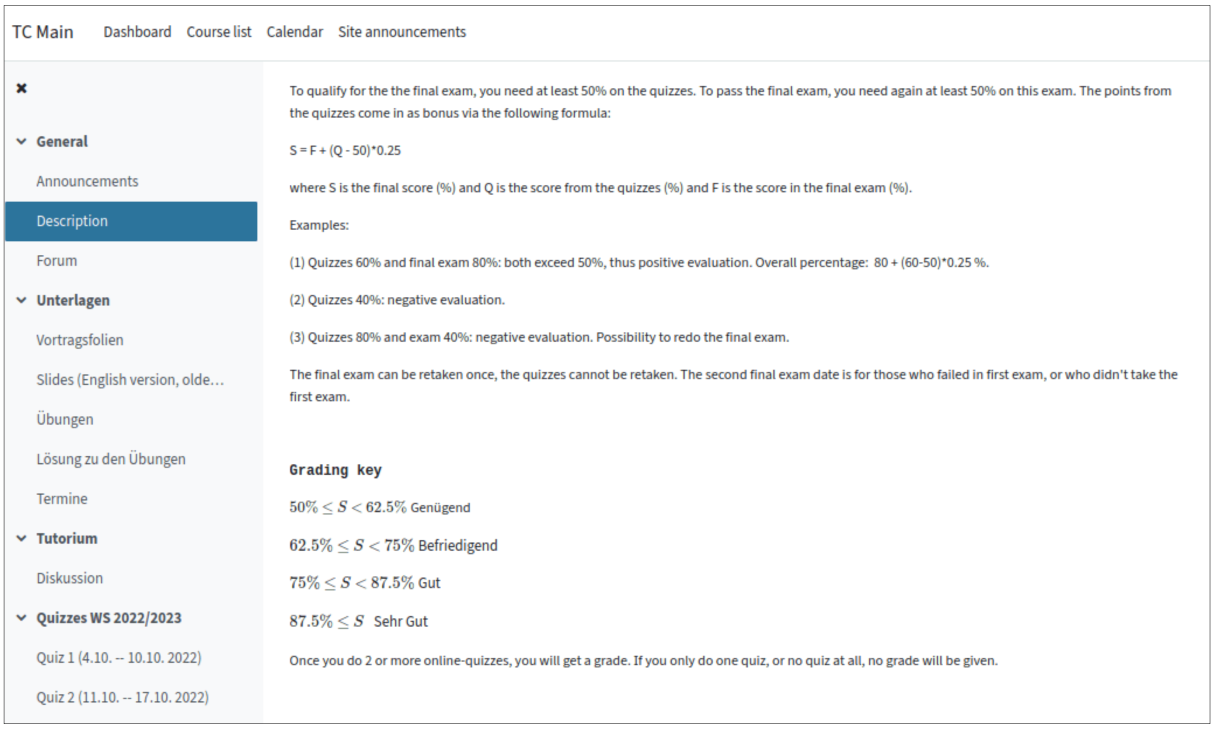
Figure 3.7: In Sapphire, students receive itemized grading details regarding specific points awarded or deducted as well as the total points achieved per exercise. [Screenshot taken by the author.]



Marking

- Maximum points: 100
- Exams: 33, 33 & 34 points, 45 Min.
- Weekly exercises for exam preparation:
  - not graded & not obligatory
  - covering exam topics
- Additional voluntary programming assignment
  - 5 bonus points
  - Groups of three registered via TeachCenter
- Grading key:
  50,01% - 62,50%: genügend
  62,51% - 75,00%: befriedigend
  75,01% - 87,50%: gut
  87,51% - 100,0%: sehr gut
- To be positive, you need
  - $>$ 50 points not counting bonus points
- After participation in an exam you will be marked.

Figure 3.8: Lecturers using Moodle might choose to communicate a course's grading scheme by describing it in a slide in the first lecture. [Screenshot taken by the author.]



To qualify for the the final exam, you need at least 50% on the quizzes. To pass the final exam, you need again at least 50% on this exam. The points from the quizzes come in as bonus via the following formula:

S = F + (Q - 50)*0.25

where S is the final score (%) and Q is the score from the quizzes (%) and F is the score in the final exam (%).

Examples:

(1) Quizzes 60% and final exam 80%: both exceed 50%, thus positive evaluation. Overall percentage: 80 + (60-50)*0.25 %.

(2) Quizzes 40%: negative evaluation.

(3) Quizzes 80% and exam 40%: negative evaluation. Possibility to redo the final exam.

The final exam can be retaken once, the quizzes cannot be retaken. The second final exam date is for those who failed in first exam, or who didn't take the first exam.

Grading key

$50\% \leq S < 62.5\%$ Genügend

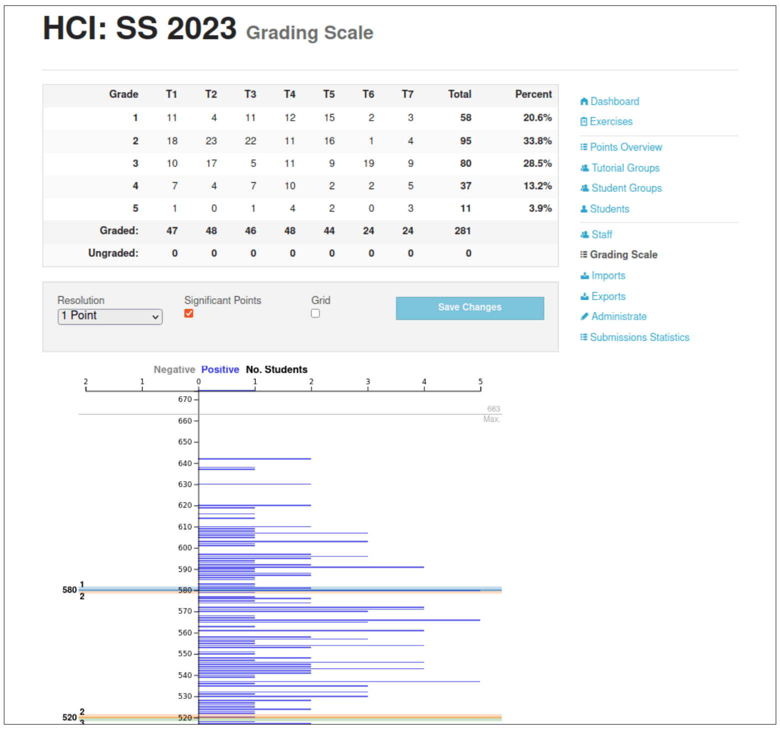
$62.5\% \leq S < 75\%$ Befriedigend

$75\% \leq S < 87.5\%$ Gut

$87.5\% \leq S$ Sehr Gut

Once you do 2 or more online-quizzes, you will get a grade. If you only do one quiz, or no quiz at all, no grade will be given.

Figure 3.9: Alternatively, lecturers using Moodle might describe the grading scheme within Moodle's Description tab. [Screenshot taken by the author.]



Grade	T1	T2	T3	T4	T5	T6	T7	Total	Percent
1	11	4	11	12	15	2	3	58	20.6%
2	18	23	22	11	16	1	4	95	33.8%
3	10	17	5	11	9	19	9	80	28.5%
4	7	4	7	10	2	2	5	37	13.2%
5	1	0	1	4	2	0	3	11	3.9%
Graded:	47	48	46	48	44	24	24	281	
Ungraded:	0	0	0	0	0	0	0	0	

Resolution: 1 Point
Significant Points: ✓
Grid: ☐

Figure 3.10: In Sapphire, lecturers can manage the grading scheme visually, by dragging the grade boundaries up or down. The current grade distribution is shown in the table above the visual interface. [Screenshot taken by the author.]

Chapter 4

Sapphire

"Technology can become the "wings" that will allow the educational world to fly farther and faster than ever before – if we will allow it."

[Jenny Arledge, Director of College and Career Readiness, 2023.]

Sapphire is an open-source web application for managing the submission and grading of academic coursework. Students can upload their submissions for particular exercises (assignments) to a simulated file system using drag and drop. Lecturers can prepare rating scales for exercises, with various kinds of point allocations and deductions. Tutors can then grade a submission by selecting the ratings that apply. Finally, students receive granular feedback about the grading of their submissions.

4.1 History of Sapphire

In the early 2000s, Prof. Keith Andrews at Graz University of Technology taught two large courses as part of the Computer Science Bachelor's degree program: Internet and New Media (INM) and Human-Computer Interaction (HCI). Several hundred students participated in each course, and grading was done by 6 to 10 tutors using Excel spreadsheets to apply various points and point deductions according to a predefined set of ratings.

Working with spreadsheets and so many students proved cumbersome. Sapphire was built to support the submission and grading of student work and to provide granular feedback to each student online as the course progresses. Two students collaborated to build the first version of Sapphire: Thomas Kriechbaumer [Kriechbaumer 2014] worked on the backend and Matthias Link on the frontend [Link 2021]. Matthias still acts as a mentor to the project and approves pull requests to the repository. Over the years, other students contributed to the codebase, including Stefan Pranger and Amel Smajic.

4.2 Software Architecture

Sapphire is an open-source project hosted on GitHub [Link and Kriechbaumer 2024a]. It is written in Ruby with Rails [Rails 2023], together with HTML, CSS, and JavaScript, and uses a Postgres database [PGDG 2023]. The software architecture of Sapphire is shown in Figure 4.1. Sapphire is a typical client-server web application, with a frontend running in the web browser and a backend running on a web (application) server. The web frontend communicates with the application server by making page requests and API calls.

The web frontend is built using HTML5 for markup, Cascading Style Sheets (CSS) for styling, and JavaScript (JS) for dynamic behavior. The JavaScript framework Turbolinks [Turbolinks 2021] is

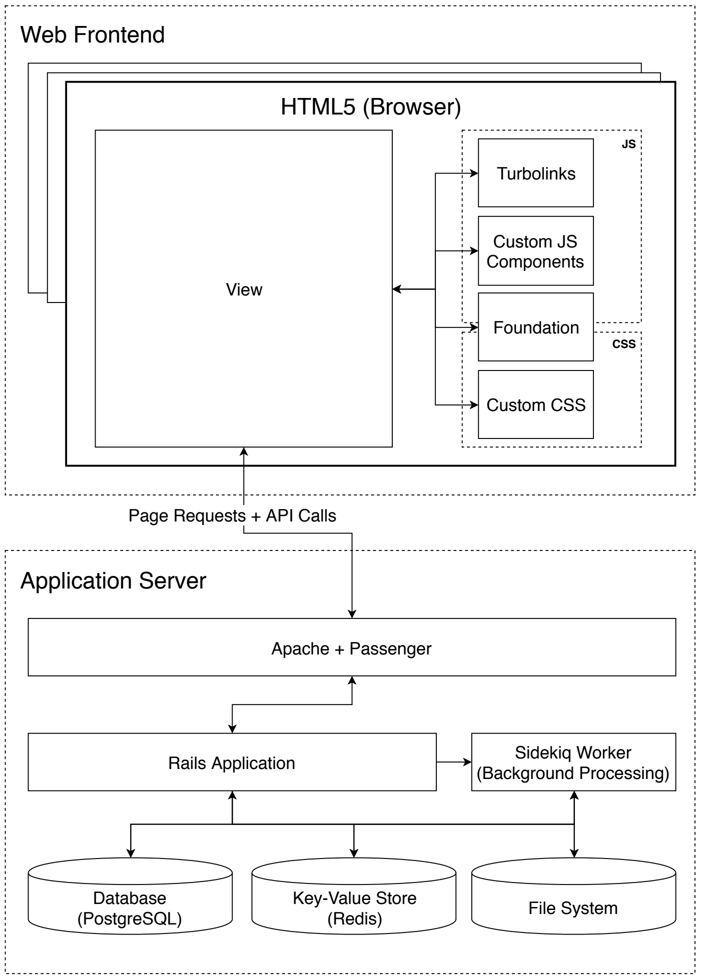


Figure 4.1: The software architecture of Sapphire. [Diagram extracted from Link [2021, page 24] and used under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) license.]

employed to speed up page loads by replacing full page reloads with AJAX requests [Antoniou 2023]. The Foundation [ZURB 2023] framework is used to provide a responsive and customizable design for the frontend.

The backend logic and application layer are implemented using the Ruby on Rails framework [Rails 2023]. A web server, originally Apache [ASF 2024] but now nginx [Nginx 2024], is coupled with the Passenger [Phusion 2024] application server to host the Ruby on Rails application. Sapphire uses the PostgreSQL [PGDG 2023] relational database management system to store and manage data. The Redis [Redis 2024] key-value store is employed for caching. The server's file system is used to store and manage files and assets. SideKiq [Perham 2024] is utilized for background job processing, allowing time-consuming tasks to be performed asynchronously in the background, separate from the main request-response cycle.

4.3 Data Model

The data model of Sapphire is shown in Figure 4.2. The model defines entities such as Courses, which can have multiple Terms for different academic semesters. Each Term consists of mandatory elements, including a name, an associated course, an optional description, and several associated entities. Other notable entities include Tutorial Groups, designed to distribute the workload among tutors for large courses; Student Groups, reflecting student collaborations; Exercises, defining units of work with a deadline and achievable

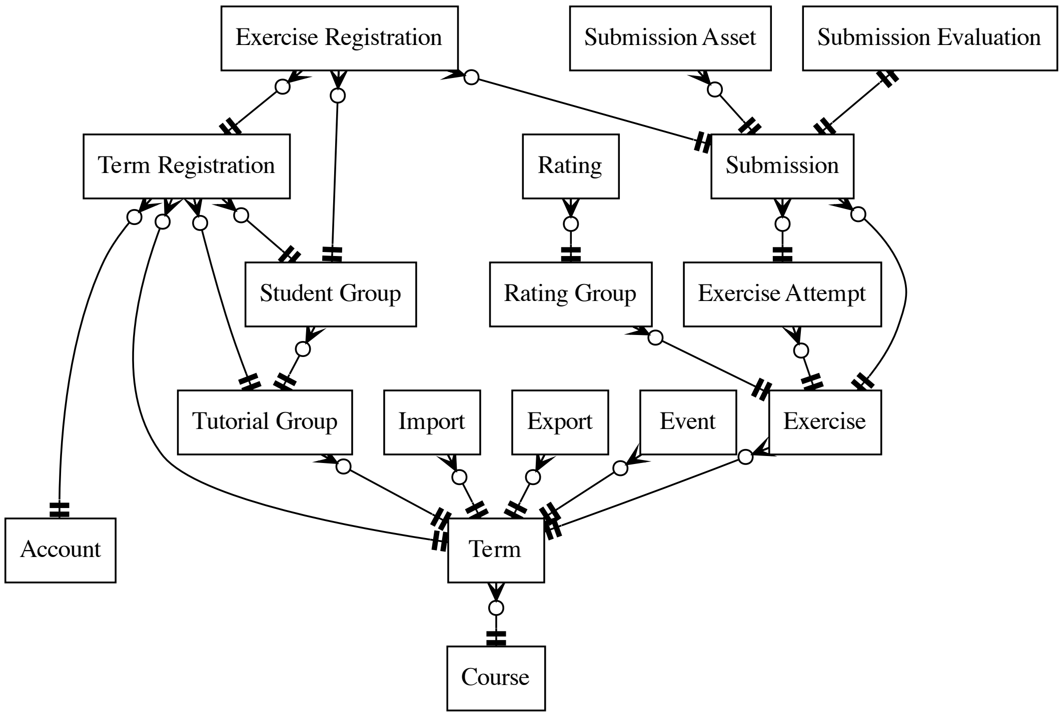


Figure 4.2: The data model of Sapphire shows entities and the relationships between them. [Diagram extracted from Link [2021, page 21] and used under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) license.]

points; and Submissions, instances of student or group work graded based on a rating system. Submission Assets represent submission files, Submission Evaluations manage grading aspects, and Exercise Attempts allow multiple submissions for an Exercise. The grading system involves Rating Groups, each with a starting number of points and configured point ranges. The Term Registration entity links user Accounts to Terms, distinguishing roles as lecturer, tutor, or student. The Exercise Registration entity associates submissions with students, and the Import and Export entities allow for data transfer with external applications. Sapphire's multi-user system automatically tracks database changes and enforces access restrictions based on roles.

4.4 Users and Roles

The Sapphire system implements four distinct roles for its users: Student, Lecturer, Tutor, and Admin. Tutors and Lecturers are considered to be staff members. An Admin is a special role with complete access to the system. Each role entails distinct permissions and rights, which will be expanded upon in the subsequent discussion.

4.4.1 Student

On Sapphire, Students possess the capability to create assignment Submissions, upload files to the submission system, establish directories within the submission system, and eventually access feedback from their tutors or lecturer, along with viewing their final grades. Students cannot independently create accounts on Sapphire; rather, the creation of student accounts occurs at the beginning of each semester, as a task undertaken by the lecturer or an admin tutor. Upon creation of the student accounts, each student is notified via email of their Sapphire login credentials. Students do not hold administrative or grading permissions. They can only review their own submissions and are unable to access and view submissions submitted by other students in the course.

4.4.2 Lecturer

In Sapphire, a Lecturer is responsible for creating a Course instance, creating student Accounts, managing Student Groups, creating and managing Exercises, establishing due dates, and overseeing ratings. Lecturers on Sapphire also possess grading permissions and can provide feedback to students, although these tasks are often undertaken by tutors. Lecturers have access to an overview of all registered students, including their submissions and assigned points. On Sapphire, lecturers have access to analytics that facilitate an understanding of students' academic performance, and they can set and manage grading schemes (points to grade mapping).

4.4.3 Tutor

Tutors are typically students who have completed the course and are now employed by the lecturer to assist with grading and coursework. In Sapphire, Tutors are considered to be staff. Notably, Tutors use the same Sapphire account initially created for them as Students, and their status is subsequently upgraded to that of Tutor.

Each tutor is assigned a Tutorial Group. Although each tutor is typically only responsible for grading the Submissions of their own Tutorial Group, tutors have access to view and grade Submissions of other tutors' Tutorial Groups as well. It is the responsibility of the current Sapphire administrator to create these groups at the beginning of each semester. This process is described in Section 6.1.1.

The rights granted to Tutors on Sapphire include the ability to view and grade all student Submissions for the specific semester. As a result, Tutors have access to specific software features that are not visible to Students. The Tutor role on Sapphire has fewer privileges than Admins or Lecturers, who have full access. Typically, one tutor might be given Admin rights to also serve as the admin tutor.

4.4.4 Admin

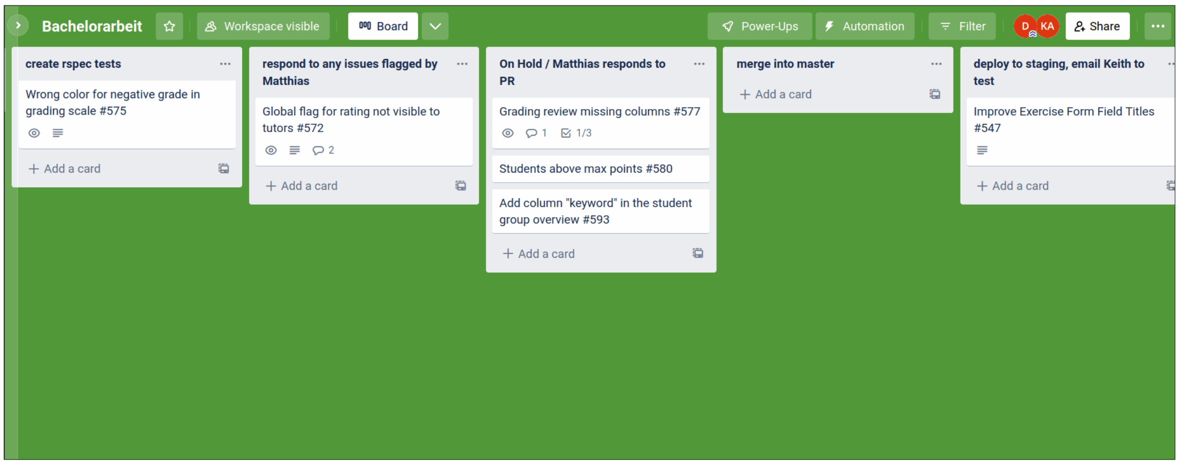
In the context of Sapphire, an Admin role deviates from conventional roles like Lecturer, Tutor, or Student by having specific additional permissions. Notably, there is no direct method to identify accounts with admin rights directly in the Sapphire web interface; instead, these rights are assigned through the Rails console and are noted in the database. A Sapphire Admin, therefore, is essentially a user provided with admin privileges. The procedure for granting such privileges is described in Chapter 6.

Typically assigned to one or two Sapphire developers proficient in Ruby on Rails, the responsibilities of Sapphire admins encompass contributing to the development of new features, ensuring smooth software operation, and addressing any arising issues. The additional permissions include the ability to confer admin permissions to other accounts, assign roles to students and tutors, and manage user accounts. Furthermore, users with admin privileges have access to the records of past classes, all Sapphire accounts, and grading information. In instances of a transition between Sapphire admins, a departing admin typically provides training and support to the succeeding admin.

4.5 Current Development

During the time of writing the thesis, the current contributors to Sapphire are Amel Smajic and Duaa Nakshbandi, with Bakir Haljevac later joining the team in place of Amel. Matthias Link reviews and approves pull requests. Keith Andrews raises issues and directs the project as a whole.

Email is used to discuss further steps, give updates on the current status of work, and arrange upcoming meetings. GitHub issues and pull requests are used to coordinate software development. Throughout the work on this thesis, the author used a Trello Kanban board [Atlassian 2023], shown in Figure 4.3, for project planning.



create rspec tests
- Wrong color for negative grade in grading scale #575

respond to any issues flagged by Matthias
- Global flag for rating not visible to tutors #572

On Hold / Matthias responds to PR
- Grading review missing columns #577
- Students above max points #580
- Add column "keyword" in the student group overview #593

merge into master

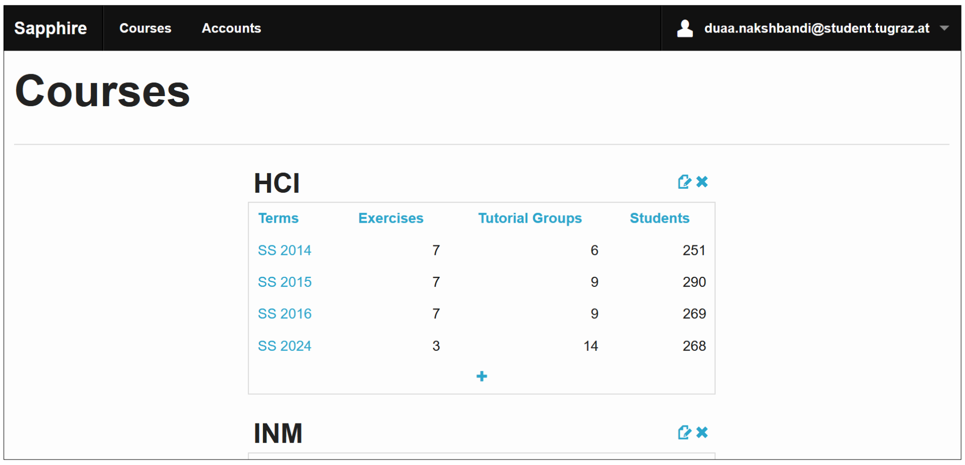
deploy to staging, email Keith to test
- Improve Exercise Form Field Titles #547

Figure 4.3: The Trello Kanban board is used for issue tracking. [Screenshot taken by the author.]

4.6 Sapphire's User Interface

This section defines the terminology used in Sapphire to refer to various user interface (UI) components. The terminology is based partly on element names in the codebase and partly on commonly accepted terms used to describe web objects. The illustrations are based on the admin view of the staging Sapphire server (see Subsection 6.2.5). The Sapphire UI has the following main components:

- View: The entire interface visible to the user of a specific web page, including all other UI components. Two examples of Views can be seen in Figures 4.4 and 4.5.
- Navigation Bar: The horizontal menu bar at the top of every view. An example is shown in Figure 4.6.
- Dropdown Menu: A dropdown menu opens to reveal of finite list of selectable items when clicked. An example is shown in Figure 4.7.
- Tab: A tab is a UI element which allows switching between a group of overlapping content panels. Some examples are shown in Figure 4.8.
- Link Side Bar: The vertical link bar down the righthand side of the view, allowing the user to navigate between different views. An example is shown in Figure 4.9.
- Link: A link is a clickable text element, which directs the user to another view. Some examples are shown in Figure 4.10.
- Panel: A defined area within a view which groups multiple related UI elements, often under a sub-heading.
- Heading: A prominent text element that informs the user of the primary content or purpose of the view. An example is shown in Figure 4.11. A sub-heading is a smaller instance of a heading.
- Icon: An icon is a graphical symbol, typically used to identify a button or menu item.
- Button: A clickable element containing text and/or an icon which triggers an action when pressed. Some buttons have the classic look of a button, like the one in Figure 4.12. Other buttons are simply clickable icons, like those in Figure 4.13.
- Form: A panel with user input elements like Text Fields and Dropdown Menus to collect user data. Form panels include a Submit, Save, or other specific action button to submit inputted information for processing. An example of a Form is shown in Figure 4.14.



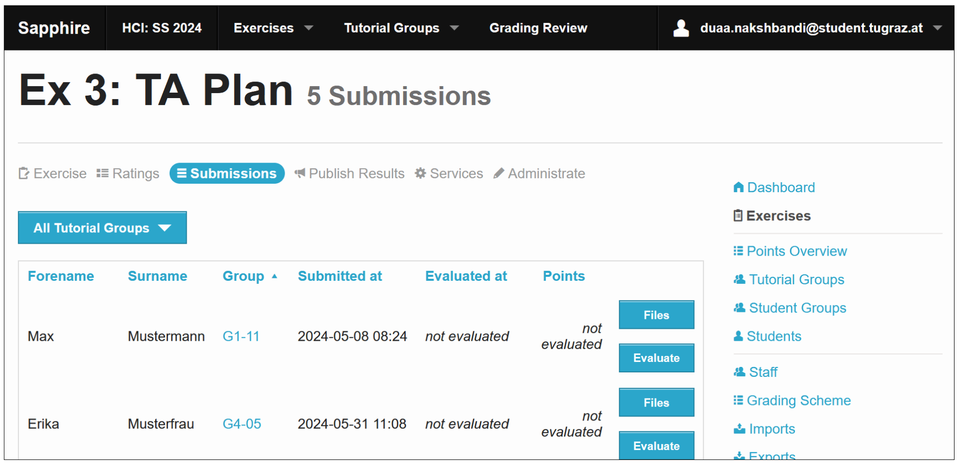
Courses

HCI

Terms	Exercises	Tutorial Groups	Students
SS 2014	7	6	251
SS 2015	7	9	290
SS 2016	7	9	269
SS 2024	3	14	268

INM

Figure 4.4: The Sapphire Courses View, showing a sortable data table. [Screenshot taken by the author.]

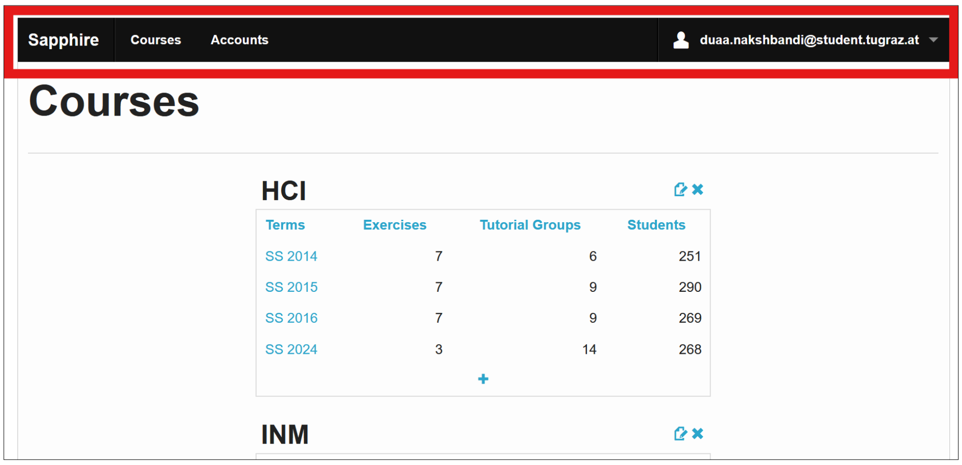


Ex 3: TA Plan 5 Submissions

Forename	Surname	Group	Submitted at	Evaluated at	Points	
Max	Mustermann	G1-11	2024-05-08 08:24	*not evaluated*	*not evaluated*	Files Evaluate
Erika	Musterfrau	G4-05	2024-05-31 11:08	*not evaluated*	*not evaluated*	Files Evaluate

Figure 4.5: The Sapphire Submissions View, showing various UI elements such as Tabs, Buttons, Links, and a Dropdown Menu. [Screenshot taken by the author.]

- Text Field: An input field to capture textual data from the user. Some examples are shown in Figure 4.14.
- Tooltip: A small, informative text box that appears when a user hovers over an element.
- Label: A non-interactive text element intended to provide information or instruction to the user.


Sapphire Courses Accounts duaa.nakshbandi@student.tugraz.at

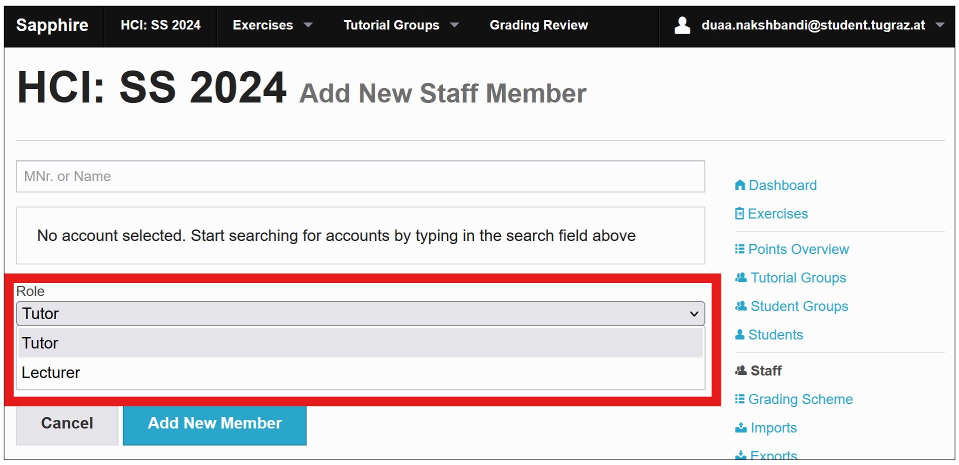
Courses

HCI

Terms	Exercises	Tutorial Groups	Students
SS 2014	7	6	251
SS 2015	7	9	290
SS 2016	7	9	269
SS 2024	3	14	268

INM

Figure 4.6: The Sapphire Navigation Bar, which is at the top of every view. [Screenshot taken by the author.]


Sapphire HCI: SS 2024 Exercises Tutorial Groups Grading Review duaa.nakshbandi@student.tugraz.at

HCI: SS 2024 Add New Staff Member

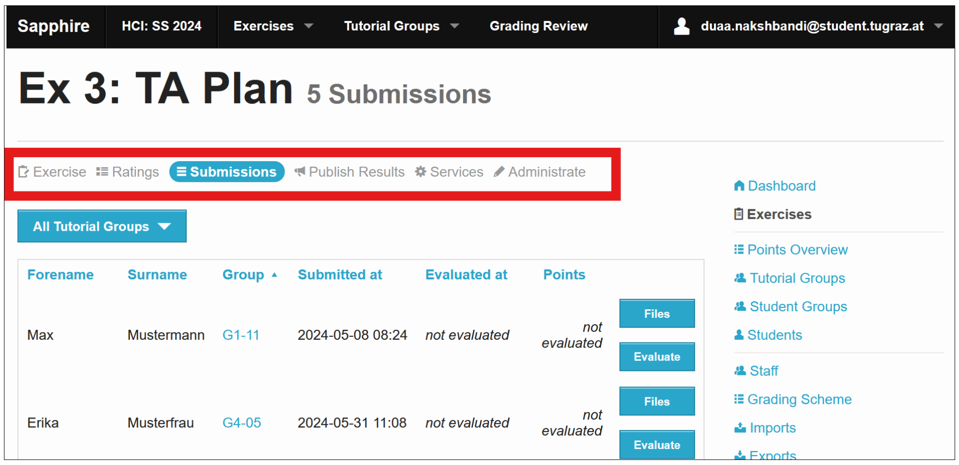
MNr. or Name

No account selected. Start searching for accounts by typing in the search field above

Role
Tutor
Tutor
Lecturer

Cancel Add New Member

Figure 4.7: The Dropdown Menu in Sapphire's Add New Staff Member View. [Screenshot taken by the author.]


Sapphire HCI: SS 2024 Exercises Tutorial Groups Grading Review duaa.nakshbandi@student.tugraz.at

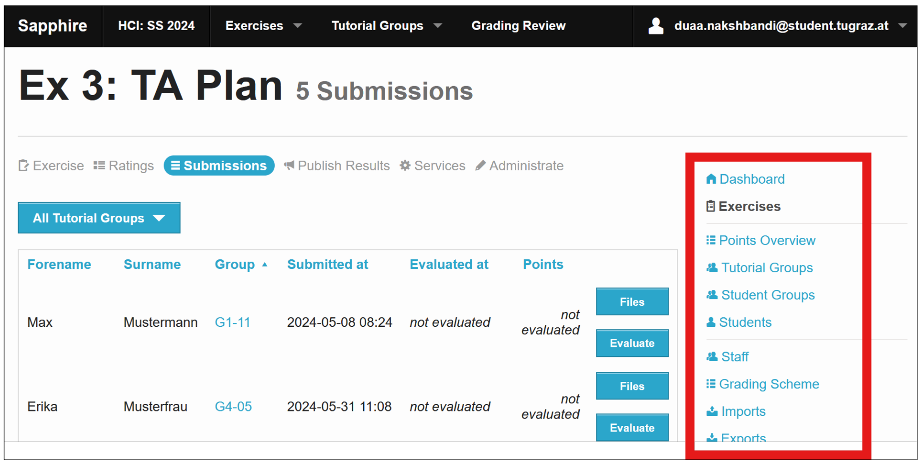
Ex 3: TA Plan 5 Submissions

Exercise Ratings Submissions Publish Results Services Administrate

All Tutorial Groups

Forename	Surname	Group	Submitted at	Evaluated at	Points	
Max	Mustermann	G1-11	2024-05-08 08:24	*not evaluated*	*not evaluated*	Files Evaluate
Erika	Musterfrau	G4-05	2024-05-31 11:08	*not evaluated*	*not evaluated*	Files Evaluate

Figure 4.8: Six Tabs in Sapphire's Submissions View. [Screenshot taken by the author.]



Sapphire | HCI: SS 2024 | Exercises | Tutorial Groups | Grading Review | duaa.nakshbandi@student.tugraz.at

Ex 3: TA Plan 5 Submissions

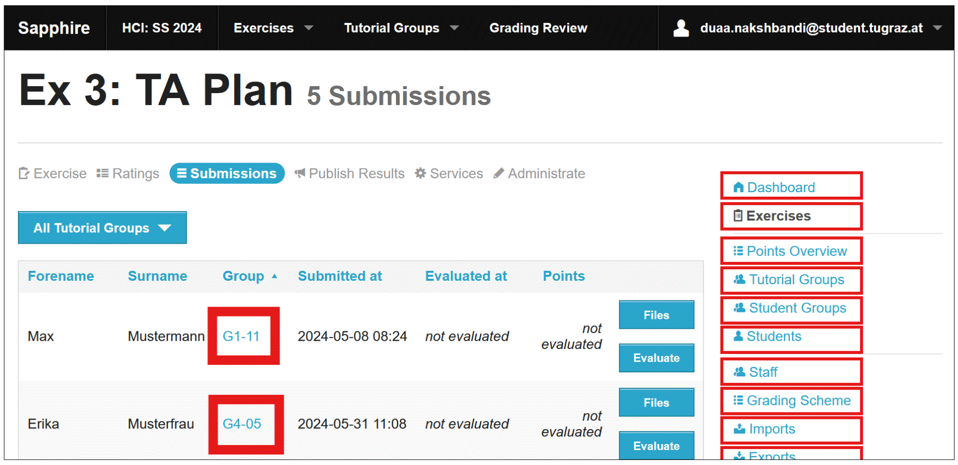
Exercise · Ratings · **Submissions** · Publish Results · Services · Administrate

All Tutorial Groups

Forename	Surname	Group ▲	Submitted at	Evaluated at	Points	
Max	Mustermann	G1-11	2024-05-08 08:24	*not evaluated*	*not evaluated*	Files Evaluate
Erika	Musterfrau	G4-05	2024-05-31 11:08	*not evaluated*	*not evaluated*	Files Evaluate

Dashboard
Exercises
Points Overview
Tutorial Groups
Student Groups
Students
Staff
Grading Scheme
Imports
Exports

Figure 4.9: The Link Side Bar in the Sapphire Submissions View, containing various links to other views. [Screenshot taken by the author.]



Sapphire | HCI: SS 2024 | Exercises | Tutorial Groups | Grading Review | duaa.nakshbandi@student.tugraz.at

Ex 3: TA Plan 5 Submissions

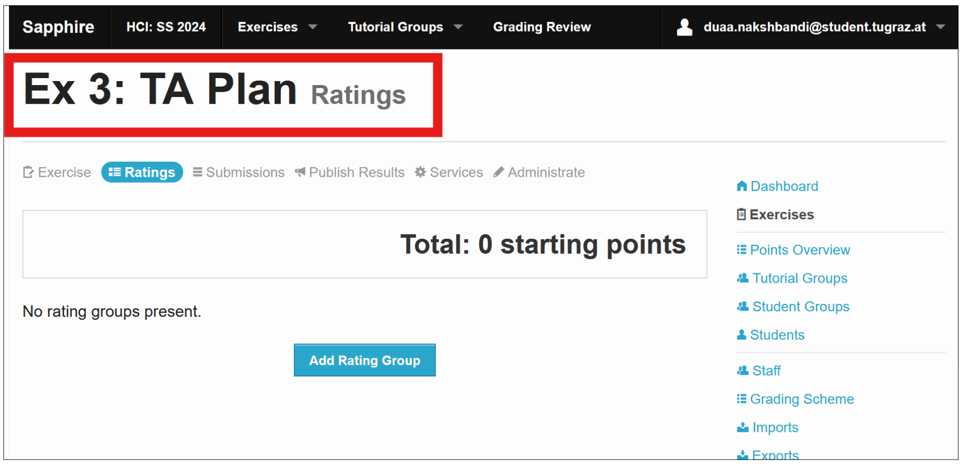
Exercise · Ratings · **Submissions** · Publish Results · Services · Administrate

All Tutorial Groups

Forename	Surname	Group ▲	Submitted at	Evaluated at	Points	
Max	Mustermann	G1-11	2024-05-08 08:24	*not evaluated*	*not evaluated*	Files Evaluate
Erika	Musterfrau	G4-05	2024-05-31 11:08	*not evaluated*	*not evaluated*	Files Evaluate

Dashboard
Exercises
Points Overview
Tutorial Groups
Student Groups
Students
Staff
Grading Scheme
Imports
Exports

Figure 4.10: Links in the Sapphire Submissions View. Links can be seen in the data table to the left and the Link Side Bar to the right. [Screenshot taken by the author.]



Sapphire | HCI: SS 2024 | Exercises | Tutorial Groups | Grading Review | duaa.nakshbandi@student.tugraz.at

Ex 3: TA Plan Ratings

Exercise · **Ratings** · Submissions · Publish Results · Services · Administrate

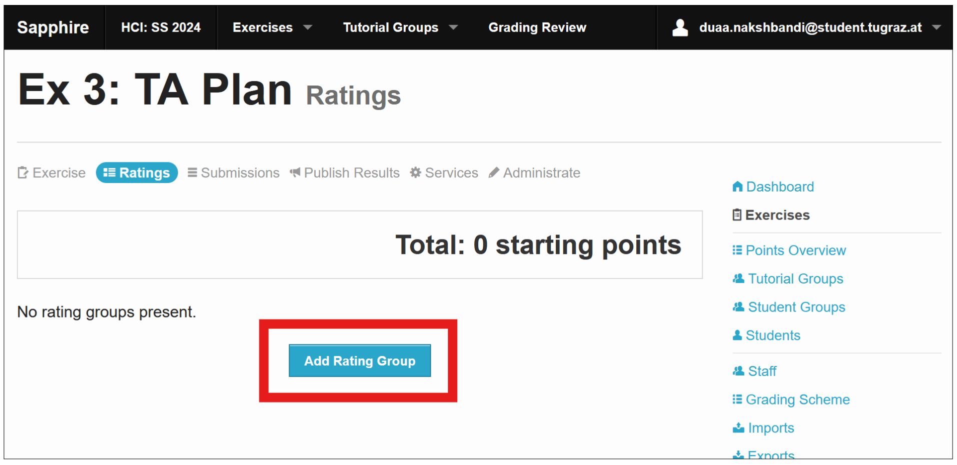
Total: 0 starting points

No rating groups present.

Add Rating Group

Dashboard
Exercises
Points Overview
Tutorial Groups
Student Groups
Students
Staff
Grading Scheme
Imports
Exports

Figure 4.11: A Heading used in the Sapphire Ratings View. [Screenshot taken by the author.]

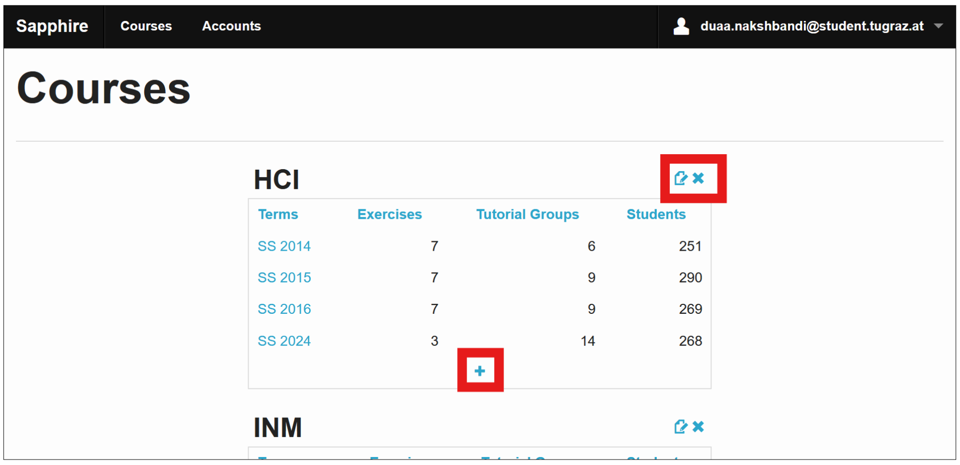


Ex 3: TA Plan Ratings

Total: 0 starting points

No rating groups present.

Figure 4.12: The Button used in Sapphire's Ratings View. [Screenshot taken by the author.]



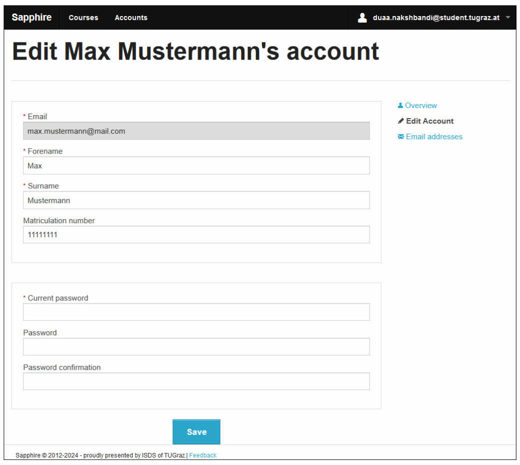
Courses

HCI

Terms	Exercises	Tutorial Groups	Students
SS 2014	7	6	251
SS 2015	7	9	290
SS 2016	7	9	269
SS 2024	3	14	268

INM

Figure 4.13: Icons used in Sapphire's Courses View. [Screenshot taken by the author.]



Edit Max Mustermann's account

* Email	max.mustermann@mail.com
* Forename	Max
* Surname	Mustermann
Matriculation number	11111111
* Current password	
Password	
Password confirmation	

Sapphire © 2012-2024 - proudly presented by ISDS of TUGraz | Feedback

Figure 4.14: The Sapphire Edit Account Form contains a number of Text Fields to capture textual input. [Screenshot taken by the author.]

Chapter 5

Improvements to Sapphire

"The biggest room in the world is the room for improvement."

[Helmut Schmidt]

As part of this thesis, a number of improvements were made to Sapphire, with the focus on implementing two main features: 1) renaming files and folders and 2) enabling drag-and-drop functionality for files and folders within the submission tree view. This chapter describes the implementation approach of these improvements.

5.1 Renaming Files and Folders

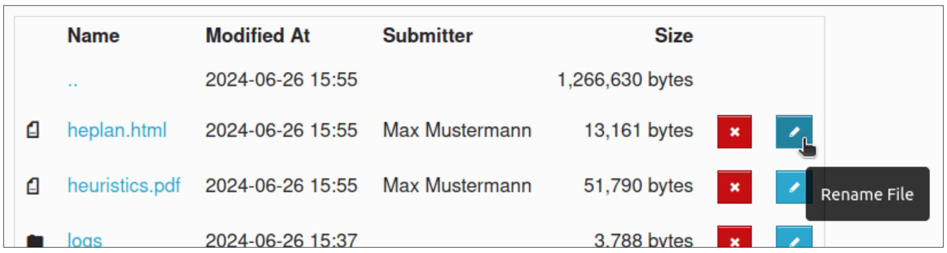
Before the renaming feature was implemented in Sapphire, users could not change file and folder names directly in the application. If a user uploaded a file named `he_plan.html` and later realized they needed to rename it to `heplan.html`, they could not do so in Sapphire directly. Instead, they had to delete the originally uploaded file and re-upload the renamed file. This was time-consuming, error-prone, and definitely not user-friendly.

5.1.1 The MVC Approach for File and Folder Renaming

For the implementation of the renaming feature in Sapphire, the Model View Controller (MVC) architecture was extended. The `submission_asset_rename.rb` and `submission_folder_rename.rb` classes were created in the `app/models/` directory to provide the logic for renaming the assets and folders. These models interact with the database to store and retrieve data related to the renaming process. The `app/views/` directory contains templates for displaying user interfaces. The `_submission_directory.html.erb` file defines the submission tree structure that allows the user to navigate through folders and assets. The `new.html.erb` files in the `submission_folder_renames/` and `submission_assets_renames/` subdirectories provide forms in which users can enter new names for folders and assets. The `submission_assets_renames_controller.rb` and `submission_folder_renames_controller.rb` classes were created in the `app/controllers/` directory to manage renaming actions for assets (files) and folders respectively. These controllers handle the user requests, interact with the respective models, and render the respective views.

5.1.2 Changes to Model Files for File and Folder Renaming

To implement the file and folder renaming feature, some modifications were needed to be made to the model files. The model files are stored in the `app/models/` directory. Two new classes, `SubmissionAssetRename` and `SubmissionFolderRename`, were introduced to handle the renaming logic for submission assets



Name	Modified At	Submitter	Size
..	2024-06-26 15:55		1,266,630 bytes
heplan.html	2024-06-26 15:55	Max Mustermann	13,161 bytes
heuristics.pdf	2024-06-26 15:55	Max Mustermann	51,790 bytes
logs	2024-06-26 15:37		3,788 bytes

Figure 5.1: The Rename File Button in the submission tree view. [Screenshot taken by the author.]

and folders, respectively. These classes make use of the ActiveModel functionality of Rails and include validations to ensure the handling of only valid renaming requests. ActiveModel is a module in Rails that provides a set of features such as validations and serialization [Rails 2024a].

In SubmissionAssetRename, a new model was created to handle the renaming of submission assets. Model attributes such as `submission_asset`, `submission`, `new_filename`, `renamed_at`, `filename_old`, and `renamed_by` were added with corresponding validations. Furthermore, other custom validations were added to ensure the new filename is unique within the submission and path. The `save!` method retains the original file name if no new name was provided and updates the `filename` of the `submission_asset` if the object is valid.

In SubmissionFolderRename, a new model was created to manage folder renaming within submissions. The model attributes include `submission`, `directory`, `renamed_at`, `path_old`, `path_new`, and `renamed_by`. Several custom validations were added to ensure that the directory is not the root folder, that it has been created, that the new name is different from the current name, and that the new name is not taken by another directory at the same level. The `save!` method checks for validation and then updates the paths of all assets within the folder to reflect the new folder name. The method ensures all assets are valid and saves the changes to the database.

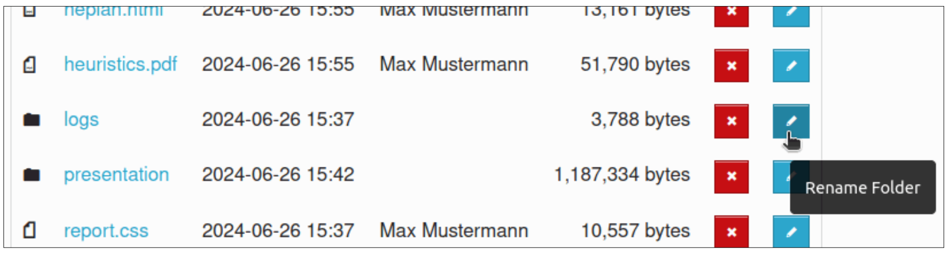
5.1.3 Changes to View Files for File and Folder Renaming

The implementation of the file and folder renaming feature in Sapphire involved the addition and modification of view files within the project's views. The view files are stored in the `app/views/` directory. In a Rails project, views are typically written in HTML with embedded Ruby (ERB). The view layer is responsible for displaying the user interface. The modifications made to the view files for the renaming feature included the addition of new forms, textual input fields, and buttons, as well as the modification of existing sidebars and view files.

The submission tree view file `app/views/submission_tree/_submission_directory.html.erb` was modified to include an Edit button (pencil icon) for renaming files and folders. Edit buttons are displayed next to each file or folder, but only if the student has the rights to modify the submission. Each Edit button links to the rename form of the respective file or folder.

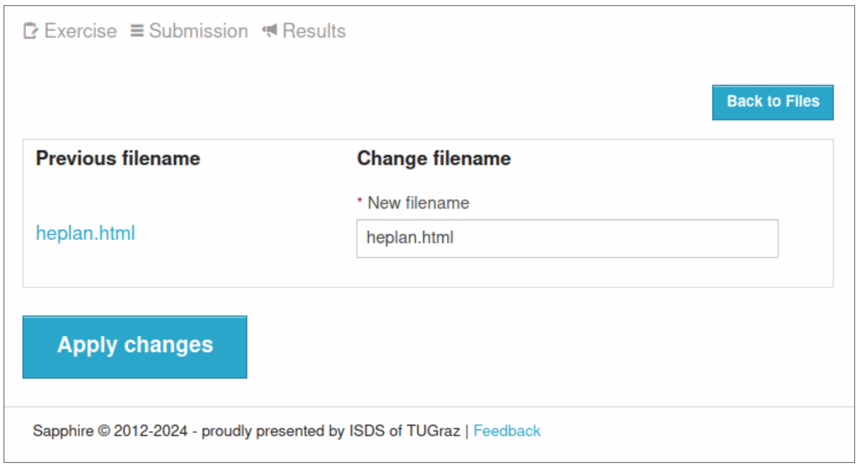
In `app/views/submission_assets_renames/new.html.erb`, the view now includes a form for renaming a submission asset (file). The form pre-fills the new file name with the current file name and provides a submit button called Apply changes to set the new file name. It also includes a Back to Files button that directs the user back to the containing folder. Similarly, the `app/views/submission_folder_renames/new.html.erb` file now features a form for renaming directories. It pre-fills the new folder name with the current folder name and provides a submit button called Apply changes to set the new folder name. It also includes a Back to Files button that directs the user back to the containing folder.

The view file `app/views/events/submission/_submission_assets_changes.html.erb` was changed to log an event whenever a submission folder or asset is renamed. The code checks the `content_type` of the



heplan.html	2024-06-26 15:55	Max Mustermann	13,161 bytes	
heuristics.pdf	2024-06-26 15:55	Max Mustermann	51,790 bytes	
logs	2024-06-26 15:37		3,788 bytes	
presentation	2024-06-26 15:42		1,187,334 bytes	
report.css	2024-06-26 15:37	Max Mustermann	10,557 bytes	

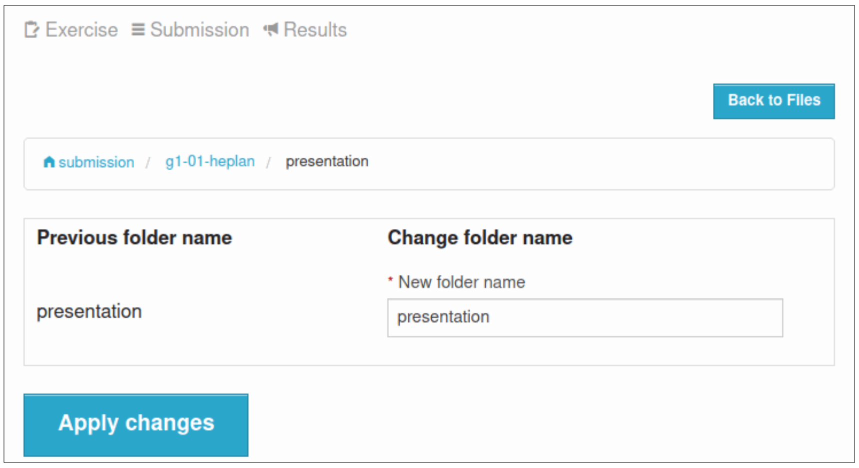
Figure 5.2: The Rename Folder button in the submission tree view. [Screenshot taken by the author.]



Previous filename	Change filename
heplan.html	* New filename: heplan.html

Sapphire © 2012-2024 - proudly presented by ISDS of TUGraz | Feedback

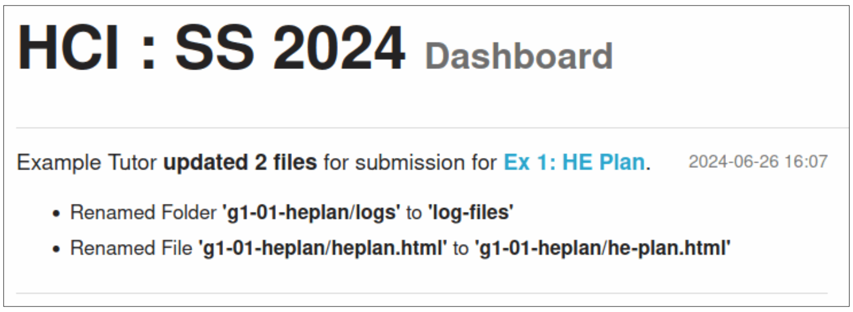
Figure 5.3: The Rename File Form. [Screenshot taken by the author.]



submission / g1-01-heplan / presentation

Previous folder name	Change folder name
presentation	* New folder name: presentation

Figure 5.4: The Rename Folder Form. [Screenshot taken by the author.]


HCI : SS 2024 Dashboard

Example Tutor **updated 2 files** for submission for Ex 1: HE Plan. 2024-06-26 16:07

- Renamed Folder **'g1-01-heplan/logs'** to **'log-files'**
- Renamed File **'g1-01-heplan/heplan.html'** to **'g1-01-heplan/he-plan.html'**

Figure 5.5: Event logging for file and folder renaming events. [Screenshot taken by the author.]

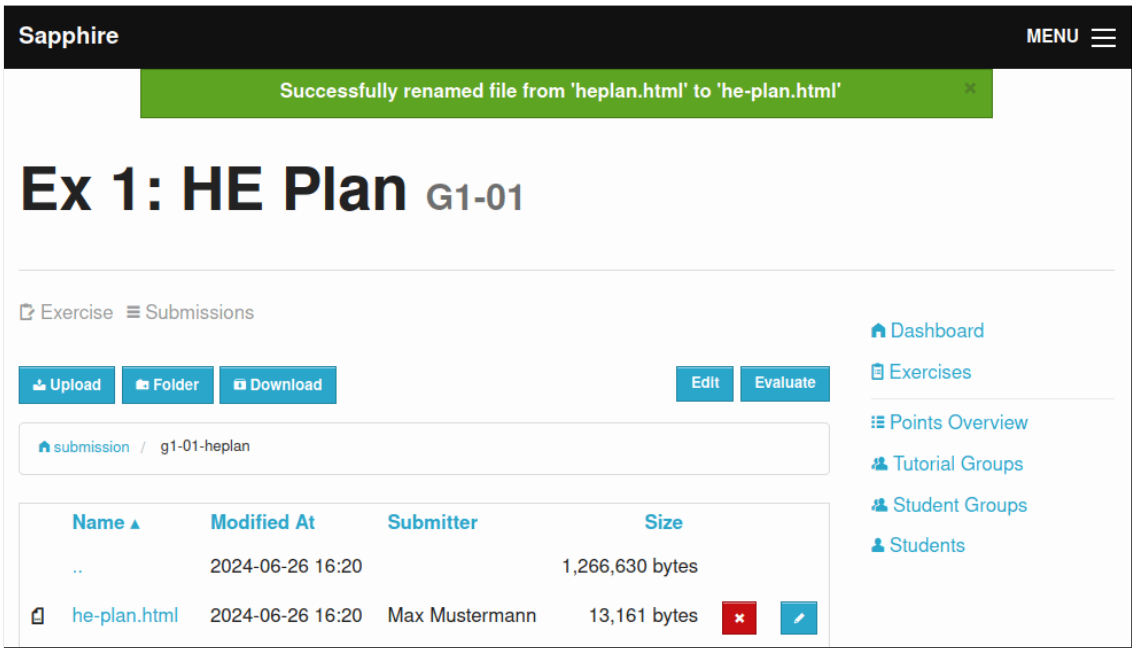
modified asset to differentiate between folders and files. If the modified asset is a folder, it logs an event indicating a folder rename, showing the old and new folder names. Otherwise, it logs an event indicating a file rename, displaying the old and new file names.

5.1.4 Changes to Controller Files for File and Folder Renaming

As part of implementing the file and folder renaming feature, two controllers were created: `submission_assets_renames_controller.rb` and `submission_folder_renames_controller.rb`. These controllers are responsible for executing, authorizing, and validating renaming requests. The controller files are stored in the `app/controllers/` directory.

The controller class SubmissionAssetsRenamesController in file `app/controllers/submission_assets_renames_controller.rb` includes the EventSourcing module and sets the context for each request by defining a `before_action` callback to load the necessary submission asset and related data. EventSourcing is a pattern where changes to the application's state are captured and stored as a series of set events, to allow historical traceability and the ability to reconstruct past states. The `new` action initializes a new SubmissionAssetRename object with the current timestamp, the old filename, the associated submission asset, and submission details, and then authorizes the rename operation. The `create` action creates a new SubmissionAssetRename object with the old filename, the current timestamp, and the account performing the rename, merging in any additional parameters permitted by the `rename_params` method. After setting the relevant `submission` and `submission_asset` associations and authorizing the rename operation, the action attempts to save the rename object. If successful, it triggers the event service to record the rename event and redirects to the submission tree view with a success message; otherwise, it redirects with an error alert indicating an invalid filename.

The controller class SubmissionFolderRenamesController in file `app/controllers/submission_folder_renames_controller.rb` also includes the EventSourcing module and sets the context for each request by defining `before_action` callbacks to load the necessary `submission`, `exercise`, `term`, `tree`, and `directory` data. The `new` action initializes a new SubmissionFolderRename object with the current timestamp, the old path, and the account performing the rename, associating it with the relevant `submission` and `directory`, and then authorizes the rename operation. The `create` action creates a new SubmissionFolderRename object with the old path, the current timestamp, and the account performing the rename, merging in any additional parameters permitted by the `submission_folder_rename_params` method. After setting the relevant `submission` and `directory` associations and authorizing the rename operation, the action attempts to save the rename object. If successful, it triggers the event service to record the rename event and redirects to the submission tree view with a success message, specifying the old and new paths of the renamed directory. If unsuccessful, it redirects with an error alert indicating that the chosen name is already in use.

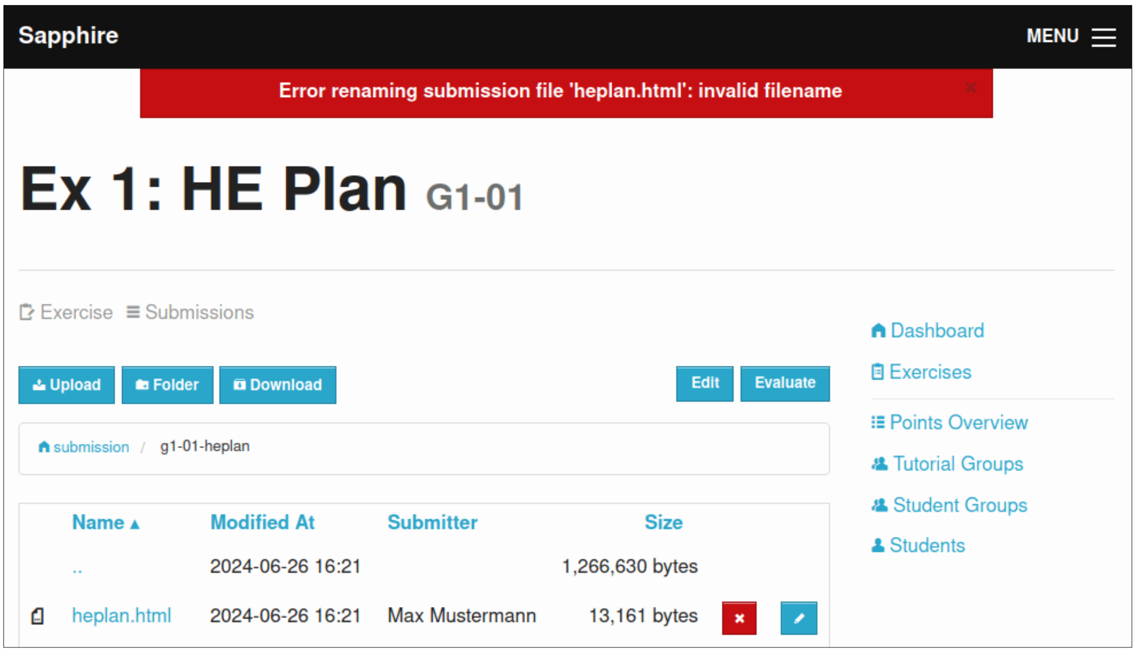

Successfully renamed file from 'heplan.html' to 'he-plan.html'

Ex 1: HE Plan G1-01

submission / g1-01-heplan

	Name ▲	Modified At	Submitter	Size
	..	2024-06-26 16:20		1,266,630 bytes
	he-plan.html	2024-06-26 16:20	Max Mustermann	13,161 bytes

Figure 5.6: Success message shown to the user after successfully renaming a file. [Screenshot taken by the author.]

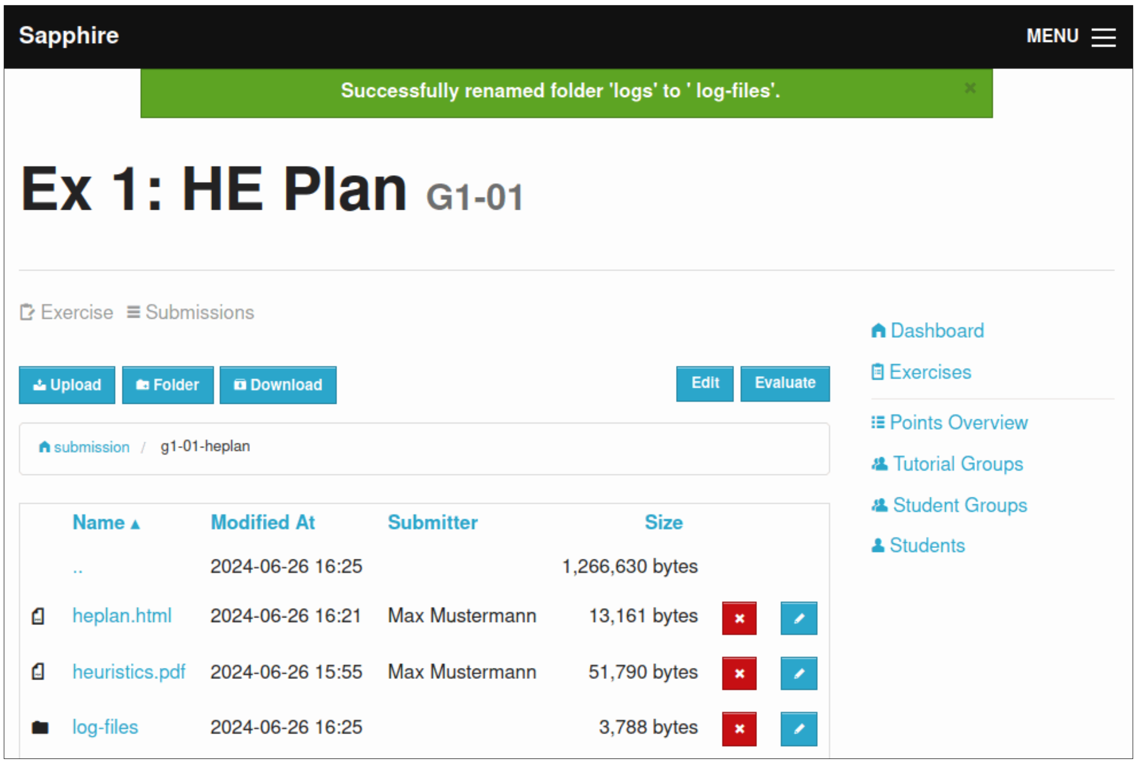

Error renaming submission file 'heplan.html': invalid filename

Ex 1: HE Plan G1-01

submission / g1-01-heplan

	Name ▲	Modified At	Submitter	Size
	..	2024-06-26 16:21		1,266,630 bytes
	heplan.html	2024-06-26 16:21	Max Mustermann	13,161 bytes

Figure 5.7: Fail message shown to the user after choosing an invalid new file name. [Screenshot taken by the author.]

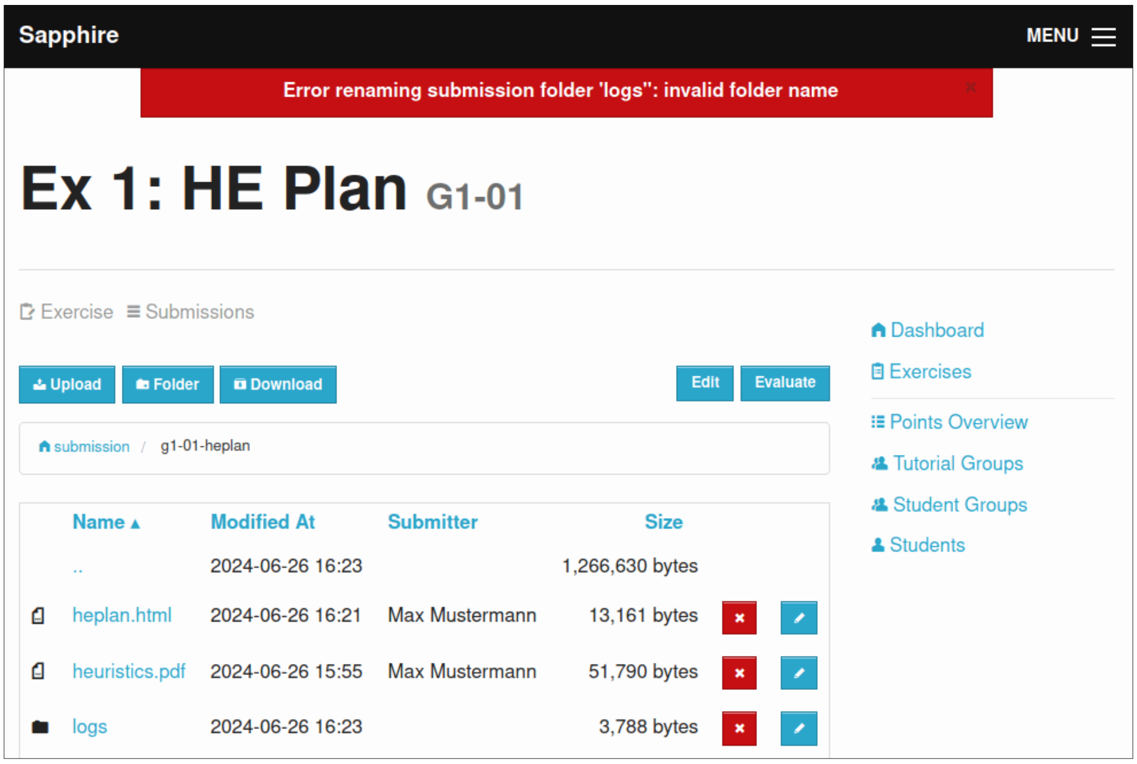


Successfully renamed folder 'logs' to ' log-files'.

Ex 1: HE Plan G1-01

Name ▲	Modified At	Submitter	Size
..	2024-06-26 16:25		1,266,630 bytes
heplan.html	2024-06-26 16:21	Max Mustermann	13,161 bytes
heuristics.pdf	2024-06-26 15:55	Max Mustermann	51,790 bytes
log-files	2024-06-26 16:25		3,788 bytes

Figure 5.8: Success message shown to the user after successfully renaming a folder. [Screenshot taken by the author.]



Error renaming submission folder 'logs'': invalid folder name

Ex 1: HE Plan G1-01

Name ▲	Modified At	Submitter	Size
..	2024-06-26 16:23		1,266,630 bytes
heplan.html	2024-06-26 16:21	Max Mustermann	13,161 bytes
heuristics.pdf	2024-06-26 15:55	Max Mustermann	51,790 bytes
logs	2024-06-26 16:23		3,788 bytes

Figure 5.9: Fail message shown to the user after choosing an invalid new folder name. [Screenshot taken by the author.]

```ruby
 1 # Define a route for folder renaming actions
 2 resource :folder_rename, controller: :submission_folder_renames, only:
 3   [:new, :create]
 4 
 5 # Define routes for submission assets
 6 resources :submission_assets, only: [:show, :update, :destroy] do
 7 member do
 8   post 'rename', to: 'submission_assets_renames#create', as: :create_rename
 9   get 'rename', to: 'submission_assets_renames#new', as: :new_rename
10 end
11 end
```

Listing 5.1: Ruby code snippet defining routes for the actions to rename folder and submission assets.

5.1.5 Configuring Routes for File and Folder Renaming

In Rails, the `config/routes.rb` file serves as the central configuration point for defining the application's routing system. This routing system determines how incoming requests are mapped to controllers and actions within the application. By default, a call to the `new` action within a controller creates a new unsaved record and renders a form to collect input for its contents from the end user. The controller's `create` action takes the newly filled record and saves it to the database. Listing 5.1 shows how the routes were defined to work with the renaming function in Sapphire. In lines 1 to 3, a new route is added to handle folder renaming actions. It redirects the user request to the `new` and `create` actions within the SubmissionFolderRenamesController. In lines 5 to 11, the routes `new_rename` and `create_rename` redirect asset (file) renaming requests to the `new` and `create` actions in the SubmissionAssetsRenamesController.

5.1.6 Authorization Logic for File and Folder Renaming

Sapphire uses the Pundit library [Pundit 2024] to manage action authorization and policies. The Pundit library is a tool for Rails applications that enables developers to define which users are allowed to perform specific actions. Defining policies in Pundit involves creating Ruby classes to encapsulate the authorization logic for various actions related to the application's models. Each policy class typically corresponds to a model in the application. The SubmissionAssetRenamePolicy and SubmissionFolderRenamePolicy classes were implemented to determine whether a user is allowed to rename submission assets and folders.

The SubmissionAssetRenamePolicy checks if a user is authorized to rename a submission asset. The authorization logic first verifies if the user has staff permissions for the specific academic term associated with the submission. If the user has these permissions, they are authorized to proceed. If the user does not have staff permissions, the system then checks if the user is one of the students associated with the submission and if the submission allows modifications by students. If both conditions are met, the user is authorized to rename the asset.

Similarly, the SubmissionFolderRenamePolicy verifies that the user is authorized to rename a folder. The authorization logic first checks if the user has general staff permissions. If the user has these permissions, they are allowed to proceed. If the user does not have staff permissions, the system checks if the user is one of the students linked to the submission and if the submission allows students to make modifications. If both conditions are satisfied, the user is authorized to rename the folder.

5.1.7 Event Logs for File and Folder Renaming

The event service in Sapphire is responsible for logging and reporting actions and changes in the application as events, as can be seen in. The logic for the event service can be found in the `app/services/event_service.rb` file. To log `events` related to the renaming functionality, the `app/services/event_service.rb` file was extended with the `submission_folder_renamed!` and `submission_asset_renamed!` methods. These methods notify the Sapphire system whenever a submission file or folder has been renamed.

The `submission_folder_renamed!` first checks if there is a recent update `event` for the `submission` associated with the renamed folder by querying `Events::Submission::Updated`. If no such `event` exists, it creates a new one. The method then initializes or retrieves the `submission_assets` hash and categorizes assets as `added`, `updated`, or `destroyed`. It adds an entry to the `updated` category with the old and new paths of the renamed folder. The method updates the `event` with the new `submission_assets` data, sets the current time as the `updated_at` timestamp, and saves the `event`.

The `submission_asset_renamed!` method handles the `event` when a `submission_asset` is renamed. It first locates the SubmissionAsset object corresponding to the renamed asset. The method retrieves the submission related to the rename and checks for a recent update `event`, similar to the folder rename method. If no `event` is found, it creates a new one. The method initializes or retrieves the `submission_asset` hash and adds an entry to the `updated` category, including the old and new filenames, the asset's path, and its content type. Finally, the method updates the `event` with the new `submission_asset`'s data, sets the current time as the `updated_at` timestamp, and saves the `event`.

The `app/views/events/submission/_submission_assets_changes.html.erb` file renders logs regarding changes in the system. This file needs to be extended to render logs related to the renaming of submission's files and folders. When a renaming `event` occurs, the template generates an appropriate message to be displayed in the view. The template processes each change by checking the `content_type` of the change object to determine whether it pertains to a folder or a file. If the `content_type` is an array, this indicates that the change involves a folder rename. In this case, the template generates a log message stating that a folder was renamed, and uses the `submission_asset_file_change` helper method to format and display the old and new folder names. This method takes the path and the file name (old or new) as arguments and returns a properly formatted string for display. If the `content_type` is not an array, this indicates a file rename. The template then generates a log message stating that a file was renamed, again using the `submission_asset_file_change` helper method to format and display the old and new file names.

5.1.8 RSpecs for File and Folder Renaming

Various RSpec tests were implemented to ensure the correct functionality and full test coverage of the implemented changes for the file and folder renaming feature. These changes span the controller, factory, model, policy, and service specs. RSpec is a testing framework for Ruby that is used to write and execute test cases to ensure that the code behaves as expected [RSpec 2024].

The `spec/controllers/submission_assets_renames_controller_spec.rb` file tests the `new` and `create` actions for renaming submission assets. The `GET#new` action verifies the correct assignment of attributes to a SubmissionAssetRename object and checks for a `HTTP302` status code. The `POST#create` action handles both valid and invalid parameters, ensuring proper redirection and flash messages based on the success or failure of the renaming process. Furthermore, the `#set_context` method is tested to confirm the proper setting of instance variables, and `#rename_params` ensures the correct permitting of parameters.

Similarly, the `spec/controllers/submission_folder_renames_controller_spec.rb` tests the `new` and `create` actions for renaming submission folders. The `GET#new` action checks for a `HTTP200` status code and the correct assignment of attributes to a SubmissionFolderRename object, including submission and directory assignments. The `POST#create` action tests the success and failure scenarios of folder renaming, as well as the correct redirection and flash messages.

To initialize the attributes required for the renaming feature RSpecs, Ruby FactoryBot factories were implemented [Thoughtbot 2024]. The `spec/factories/directory_factory.rb` was created to define a factory for SubmissionStructure::Directory, allowing nodes to be set up dynamically. The `spec/factories/submission_asset_rename_factory.rb` and `spec/factories/submission_folder_rename_factory.rb` were defined to create SubmissionAssetRename and SubmissionFolderRename objects, respectively.

In `spec/models/submission_asset_rename_spec.rb`, attributes and validations are checked for the SubmissionAssetRename model. The spec ensures that attributes like `new_filename`, `renamed_at`, and `filename_old` are present and correctly validated. Delegation methods are tested to ensure correct behavior, and the `save!` method is tested for proper functionality under valid and invalid conditions. Similarly, `spec/models/submission_folder_rename_spec.rb` validates the attributes for SubmissionFolderRename and tests delegation methods to confirm correct term and student exercises.

Two policy specs test the various authorization scenarios for renaming actions. The `spec/policies/submission_asset_rename_policy_spec.rb` and `spec/policies/submission_folder_rename_policy_spec.rb` files define the authorization checks for different user roles, including admin, lecturer, tutor, and student roles. These specs ensure that only authorized users can perform renaming actions based on their association with the term and submission.

Lastly, the service spec in `spec/services/event_service_spec.rb` was updated to handle events triggered by renaming actions. The newly created methods `submission_folder_renamed!` and `submission_asset_renamed!` ensure that the appropriate events are created or updated when a folder or file is renamed. These methods validate the correct setup of event attributes, such as submission ID, exercise ID, and updated assets.

5.2 Moving Files and Folders by Drag and Drop

Before the implementation of the drag and drop feature, students were unable to move files or folders directly within the submission tree after uploading. Instead, if they wanted to reorganize their submissions, they had to delete the files or folders from their current location and then re-upload them to the desired location. The new drag and drop functionality addresses this issue by allowing students to easily and efficiently reorganize their submissions by dragging and dropping without the need for deletion and re-uploading.

5.2.1 Architecture of the Drag and Drop Feature

The core functionality of the drag and drop feature was implemented in `app/assets/javascripts/submission_assets/moves.coffee`. The JavaScript code (written in CoffeeScript) manages the drag and drop events, updates the user interface and interacts with the backend. The style declarations for the drag and drop feature are defined in `app/assets/stylesheets/submission_assets/moves.scss`.

The backend logic is managed by two controllers: `app/controllers/submission_assets/moves_controller.rb` and `app/controllers/submission_folder_moves_controller.rb`. The first handles the move operations for individual submission assets, while the second handles the move operations for entire folders within the submission tree. To ensure users have the necessary permissions to move files and folders, authorization policies were defined in `app/policies/submission_asset_policy.rb` and `app/policies/submission_structure/tree_policy.rb`. The first defines the permission for dragging and dropping individual submission assets within the submission tree structure, while the second defines the permission for dragging and dropping folders within the submission tree structure.

The view template `app/views/submission_tree/_submission_directory.html.erb` was updated to include the necessary HTML and embedded Ruby code implementing the drag and drop functionality. This view class renders the submission tree and integrates the frontend JavaScript logic. Finally, the `config/routes.rb` file was extended with routes for the drag and drop operations. The newly added routes direct requests to the appropriate controller actions and enable the backend logic to process the moves.

5.2.2 JavaScript Logic for Moving Files and Folders

The JavaScript logic for managing the dragging and dropping of files and folders within the submission tree was implemented in `app/assets/javascripts/submission_assets/moves.coffee`. Event listeners are registered for drag-and-drop operations on elements identified as either folders or files. This is accomplished by querying the document for elements with specific classes and attaching event listeners for various HTML drag-and-drop events, such as `dragstart`, `dragenter`, `dragover`, `dragleave`, and `drop`.

The `dragStart` function initiates the dragging process by capturing the ID of the dragged element and storing it in the `dataTransfer` object. This allows the dragged item's data to be accessible during the drop event. During the `dragOver` event, default behaviors are prevented to allow for a valid drop. The function checks if the target element is a folder, enabling the drop effect to be set to `move` and visually indicating the drag operation by adding an `over` class. The `dragEnter` function adds a visual cue when a dragged item enters a valid drop target, indicating that the element is ready to receive the drop. The `dragLeave` function removes the visual cue when the dragged item leaves a potential drop target.

The drop function is the core of the drag and drop logic. It captures the data transferred during the drag operation, identifies the source and target elements, and determines the appropriate action based on whether the source and target are files or folders. If the target is a folder, an AJAX POST request is made to move the file or folder to the new location. The success and error callbacks handle the outcomes, either redirecting the user to a new URL or logging errors for troubleshooting.

Additionally, a window-level event listener for dragover is added to handle automatic scrolling when the mouse is near the top or bottom of the window. This ensures that users can navigate through long lists of files and folders without interrupting the drag and drop operation.

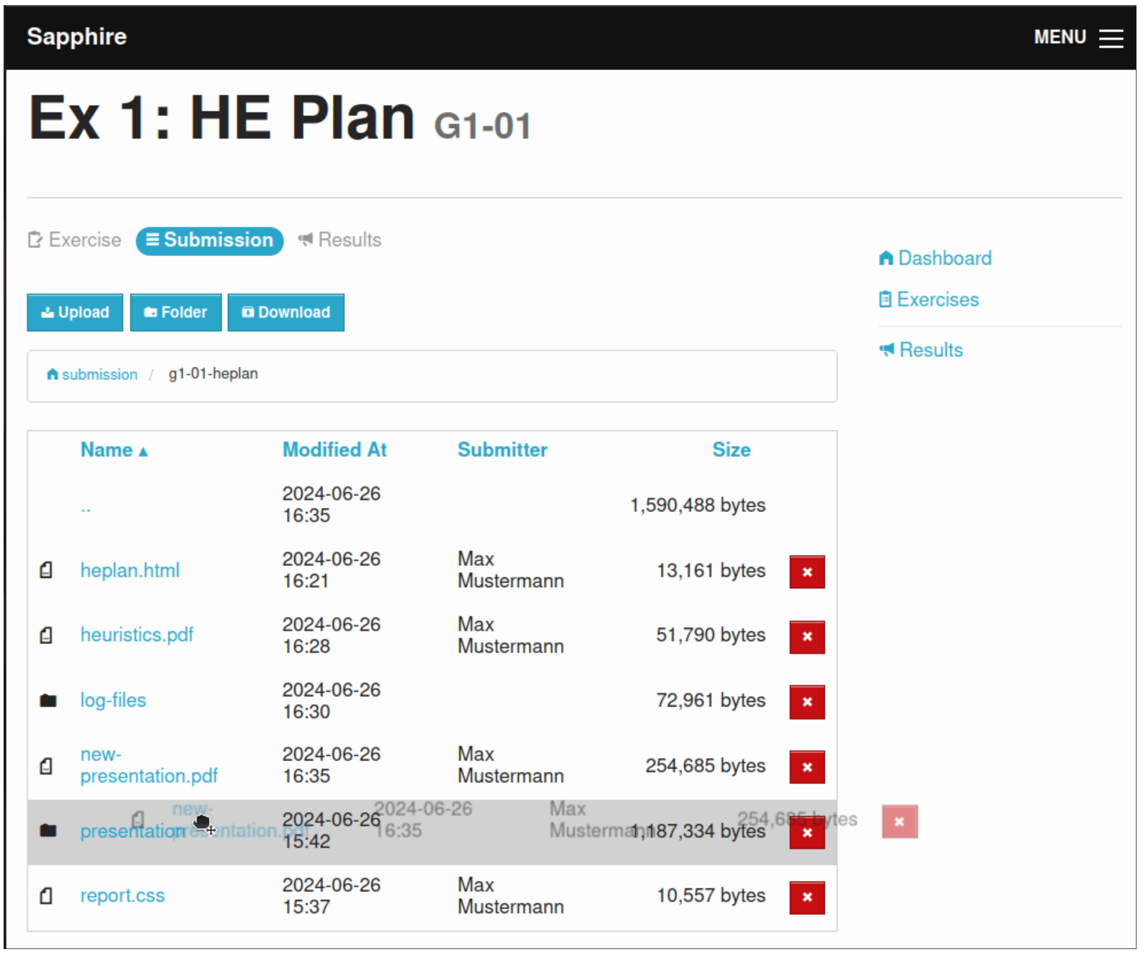
5.2.3 Styling for Moving Files and Folders

As a means to give visual feedback when a folder is in a state of being dragged over, the CSS class `.folder-entry.over` was defined in `app/assets/stylesheets/submission_assets/moves.scss`. To help users easily identify the target folder, the background color of the folder entry changes to a light gray shade (`#d0d0d0`).

5.2.4 Controller Logic for Moving Files and Folders

The SubmissionAssets::MovesController controller class is responsible for handling the movement of individual files within the submission tree. The process begins with setting up the necessary context through the `set_context` method. This method retrieves the relevant submission asset and its associated submission. Upon initiating a file move by the user, the `move` action is invoked. It first verifies if the current user is authorized to move the file. The existing path of the file is stored, and then the file's path is updated to the target location specified in the parameters. The controller attempts to save the updated file path. If successful, a success message is flashed, and a JSON response is returned, prompting a redirection to the updated file tree view. This visual feedback informs users that the file has been moved successfully. However, if the save operation fails, an error message is displayed, and the user is redirected to the previous state in the file tree.

The SubmissionFolderMovesController controller class handles scenarios where entire directories, possibly containing multiple nested files and folders, are moved to new locations within the submission tree. Similar to the file move controller, it begins by establishing the context using the `set_context` method. This method identifies the submission, retrieves the file tree, and resolves the target directory from the provided parameters.



Ex 1: HE Plan G1-01

	Name ▲	Modified At	Submitter	Size	
	..	2024-06-26 16:35		1,590,488 bytes	
	heplan.html	2024-06-26 16:21	Max Mustermann	13,161 bytes	×
	heuristics.pdf	2024-06-26 16:28	Max Mustermann	51,790 bytes	×
	log-files	2024-06-26 16:30		72,961 bytes	×
	new-presentation.pdf	2024-06-26 16:35	Max Mustermann	254,685 bytes	×
	presentation	2024-06-26 15:42		1,187,334 bytes	×
	report.css	2024-06-26 15:37	Max Mustermann	10,557 bytes	×

new-presentation.pdf 2024-06-26 16:35 Max Mustermann 254,685 bytes ×

Figure 5.10: Moving a submission file to a deeper-level sub-folder in the submission tree using the drag and drop feature. [Screenshot taken by the author.]

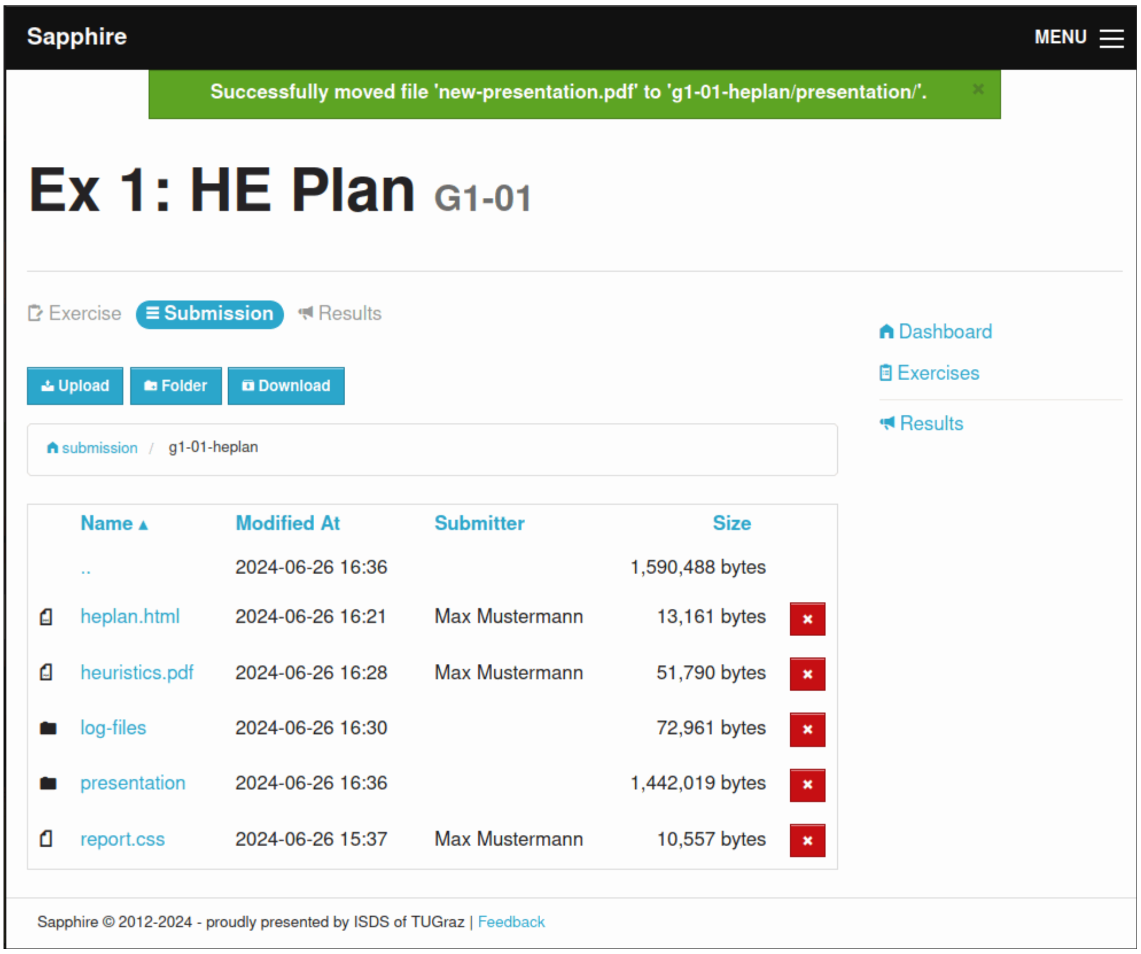
When a folder move is requested, the `move` action performs several critical checks and operations. It first ensures that the user has the necessary permissions to move the folder. The source and target paths are then analyzed to determine their base names and construct the new directory path where the folder will be moved. If the source and target paths are identical, the operation is aborted early to prevent a folder from being moved into itself.

The controller then checks if a folder with the same name already exists in the target location. If it does, an error message is flashed, and the operation is halted, preventing conflicts or overwriting. If no conflicts are detected, the controller proceeds to update the paths of all assets within the source folder, including its subfolders and files. Each asset's path is recalculated relative to the source and then adjusted to reflect its new position in the target directory.

After adjusting the paths, the controller attempts to save each asset. If all assets are successfully saved, a success message is displayed, and the user is redirected to view the updated folder structure. If any asset fails to save, an error message is displayed, and the user is redirected to the previous state.

5.2.5 Authorization Policies for Moving Files and Folders

The authorization logic for the dragging and dropping of files and folders within the submission tree was implemented within the two already existing policy files: `submission_asset_policy.rb` and `submission_structure/tree_policy.rb`. The authorization policies determine who has the ability to move files and folders within the submission tree.



Successfully moved file 'new-presentation.pdf' to 'g1-01-heplan/presentation/'.

Ex 1: HE Plan G1-01

submission / g1-01-heplan

	Name ▲	Modified At	Submitter	Size	
	..	2024-06-26 16:36		1,590,488 bytes	
	heplan.html	2024-06-26 16:21	Max Mustermann	13,161 bytes	×
	heuristics.pdf	2024-06-26 16:28	Max Mustermann	51,790 bytes	×
	log-files	2024-06-26 16:30		72,961 bytes	×
	presentation	2024-06-26 16:36		1,442,019 bytes	×
	report.css	2024-06-26 15:37	Max Mustermann	10,557 bytes	×

Sapphire © 2012-2024 - proudly presented by ISDS of TUGraz | Feedback

Figure 5.11: The user receives a success message after moving a submission file to a deeper-level sub-folder in the submission tree using the drag and drop feature. [Screenshot taken by the author.]

The `submission_asset_policy.rb` policy file was extended with a `move?` method that defines the conditions under which a user can move a submission asset. The policy checks for one of two conditions. Either the user has modifiable permission for the asset's submission, or has staff permissions for the term associated with the exercise to which the submission belongs.

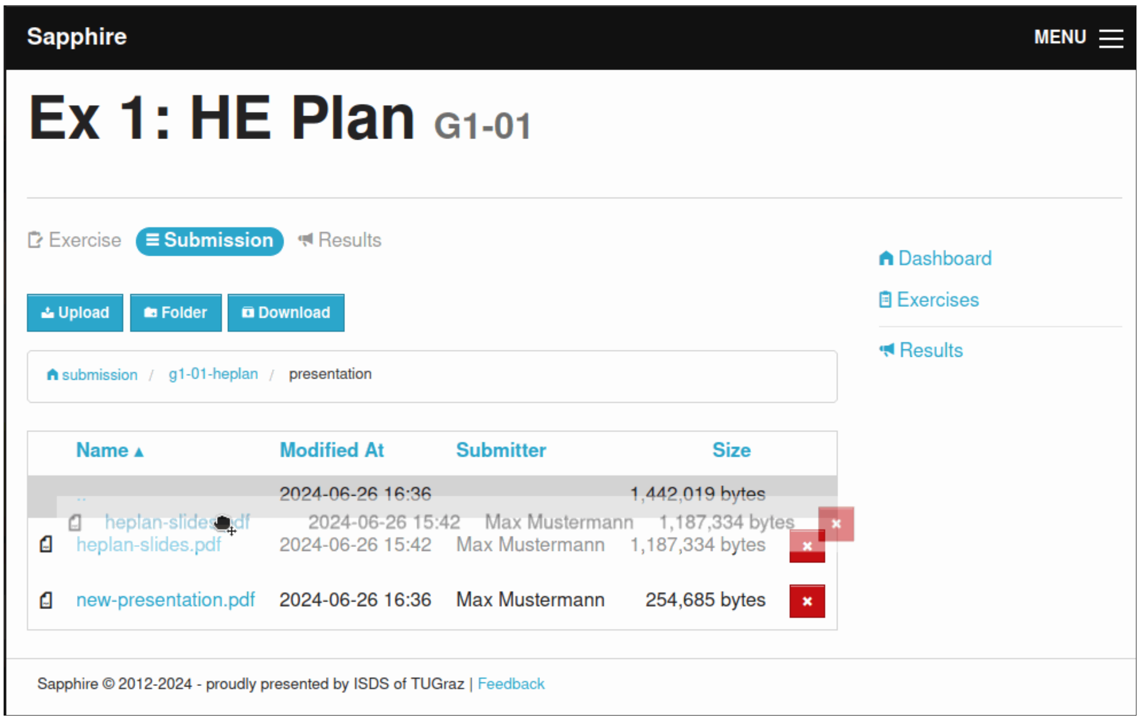
For dragging and dropping folders, the `submission_structure/tree_policy.rb` policy file was extended with a `move?` method. This method simply checks if the user has write permission. This simplification means that any user with the ability to write to the submission structure, typically those who can add or modify content, can move folders within the submission tree.

5.2.6 View Changes for Moving Files and Folders

To enable dragging and dropping files and folders within the submission tree in the UI, the tree content rows HTML elements in the `_submission_directory.html.erb` view file were modified. The `draggable="true"` attribute to each row representing a directory or file was added. This change enables users to click and drag these elements across the tree interface.

5.2.7 Routing for Moving Files and Folders

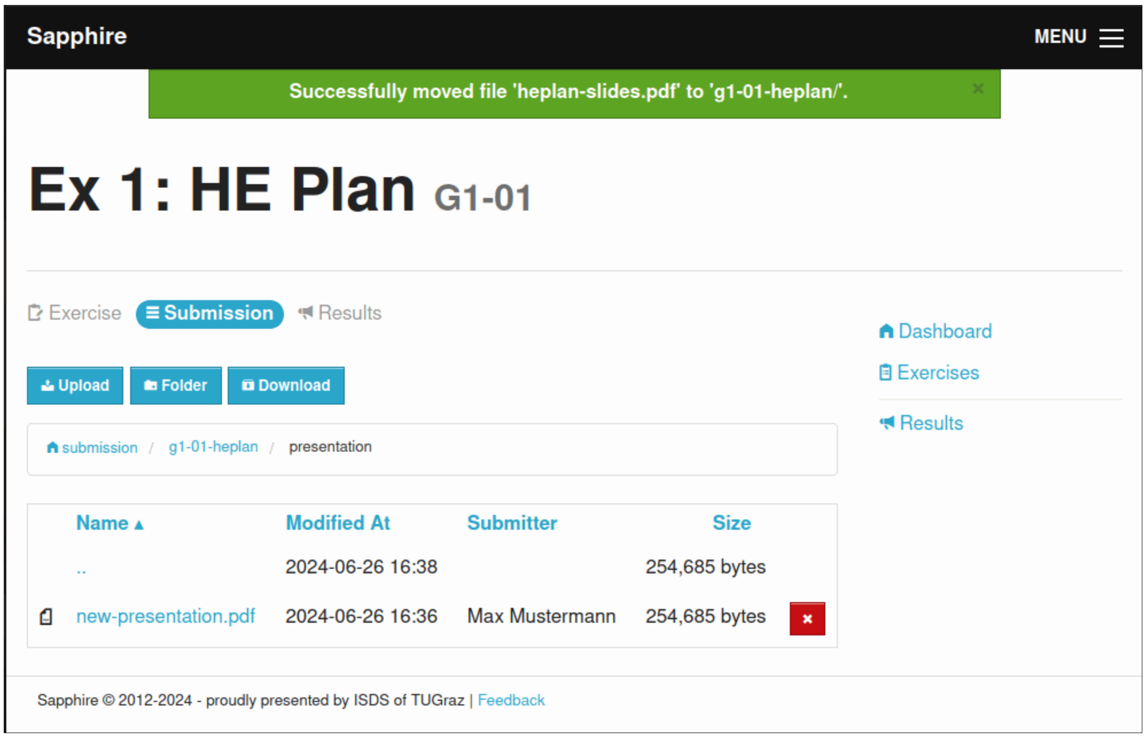
To handle the moving of files, the routing configuration within `config/routes.rb` was expanded to include specific routes for moving submission assets. The first addition is a direct POST route defined as `submission_assets/move`, which maps to the `move` action within the `submission_assets/moves` controller.



Name ▲	Modified At	Submitter	Size
..	2024-06-26 16:36		1,442,019 bytes
heplan-slides.pdf	2024-06-26 15:42	Max Mustermann	1,187,334 bytes
new-presentation.pdf	2024-06-26 16:36	Max Mustermann	254,685 bytes

Sapphire © 2012-2024 - proudly presented by ISDS of TUGraz | Feedback

Figure 5.12: Moving a submission file to a higher-level folder in the submission tree using the drag and drop feature. [Screenshot taken by the author.]



Successfully moved file 'heplan-slides.pdf' to 'g1-01-heplan/'.

Name ▲	Modified At	Submitter	Size
..	2024-06-26 16:38		254,685 bytes
new-presentation.pdf	2024-06-26 16:36	Max Mustermann	254,685 bytes

Sapphire © 2012-2024 - proudly presented by ISDS of TUGraz | Feedback

Figure 5.13: The user receives a success message after moving a submission file to a higher-level folder in the submission tree using the drag and drop feature. [Screenshot taken by the author.]

To handle the moving of folders, the `resources :submissions` block was extended to include an additional member route to handle the moving of folders within the submission's tree structure. This member route is expressed as a POST route with the pattern `tree(/*path)/move`, targeting the move action within the `submission_folder_moves` controller. The wildcard `/*path` enables this route to dynamically capture and process various paths within the tree.

Chapter 6

Tutorials for Sapphire Admins

“Risk comes from not knowing what you’re doing.”

[Warren Buffett]

This chapter contains tutorials for prospective Sapphire admins about various administration and development tasks in Sapphire. The procedures were documented by former Sapphire admins Amel Smajic and Duaa Nakshbandi.

6.1 Administrative Tutorials

At the beginning of each academic semester, the Sapphire admin needs to perform various administrative tasks to prepare the course in Sapphire. The administrative tasks do not involve coding and are done through the Sapphire user interface. These administrative tasks can only be executed by accounts with admin rights.

6.1.1 Establishing Tutorial Groups in Sapphire

At the beginning of each new academic semester, the Sapphire admin creates a tutorial group for each tutor. Admins are needed for this task because only they can assign the tutor role and form tutorial groups. These groups are typically named “T1” to “T*n*”, depending on the number of tutors for that semester. This example is based on the SS 2023 academic semester.

The Sapphire admin tutor typically receives an email before the start of the semester from the lecturer containing the details (group number, first name, last name, and email address) of each tutor. To set up these tutorial groups in Sapphire, the admin tutor needs to log in to the Sapphire production server and choose the specific academic term. After clicking the Tutorial Groups link in the sidebar, the New Tutorial Group button should be clicked. In the example, seven tutors were employed, leading to the creation of seven tutorial groups.

Next, the admin tutor must assign each tutor to their respective tutorial group. This is done by clicking the Staff link in the sidebar and then clicking the + Add button. The tutor’s name or matriculation number is entered into the search box, and the appropriate account is chosen. Then, the role Tutor is selected together with the corresponding tutorial group.

6.1.2 Creating Student Accounts on Sapphire

At the beginning of each semester, the Sapphire admin is responsible for setting up student accounts in Sapphire. This process begins with the lecturer exporting the current list of students (TN = Teilnehmer, in German) from the university's intranet system TUGRAZonline as an Excel (`.xls`) file. The Excel file must first be converted to a CSV file, setting the quote character to `"` and the column separator to `;`. The CSV file is then uploaded to Sapphire.

After choosing the relevant term for the course in Sapphire, the Imports link in the sidebar should be clicked. The CSV file from the previous step is then uploaded under the Input File panel. In the panel Data Representation Settings, the field Match tutorial and student groups must be selected, with the regex pattern:

```
\AG(?<tutorial>[\d]+)-(?<student>[\d]+)e?\z
```

to extract tutorial groups and student groups correctly. Ensure that the quotation mark and the column separator correspond to those used when creating the CSV file.

In the Import Students panel, the Send welcome notifications option should be used with caution since enabling it will result in automatic emails being sent to students' email addresses. Prior consultation with the lecturer about the appropriateness of sending welcome emails is essential to avoid sending unauthorized communications to students. After clicking the Next button, a preview table is displayed. Make sure to assign the correct heading to each column. Then start the import and look for errors, if any.

It is advisable to try these steps first on a separate staging server (StagingSapphire) rather than directly on the production server. Make sure in this case to replace the student emails in the TN list with fake emails so that no real student accounts are created on StagingSapphire; also disable the Send welcome notifications option as a second security measure, so no emails are inadvertently sent out to real student email addresses from the staging server.

6.1.3 Creating a Submission for an Individual Exercise After the Deadline

In the current Sapphire implementation, if a student misses the late deadline for an individual exercise, they cannot upload their Submission and must send the files to their tutor by email or hand them in on a USB stick. If a submission for the exercise was created by the student before the deadline, then the tutor can upload the files to the submission on the student's behalf, even after the deadline. However, if no submission was created by the student for the exercise before the deadline, an admin must take over the task of first creating an empty submission using a (somewhat messy) workaround.

A requirement for the workaround to create an empty submission after the deadline is a special account with both student and admin roles, but *without* the tutor role. It is helpful to have a designated admin account, to which the student role for a particular tutorial group and student group can be added and removed. The role should be added before the workaround and removed again afterwards. This special admin account with a temporary student role will be referred to as the `admin-student` account.

To create such a temporary `admin-student` account, a designated admin account who is not currently enrolled as a tutor, is enrolled as a student in the course. To do this, log in to any admin account. Navigate to Home, choose the appropriate Term, go to Students, scroll down, and click on the + Add button at the bottom of the page. In the Add New Student panel, search for the designated admin account and select the tutorial group and student group of the student who missed the deadline for the individual exercise. Click Save.

To create an empty Submission, impersonate the recently enrolled `admin-student` account. While impersonating, navigate to the appropriate Term and Exercise, click on the Submission tab (*not* Submissions in plural with a trailing s), and click on the button Create a Submission for .... The empty Submission has now been created, but is still assigned to the `admin-student`, and not yet to the correct student.

To remove the `admin-student` and assign the correct student, from the newly created empty Submission, click on the Edit button. Then, add the student who missed the deadline to the submission by clicking on the + Add Student button and searching for and selecting the corresponding student. Next, remove the `admin-student` from the submission by clicking on the red × button. To save the changes, click on the Submit button.

To perform the workaround for another student, change the tutorial group and student group of the `admin-student` accordingly. To do this, click the Students link in the right-hand panel. Locate the `admin-student` account in the list of enrolled students and click on the Show button. In the student overview panel of the `admin-student` account, click the Edit button. To assign the `admin-student` account to a different tutorial group and student group use the corresponding drop-down menus and click on the Save button to save the changes.

Finally, after creating empty submissions for all desired students, it is *very* important to remove any student role from the `admin-student` account. Otherwise, unwanted inconsistencies may be introduced. To remove the student role from the `admin-student` account, go to the student overview panel of the `admin-student` account, scroll down, and click on the red Delete Student button. The admin account itself is not deleted, but the role of being a student in a particular term of a course is removed.

6.2 Development Tutorials

Maintaining and extending Sapphire's software codebase with new features and fixing bugs are tasks that fall under the responsibilities of Sapphire admins. This section contains tutorials that document the technical procedures required for software development-related tasks.

6.2.1 Development Workflow

To ensure the correct implementation, testing, and deployment of changes, Sapphire admins follow a structured development workflow. The process starts with opening an issue on Sapphire's GitHub repository [Link and Kriechbaumer 2024a] for new features or bugs. Since Sapphire is a software project based on Ruby [Ruby 2023a] and Ruby on Rails [Rails 2023], the use of a Ruby-tailored IDE such as RubyMine [JetBrains 2024] is recommended.

RSpec [RSpec 2024] is a behavior-driven development testing tool for Ruby. RSpec tests are an important part of the Sapphire repository to validate the new functionality and ensure that the existing code remains intact. After implementing a feature or changing the code to fix a bug, the author of the changes is expected to write a corresponding RSpec which covers the changes to the codebase. Once the code has been tested locally, it is pushed to the Sapphire GitHub repository for review by other team members. This cycle continues until the changes are ready to be merged with the master branch. Once fully approved, the changes are merged with the master branch and deployed to the Sapphire staging environment (discussed in Section 6.2.5) for further manual interface testing. Only then, after successful interface testing, are the changes deployed to the production environment, completing the development workflow.

6.2.2 Configuring SSH Keys

This tutorial explains the process of setting up a Secure Shell (SSH) connection to the Graz University of Technology (TU Graz) servers hosting the staging and production instances of Sapphire and configuring the necessary SSH keys and admin rights on each Sapphire instance. A secure connection via the Virtual Private Network (VPN) of the TU Graz is required to be able to connect via SSH to the TU Graz servers. In the following commands, <user> should be replaced with the Sapphire admin's SSH login username, as configured by the server administrator.

1. Connect to the Sapphire instance via SSH with the server's admin user.

   To connect to staging Sapphire, the command is:

   ```
   ssh <user>@stagingsapphire.isds.tugraz.at
   ```

   To connect to production Sapphire, the command is:

   ```
   ssh <user>@sapphire.isds.tugraz.at
   ```

   If the SSH connection is successful the remote server's hostname and IP address are displayed in the terminal prompt. To execute commands inside the SSH terminal, use of sudo to gain superuser status might be required. The password required for sudo is the SSH login password configured by the server administrator.

2. Navigate to the home directory of the "sapphire" user.

   After establishing an SSH connection with the admin's login credentials to the server, navigate to the home directory of the "sapphire" user with the command:

   ```
   cd /home/sapphire
   ```

3. Create a backup of the SSH keys file.

   Create a backup copy of the currently authorized SSH keys file using the command:

   ```
   cp .ssh/authorized_keys .ssh/authorized_keys_copy
   ```

   These commands copy the content of the currently authorized SSH keys file `.ssh/authorized_keys` into a backup file `.ssh/authorized_keys_copy`.

4. Edit the SSH keys file.

   Open the SSH-authorized keys file for editing using any text editor (here nano is used):

   ```
   nano .ssh/authorized_keys
   ```

   Paste the public key of the user to be granted admin rights into the file `.ssh/authorized_keys`, then save, and close the editor.

   Note: If anything goes wrong during the editing process and there is a need to revert to the original authorized keys file, the backup copy can be restored using the command:

   ```
   cp .ssh/authorized_keys_copy .ssh/authorized_keys
   ```

   These commands copy the content of the backup file `.ssh/authorized_keys_copy` into the currently authorized SSH keys file `.ssh/authorized_keys`.

6.2.3 Granting Admin Rights for a Sapphire Account

If the user to be an admin already has an account on the Sapphire instance, that account can simply be granted admin rights. This is done by first finding the account using Active Record Querying [Rails 2024b]. In Ruby on Rails, Active Record Querying refers to retrieving and modifying database data using built-in methods provided by the Active Record ORM framework instead of writing raw SQL queries. Some common Active Record Querying methods include "find" to retrieve a record by its primary key and "where" to filter records based on specified conditions.

The SSH connection to Sapphire's server is only possible with a connection to the TU Graz VPN.

1. Connect again to the Sapphire instance via SSH, this time with the sapphire user.

   Connect to the staging Sapphire server with the command:

   ```
   ssh sapphire@stagingsapphire.isds.tugraz.at
   ```

or to the production Sapphire server with the command:

```
ssh sapphire@sapphire.isds.tugraz.at
```

2. Navigate to the current Sapphire directory.

   Navigate to the current Sapphire directory with the command:

   ```
   cd sapphire/current
   ```

3. Launch the Rails console.

   Launch the Rails console using the command:

   ```
   bundle exec rails c -e production
   ```

4. Look for the Sapphire account.

   To look for an account based on an email address, the following command can be used inside the Rails console, with `<user@tugraz.at>` being the email of the user to be made an admin:

   ```
   Account.where(email:"<user@tugraz.at>")
   ```

5. Grant admin rights.

   If an account with that email is found, the query returns the account ID, which can be used to find the account and update its admin attribute:

   ```
   Account.find(<ID>).update_attribute(:admin, true)
   ```

   In Ruby on Rails' Active Record, "update_attribute" is a method used to update a single attribute of a record in the database. Active Record queries typically return either the requested data or nil if no matching records are found. To ensure that an Active Record query was successful, check the returned value for nil or the presence of the expected data.

6. Or revoke admin rights.

   To revoke admin rights, the following query can be used:

   ```
   Account.find(<ID>).update_attribute(:admin, false)
   ```

6.2.4 Accessing and Interacting with Sapphire's Rails Console

This tutorial describes how to access and interact with Sapphire's Rails application server environment. The SSH connection to Sapphire's server is only possible with a connection to the TU Graz VPN. For the following commands, the admin could try using 'sudo' with the password provided by the server administrator if the commands do not work.

1. Connect to the Sapphire server via SSH.

   After establishing the VPN connection, the admin should use SSH to connect to the server using either:

   ```
   ssh sapphire@stagingsapphire.isds.tugraz.at
   ```

   to connect via SSH to Sapphire's staging Rails console, or:

   ```
   ssh sapphire@sapphire.isds.tugraz.at
   ```

   to connect via SSH to Sapphire's production Rails console.

2. Navigate to the current Sapphire application directory.

   Once connected via SSH, the admin needs to navigate to the current Sapphire application directory using the command:

```
cd sapphire/current
```

3. Run the Rails console.

   To interact with the Rails application, the admin should run the Rails console in production mode with the command:

   ```
   bundle exec rails c -e production
   ```

4. Interact with the Rails console.

   The Active Record Query Interface in Rails provides a way to interact with the database by constructing and executing SQL queries using intuitive and readable Ruby syntax.

   Once connected, the admin can query and modify ActiveRecord objects in the Rails console by leveraging the Active Record Query Interface. For example, the admin can find a submission using the query:

   ```
   Submission.where(exercise_id:xxx,student_group_id:xxx)
   ```

   replacing xxx with the appropriate IDs.

   ActiveRecord objects can also be modified through the Rails console. For example, the admin can delete a submission using the query:

   ```
   Submission.where(exercise_id:xxx,student_group_id:xxx).delete_all
   ```

   replacing xxx with the appropriate IDs.

6.2.5 Deploying Code to Staging and Production Environments

The production environment is where Sapphire operates live, hosting the finalized and tested version of the application. Changes here directly impact users and therefore must be well-considered! The production instance of Sapphire can be accessed through the link:

https://sapphire.isds.tugraz.at/

The staging environment mirrors the live platform, enabling testing of new features or updates before they are deployed to the live (production) environment. The staging Sapphire instance can be accessed through the link:

https://stagingsapphire.isds.tugraz.at

This tutorial describes how to deploy a specific revision of the Sapphire source code to both the staging and the production environment using Capistrano [Buck and Hambley 2024]. Capistrano is an open-source tool used for deploying Ruby on Rails web applications and executes scripts or commands in parallel on multiple servers. The following instructions assume that the reader has a basic understanding of the command line interface and has Capistrano installed and configured on their system.

To deploy a specific revision of the Sapphire source code to the staging environment using Capistrano the following command is used:

```
cap staging deploy REVISION=<COMMIT_ID>
```

This command deploys the code in the specified commit ID to the staging environment. By deploying to a staging environment first, it is possible to test the code and ensure that it is functioning as expected before deploying to the production environment.

Once the code has been tested in the staging environment, it can be deployed to the production environment. This is done using the command:

```
cap production deploy REVISION=<COMMIT_ID>
```

This command deploys the code in the specified commit ID to the production environment. It is important to note that deploying to the production environment should only be done after the code has been thoroughly tested, and any necessary adjustments have been made.

6.2.6 Managing Database Changes with Migrations

In Ruby on Rails, migrations are necessary when changes are made to the underlying structure of the Sapphire database. Existing data has to be migrated to the new structure. Migrations are used to create, modify, rename, or delete tables and columns in the database, as well as update indices, or create constraints. Migrations are written in Ruby code and stored in files located in the `db/migrate/` directory of a Rails application. Each migration file has a unique timestamp-based name and contains a set of instructions for applying or reversing changes to the database schema.

Knowing how to execute database migration is an essential skill for Sapphire developers, as it is often needed with new features to extend or modify the database structure.

A new migration file can be automatically created by executing the following command in the Sapphire repository terminal (with `<migration_name>` being the name of the migration chosen by the developer):

```
rails generate migration <migration_name>
```

This command will generate a new migration file with a timestamp prepended to its name, and place it in the `db/migrate` directory.

To specify the desired changes to the database, edit the migration file by opening it and editing the generated `change` method. For example, a migration file for creating a new table called students would look the following:

```
class CreateStudents < ActiveRecord::Migration[4.2]
  def change
    create_table :students do |t|
      t.references :tutorial_group
      t.references :submission_group
      t.string :forename
      t.string :surname
      t.integer :matriculum_number
      t.string :email
      t.datetime :registration_date

      t.timestamps null: false
    end
    add_index :students, :tutorial_group_id
    add_index :students, :submission_group_id
  end
end
```

To apply the migration changes to the database, execute the following command in the Sapphire repository terminal:

```
rails db:migrate
```

To undo a migration, the rollback command can be used to undo the last migration:

```
rails db:rollback
```

6.2.7 Extracting Sections from HTML Submissions

Some exercises require the submission of reports in HTML format. HTML `<section>` elements are used to structure the reports, and sections are assigned a unique `id` to identify them. Take the example of extracting a section called Evaluation Methodology from each submissions in Sapphire, so as to send it to a plagiarism detection service.

Step 1: Extract the Evaluation Methodology

To extract the Evaluation Methodology as HTML, follow the steps below.

1. Connect to production Sapphire via SSH.
   Connect to Sapphire with SSH using the command:
   ```
   ssh sapphire@sapphire.isds.tugraz.at
   ```
2. Navigate to Sapphire's current directory.
   Navigate to the Sapphire's current directory:
   ```
   cd sapphire/current
   ```
3. Copy over a previous `export_html_sections.rb` file.
   Navigate to a previous deploy folder from the last semester by running the following command:
   ```
   cd ~/sapphire/releases/<previous semester>
   ```
   Then, copy the `export_html_sections.rb` file into the current directory by running the following command:
   ```
   cp export_html_sections.rb ../../current/
   ```
4. Return to the `sapphire/current/` directory.
   Navigate back to the `sapphire/current/` directory.
5. Edit the `export_html_sections.rb` file.
   Edit the `export_html_sections.rb` file using the nano editor by running the following command:
   ```
   nano export_html_sections.rb
   ```
   Make the following changes:
   - `exercise_id`: Set the `exercise_id` (which can be read from the URL after clicking on the current exercise).
   - `term_name`: Choose a `term_name`.
   - `exercise_name`: Choose an `exercise_name`.
   - `section_css_selector`: Set the `section_css_selector` to the section id in Prof. Andrews's template. For example, the section id `section_css_selector` should be set to `section#secEvaluationMethodology` to extract the section with that `id`.
     ```
     <section id="secEvaluationMethodology" aria-labelledby="
         secEvaluationMethodologyHeading">
     ```
   - `filename_matcher`: Set the `filename_matcher` to the name of Prof. Andrews's template, which might be `heplan.html` in this case.

   Save the changes and exit the nano editor.

6. Perform the extraction.

   To extract the Evaluation Methodology section, execute the command:

   ```
   bundle exec rails runner export_html_sections.rb c -e production
   ```

   This will create plain text files for each plan or report containing the content of the Methodology section.

Step 2: Download the Extracted Results

After performing the extraction the next step would be to access and download the extracted results from the production server.

1. Navigate to the `exports/` directory.

   To access the extracted results, navigate to the following directory by running the following command:

   ```
   cd ~/exports/
   ```

2. Locate the current extraction export folder.

   Choose the appropriate date folder and locate the following files:

   - `.planextr`: contains the HTML of the extracted section.
   - `.planhtml`: contains the entire HTML submission.
   - `.-he-plantext`: contains the plain text of the extracted section.

3. Note the path and exit SSH.

   Make note of the current path by running the following command:

   ```
   pwd
   ```

   Then, exit the SSH connection by running the following command:

   ```
   exit
   ```

4. Download the export files from the production server.

   Download the export files from the server to a valid local path:

   ```
   scp -r sapphire@sapphire.isds.tugraz.at:<path-to-export> <local-path>
   ```

Step 3: Handle Failed Extractions

If an extraction fails, download the HTML file for that submission manually from Sapphire and convert it to plain text. To convert the HTML file to plain text, use the following command:

```
w3m -cols 76 -dump -o display_charset=UTF-8 <submission-file>
```

This will output the plaintext version of the HTML file to the terminal.

Copy the plain text output and paste it into a new text file using a text editor. Add the new plain text file to the tutor group for the corresponding group submission in the <local path for exports>. Also, add the HTML file for the corresponding group submission to the same tutor group.

Delete the folder containing all the `.planextr` files in `<localpathforexports>`. Once the methodology section has been manually extracted for any group submission that failed, proceed with zipping up all the folders containing the extracted methodology sections, including any manually extracted files and HTML files, and sending them to Prof. Andrews. Once the zip file has been created, send it to Prof. Andrews via email.

Chapter 7

Future Work

"Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."

[Antoine de Saint Exupéry]

This chapter describes some future work planned for Sapphire. For a comprehensive list of all planned features, bug fixes, and code changes, the reader is referred to the full issue list on the Sapphire GitHub repository [Link and Kriechbaumer 2024b].

7.1 Modify Sapphire Exception Notifier for Efficient Error Handling

One of the currently planned future improvements to Sapphire is to change the mailing logic of the Sapphire Exception Notifier. The Sapphire Exception Notifier sends out an email to all current administrators when errors occur in Sapphire's Rack/Rails application, in both the staging and production environments. However, in the current implementation, each error that happens triggers a separate email notification, which can lead to an overwhelming number of emails, especially during attempted cyber attacks. In such scenarios, the current Sapphire administrators can receive hundreds or even thousands of individual emails.

There are currently two planned solutions for limiting the number of emails sent to the administrators during cyberattack periods without losing important error alerts. One possible solution is to log each error to a file and send a single email every N (default = 2) hours after a certain number of errors. Another proposed solution is to combine multiple error notifications into a single email. This summary would be sent after a certain number of errors occur within a certain time frame.

7.2 Send Export Email Only to the Person Creating the Export

One of the relatively easy-to-implement planned improvements in Sapphire is to change the export email feature so that only the person creating the export receives an email notification once an export is complete. Currently, the export feature in Sapphire sends emails to all current administrators. Implementing this change would reduce spam emails to the administrators.

7.3 Implement Percentage-Based Individual Deductions for Group Exercises

In the current Sapphire implementation, tutors have the ability to deduct points from individual students for group work for various reasons, such as arriving late for group presentations. However, the existing system only supports point-based deductions. Experience with Sapphire has shown that a more flexible method of entering these individual deductions is required, e.g. the use of percentage deductions. To address this need, it is planned to implement a feature that will allow tutors to deduct points based on a percentage of the overall assignment grade. For example, if Student A is late and receives a 25% deduction, the tutor would enter a deduction of -25% in the Individual Deductions section. The system would then calculate and apply 25% of the total score for that assignment as an individual deduction.

Chapter 8

Concluding Remarks

"Great is the art of beginning, but greater is the art of ending."

[Henry Wadsworth Longfellow]

This thesis describes extensions and improvements made to Sapphire, an open-source, web-based platform for course management and grading [Link and Kriechbaumer 2024a] during the period from Aug 2022 to Jul 2024. One significant technical improvement was the implementation of file and folder renaming using the MVC approach. This involved changes to model, view, and controller files, as well as the configuration of routes, authorization logic, event logging, and RSpec tests. Another important new feature is the introduction of drag-and-drop functionality for moving files and folders. The architecture of this feature is explained in detail, including JavaScript logic, styling, controller logic, authorization policies, view changes, and routing configurations.

Furthermore, extensive technical documentation was created in the form of tutorials, which provide the knowledge needed for the maintenance and further development of Sapphire, ensuring continued maintenance and improvement of the platform.

The code contributions made in the scope of this thesis significantly enhance the usability, functionality, and efficiency of the platform for all its users, including students, tutors, lecturers, and administrators. The detailed tutorials serve as a resource for future Sapphire developers and administrators.

Bibliography

Andrews, Keith [2021]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. 03 Dec 2021. `https://ftp.isds.tugraz.at/pub/keith/thesis/` (cited on page xi).

Antoniou, Tassos [2023]. *Introduction to Ajax Calls*. 02 May 2023. `https://pressidium.com/blog/introduction-to-ajax-calls/` (cited on page 20).

Apple [2018]. *Model-View-Controller*. 06 Apr 2018. `https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html` (cited on page 9).

ASF [2024]. *The Apache HTTP Server Project*. Apache Software Foundation, 04 Apr 2024. `https://httpd.apache.org/` (cited on page 20).

Atlassian [2023]. *Trello*. 22 May 2023. `https://trello.com/` (cited on page 22).

Berners-Lee, Tim, Roy T. Fielding, and Larry Masinter [2005]. *Uniform Resource Identifier (URI): Generic Syntax*. Jan 2005. `https://tools.ietf.org/html/rfc3986` (cited on page 3).

Bishop, Mike [2022]. *HTTP/3*. Technical report. Internet Engineering Task Force, 27 Sep 2022. `https://datatracker.ietf.org/doc/rfc9114/` (cited on page 3).

Bootstrap [2023]. *Build Fast, Responsive Sites with Bootstrap*. 29 Jan 2023. `https://getbootstrap.com/` (cited on page 7).

Buck, Jamis and Lee Hambley [2024]. *A Remote Server Automation and Deployment Tool Written in Ruby*. Capistrano, 10 Mar 2024. `https://capistranorb.com/` (cited on page 48).

Bulma [2023]. *Bulma: The Modern CSS Framework That Just Works*. 29 Jan 2023. `https://bulma.io/` (cited on page 7).

Cerf, Vinton G. and Robert E. Kahn [1974]. *A Protocol for Packet Network Interconnection*. IEEE Transactions on Communications 22.5 (05 May 1974). doi:10.1109/TCOM.1974.1092259. `https://cs.princeton.edu/courses/archive/fall06/cos561/papers/cerf74.pdf` (cited on page 3).

Ecma [2024]. *Specifying JavaScript*. Ecma International, 23 Feb 2024. `https://tc39.es/` (cited on page 7).

Ellingwood, Justin [2020]. *An Introduction to DNS Terminology, Components, and Concepts*. DigitalOcean Tutorial, 04 Nov 2020. `https://digitalocean.com/community/tutorials/an-introduction-to-dns-terminology-components-and-concepts` (cited on page 4).

Ferrari, Luca and Enrico Pirozzi [2020]. *Learn PostgreSQL: Build and Manage High-Performance Database Solutions using PostgreSQL 12 and 13*. Packt, 09 Oct 2020. 650 pages. ISBN 183898528X (cited on page 10).

Fielding, Roy Thomas, James Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee [1999]. *Hypertext Transfer Protocol – HTTP/1.1*. Jun 1999. `https://tools.ietf.org/html/rfc2616` (cited on page 3).

Foreman, Steve [2017]. *The LMS Guidebook: Learning Management Systems Demystified*. Association for Talent Development, 28 Dec 2017. 274 pages. ISBN 1607281651 (cited on page 11).

Hinden, Bob and Dr. Steve E. Deering [1998]. *Internet Protocol, Version 6 (IPv6) Specification*. Technical report. Network Working Group, 01 Dec 1998. https://datatracker.ietf.org/doc/html/rfc2460 (cited on page 4).

ISOC [2023]. *Build, Promote, and Defend the Internet*. Internet Society, 21 Feb 2023. https://internetsociety.org/ (cited on page 3).

Iyengar, Jana and Martin Thomson [2022]. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Technical report. Internet Engineering Task Force, 19 Feb 2022. https://datatracker.ietf.org/doc/rfc9000/ (cited on page 3).

JetBrains [2024]. *RubyMine: The Ruby on Rails IDE by JetBrains*. 03 May 2024. https://jetbrains.com/ruby/ (cited on page 45).

Kriechbaumer, Thomas [2014]. *Sapphire Back-End: A Web-Based Course Grading Management System*. Bachelor's Thesis. Graz University of Technology, Austria, 28 Feb 2014. https://ftp.isds.tugraz.at/pub/theses/kriechbaumer-2014-bsc.pdf (cited on page 19).

Link, Matthias [2021]. *Sapphire Frontend: A Web-Based Course Grading Management System*. Master's Thesis. Graz University of Technology, Austria, 27 Oct 2021. 194 pages. https://ftp.isds.tugraz.at/pub/theses/mlink-2021-msc.pdf (cited on pages xi, 19–21).

Link, Matthias and Thomas Kriechbaumer [2024a]. *Sapphire – Course Management System*. Graz University of Technology, 21 Jan 2024. https://github.com/Sapphire-CM/Sapphire/ (cited on pages 1, 19, 45, 55).

Link, Matthias and Thomas Kriechbaumer [2024b]. *Sapphire – Course Management System*. Graz University of Technology, 08 Apr 2024. https://github.com/Sapphire-CM/Sapphire/issues (cited on page 53).

Microsoft [2022]. *Overview of ASP.NET Core MVC*. 27 Jun 2022. https://docs.microsoft.com/en-us/aspnet/core/mvc/ (cited on page 9).

Microsoft [2023a]. *SharePoint*. 30 Mar 2023. https://microsoft.com/microsoft-365/sharepoint (cited on page 11).

Microsoft [2023b]. *TypeScript*. 21 Feb 2023. https://typescriptlang.org/ (cited on page 7).

Moodle [2023]. *Moodle: Open-Source Learning Platform*. 30 Mar 2023. https://moodle.org/ (cited on page 12).

Nginx [2024]. *nginx*. 04 Apr 2024. https://nginx.org/ (cited on page 20).

Notodikromo, Adam [2020]. *Learn Rails 6: Accelerated Web Development with Ruby on Rails*. Apress, 25 Oct 2020. 560 pages. ISBN 1484260252 (cited on page 9).

OJSF [2023a]. *Event Loop, Timers, and process.nextTick()*. OpenJS Foundation, 22 Feb 2023. https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/ (cited on page 10).

OJSF [2023b]. *Node.js*. OpenJS Foundation, 22 Feb 2023. https://nodejs.org/ (cited on page 10).

Oracle [2023]. *MVC Architecture*. 22 Feb 2023. https://docs.oracle.com/en/java/java-components/overview/model-view-controller-pattern.html (cited on page 9).

Perham, Mike [2024]. *Sidekiq*. 10 Apr 2024. https://github.com/sidekiq/sidekiq (cited on page 20).

PGDG [2023]. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. PostgreSQL Global Development Group, 09 Feb 2023. https://postgresql.org/ (cited on pages 10, 19–20).

Phusion [2024]. *Passenger*. 22 Apr 2024. https://github.com/phusion/passenger (cited on page 20).

Pundit [2024]. *Pundit: Minimal Authorization through OO Design and Pure Ruby Classes*. Varvet, 17 Jun 2024. https://github.com/varvet/pundit/ (cited on page 35).

Rails [2023]. *Ruby on Rails*. Rails Contributors, 22 Feb 2023. https://rubyonrails.org/ (cited on pages 9, 19–20, 45).

Rails [2024a]. *Active Model Basics*. Rails Contributors, 26 Jun 2024. https://guides.rubyonrails.org/active_model_basics.html (cited on page 30).

Rails [2024b]. *Active Record Query Interface*. Rails Contributors, 03 May 2024. https://guides.rubyonrails.org/active_record_querying.html (cited on page 46).

Redis [2024]. *Redis*. 16 Apr 2024. https://github.com/redis/redis (cited on page 20).

RSpec [2024]. *RSpec: Behaviour Driven Development for Ruby*. 03 May 2024. https://rspec.info/ (cited on pages 36, 45).

Ruby [2023a]. *Ruby Programming Language*. Ruby Community, 21 Feb 2023. https://ruby-lang.org/ (cited on pages 8, 45).

Ruby [2023b]. *RubyGems.org*. Ruby Community, 21 Feb 2023. https://rubygems.org/ (cited on page 8).

Sunyaev, Ali [2020]. *Internet Computing*. Springer Cham, 13 Feb 2020. 413 pages. ISBN 303034956X. doi:https://doi.org/10.1007/978-3-030-34957-8 (cited on page 9).

Tailwind [2024]. *Tailwind CSS*. Tailwind Labs, 03 May 2024. https://tailwindcss.com/ (cited on page 7).

Tanenbaum, Andrew S., Nick Feamster, and David J. Wetherall [2020]. *Computer Networks*. 6th. Pearson Education, 24 Apr 2020. 960 pages. ISBN 0136764053 (cited on page 3).

Thomson, Martin and Cory Benfield [2022]. *HTTP/2*. Technical report. Internet Engineering Task Force, 01 Jun 2022. https://datatracker.ietf.org/doc/html/rfc9113 (cited on page 3).

Thoughtbot [2024]. *Thoughtbot/Factory_bot: A Library for Setting Up Ruby objects as Test Data*. 21 Jun 2024. https://github.com/thoughtbot/factory_bot (cited on page 37).

Turbolinks [2021]. *Turbolinks*. 25 Sep 2021. https://github.com/turbolinks/turbolinks (cited on page 19).

UC Berkeley [1999]. *The 3-Clause BSD License*. University of California, 1999. https://opensource.org/licenses/BSD-3-Clause (cited on page 10).

UIkit [2023]. *UIkit*. 29 Jan 2023. https://getuikit.com/ (cited on page 7).

USC [1981]. *RFC 791 - Internet Protocol*. Technical report. University of Southern California, 01 Sep 1981. https://datatracker.ietf.org/doc/html/rfc791 (cited on page 4).

W3C [2011]. *Scalable Vector Graphics (SVG) 1.1 Specification*. Technical report. World Wide Web Consortium, 16 Aug 2011. https://w3.org/TR/SVG11/ (cited on page 5).

W3C [2018]. *Scalable Vector Graphics (SVG) 2.0 Specification*. Technical report. World Wide Web Consortium, 04 Oct 2018. https://w3.org/TR/SVG2/ (cited on page 5).

W3C [2023]. *Cascading Style Sheets (CSS)*. World Wide Web Consortium, 18 Feb 2023. https://w3.org/Style/CSS/ (cited on page 7).

WHATWG [2022]. *HTML Living Standard*. Web Hypertext Application Technology Working Group, 2022. https://html.spec.whatwg.org/ (cited on page 5).

WHATWG [2023]. *Web Hypertext Application Technology Working Group*. 21 Feb 2023. `https://whatwg.org/` (cited on page 5).

Wikipedia [2023a]. *HTML Element*. 29 Jan 2023. `https://wikipedia.org/wiki/HTML_element` (cited on page 6).

Wikipedia [2023b]. *Model–View–Controller*. 29 Jan 2023. `https://wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller` (cited on page 9).

WP [2023]. *WordPress.com*. WordPress, 30 Mar 2023. `https://wordpress.com/` (cited on page 11).

ZURB [2023]. *Foundation Responsive Front-End Framework*. 18 Aug 2023. `https://get.foundation/` (cited on page 20).