JOHANNES KEPLER
UNIVERSITY LINZ

Author
Adrian Vinojčić, BSc
k11904250

Submission
Institute for System Software

Thesis Supervisor
o.Univ.-Prof. Dr. Dr. h.c.
Hanspeter Mössenböck

February 2025

A Tool for Creating Personalized Semester Schedules

Master Thesis
to obtain the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Inhaltsverzeichnis

Abstract

At Johannes Kepler University (JKU), thousands of students encounter difficulties in the process of planning their academic schedules for each new semester. This issue arises from the absence of university-provided planning tools, forcing students to manually search for courses on the university website and subsequently enter them into their personal calendars avoiding any time conflicts. This process is both time-consuming and prone to errors. To address this issue, we have developed a tool with an interactive user interface that lists available and not yet completed courses and allows students to efficiently create optimized schedules. This solution not only saves students countless hours but also helps them identify and resolve scheduling conflicts early. Consequently, the implementation of this tool ensures a more efficient academic planning experience for students.

Kurzfassung

An der Johannes Kepler Universität (JKU) haben Tausende von Studierenden jedes Semester Schwierigkeiten bei der Planung ihres akademischen Studienplans. Dieses Problem geht darauf zurück, dasses keine adäquaten universitären Planungstools gibt. Daurch sind die Studierenden gezwungen, Kurse manuell auf der Universitätswebsite zu suchen und diese anschließend in ihre persönlichen Kalender einzutragen, wobei Terminkonflikte zu vermeiden sind. Dieser Prozess ist sowohl zeitaufwändig als auch fehleranfällig. Um dieses Problem zu lösen, haben wir ein Tool mit einer interaktiven Benutzeroberfläche entwickelt, das verfügbare und noch nicht absolvierte Kurse auflistet und es den Studierenden ermöglicht, effizient optimierte Zeitpläne zu erstellen. Diese Lösung spart den Studierenden nicht nur unzählige Stunden, sondern hilft auch dabei, Konflikte im Zeitplan frühzeitig zu erkennen und zu beheben. Daher gewährleistet die Implementierung dieses Tools eine effizientere akademische Planungserfahrung für die Studierenden.

JOHANNES KEPLER
UNIVERSITÄT LINZ

o.Univ.-Prof. Dr.
Hanspeter Mössenböck
Institute for System Software

T +43 732 2468 4340
F +43 732 2468 4345
hanspeter.moessenboeck@jku.at

Secretary:
Karin Gusenbauer
Ext 4342
karin.gusenbauer@jku.at

Master's Thesis

A Tool for Creating Personalized Semester Schedules

Student:	Adrian Vinojcic (k11904250)
Advisor:	Prof. Hanspeter Mössenböck
Begin:	01.03.2024

Students usually plan the courses they want to complete in a semester by hand. They search for the desired courses using *Kusss* or *myJKU* and enter them manually in a weekly calendar. This often leads to time conflicts, especially if students are not studying according to the recommended course of study (e.g., due to work commitments), and requires rescheduling, which is tedious and error-prone.

The goal of this master's thesis is to develop a prototypical tool that can be used in a similar way as *myJKU* (on the web and on a mobile phone) and facilitates semester planning. The process and the functionality should be as follows:

- Students log in with their student ID and their password. This will bring up a list of all courses of their degree program, which are offered in the relevant semester and which they have not yet completed.
- Students select one or more courses from this list, which are then placed on a weekly schedule. If there are several groups for a course (e.g. for a lab), all groups are placed. Selected block courses are displayed outside the weekly schedule with their date and time. It should also be possible to remove each placed course from the weekly schedule.
- It should also be possible to place courses that do not appear in the list (e.g. courses from other degree programs) manually.
- If there are courses with overlapping dates, students can select one of them and click away the others. It should also be possible to select additional courses at any time and place them in the weekly schedule.
- A warning should be displayed if a lecture has been selected without the corresponding lab (or vice versa).
- It should be possible to use a filter with which students can specify on which days of the week and in which time range of a weekday they are free to attend courses. The list of available courses should then only show those for which at least one of the groups falls within this period. It should be possible to change the filter at any time, for example to list all courses that fall within a certain time range of a weekday that is still free. However, courses that have already been placed should not be removed from the weekly schedule.
- It should be possible to save the schedule locally so that the planning can be continued later.
- It should be possible to print out the schedule.

The front end of the tool should be implemented with *Angular* so that it is possible to integrate it into *myJKU* later. The backend is to be implemented in Java using *Spring Boot*.

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

As the tool accesses data in the IM's Oracle database and should be similar to *myJKU*, cooperation with the JKU's Information Management (IM) department is essential. If a *Rest* API already exists for certain requirements, this can be used. In addition to the supervisor of the master's thesis at the Institute for System Software, there will also be contact persons at the IM with whom the progress of the thesis should be regularly discussed.

The work's progress should be discussed with the advisor at least every 2 weeks. Please follow the guidelines of the Institute for System Software when preparing the written thesis. The deadline for submitting the written thesis is 28.02.2025.

1 Introduction

At JKU, the responsibility for planning one's schedule for the upcoming semester falls primarily upon the students themselves. This process entails careful coordination and attention to course offerings, ensuring that academic progress remains on track. Twice per year, at the beginning of February and mid-August, university lecturers publish a list of courses to be held in the forthcoming months via the university's web interface known as Kepler University Study Support System (KUSSS). This interface is designed to offer distinct functionalities tailored to the needs of instructors and students. Students can utilize this interface for various purposes, including registering for courses, viewing their grades, registering for exams, and accessing additional resources.

Currently, students are required to plan their semesters manually, by browsing through their desired courses in the KUSSS system or the myJKU portal (the latter will be detailed in in Section 2.2). Once a suitable course is identified, students typically record it in a calendar to keep track of their schedule. They repeat this process, carefully balancing their workload and ensuring that there are no conflicts in timing, until they are satisfied with their semester plan. The goal is to ensure that students have sufficient European Credit Transfer and Accumulation System (ECTS)-credits for the upcoming semester, and thus, they can make steady progress in their academic careers.

Semester planning is an essential process, given that the result determines the course of students' lives for the upcoming semester. Additionally, planning becomes more challenging when students deviate from the recommended study plan due to work commitments or similar duties. Furthermore, complex curricula require smart planning across multiple semesters, in order to finish the studies as quickly as possible. Consequently, this process is error-prone because places in many courses are limited and late registration is almost impossible for popular courses. Planning errors can result in students having to register for a course in a subsequent semester when it is offered again. Despite these challenges, the JKU has not yet implemented a tool to assist students in scheduling their courses in a guided manner, with the aim of reducing errors.

Two decades after an initial attempt to create a tool for planning semesters by Fröller [1], our application, the *Semesterplaner* was developed in collaboration with the Department of Information Management (IM) at JKU as a prototype specifically designed for computer science students. To ensure the proper functioning of the tool and to explore multiple approaches, two students were independently assigned the task of building separate versions of a semester planning application. While the backend is shared between both projects to provide consistent data to each frontend, the implementation of the frontends remains entirely distinct, with no code being shared. In regular meetings with our professor throughout the development process, we gained valuable insights from the other student's advancements, which further pushed our own ideas in return. An alternative implementation of the Semesterplaner is built and partly documented by Burghuber [2] as part of his bachelor thesis. It is recommended that readers review this additional work, as it offers perspectives and solutions that complement this project.

The primary purpose of the Semesterplaner is to provide an intuitive user interface (UI) that simplifies the planning process by presenting a comprehensive list of courses that the user (i. e. a student) has not yet completed. Once courses are selected, they are displayed in a calendar view, where any scheduling conflicts are immediately visible, ensuring that students can resolve overlapping time slots early in the planning process, reducing potential issues later on. The Semesterplaner incorporates all the essential tools required to search for courses within all curricula a student is enrolled. Users can filter courses based on criteria such as title, lecturer, time, and additional attributes, making it easier to identify suitable options. In addition, students designate can certain time ranges as unavailable, so that the system alerts them if courses conflict with their personal schedules or commitments.

In the following pages, a comprehensive documentation of the application will be presented. Chapter 3 presents a user guide of the Semesterplaner. Subsequently, a thorough investigation of essential implementation details will be performed in Chapter 4 and Chapter 5. Lastly, Chapter 6 presents the findings of a small-scale study conducted with a selected group of computer science students. Chapter 7 finally proposes potential enhancements to the Semesterplaner for future consideration.

2 Background

In this section, we will explore the process of planning a semester as a student at the JKU. We will also examine the role of myJKU, a platform for using JKU services by students. The final section will discuss the frameworks used for the Semesterplaner and the rationale behind their selection.

2.1 Planning a Semester

Semester planning at JKU consists of several steps. Initially, the university publishes the courses for the upcoming semester for inspection. After that, the students can obtain an overview of the courses and begin to plan their individual semester. From the publication of courses until the start of course registration, students have one to four weeks [3] to plan their schedule, depending on whether they are planning the summer or winter semester.

The planning phase involves multiple steps, starting with looking up a desired courses in KUSSS. If a course matches the schedule, it can be added. Otherwise, either the course cannot be added, or rescheduling has to be performed, to fit the course into the schedule if possible.

The tools for noting down a schedule range from a plain sheet of paper over simple text files on the computer to more elaborate spreadsheets. During the planning phase of a semester, it is important to maintain a record of crucial information, such as the date and time of scheduled courses and the total number of the ECTS credits to be earned in this semester. Furthermore, students must keep track of mandatory courses in their curriculum, to ensure timely completion of their studies. Additionally, students at the beginning of their studies are obliged to complete the Studies Introduction and Orientation

Phase (StEOP), which only allows them to take certain courses [4]. The StEOP has been implemented, to provide students with an overview of their program of study.

Following the completion of the plan, students must then proceed to the registration phase, during which they register for their desired courses in KUSSS. Given that places in some courses are limited, it is crucial to register for low-capacity and high-demand courses first.

2.2 myJKU and Shibboleth

The IM recently initiated the development of myJKU [5]. myJKU is a web application that provides similar functionality analogous to that of KUSSS, allowing students to look at their grades, their courses and much more. The IM's plan with myJKU is to incrementally provide more and more features from KUSSS in a modern and user-friendly format. Additionally, they want to provide features that are not present in KUSSS with the final goal of replacing KUSSS in its entirety [2, 6]. The Semesterplaner is intended to be part of myJKU in the future and is thus created in collaboration with IM.

Since myJKU needs to know information about the current user, the IM utilized the existing Shibboleth system [7] for the new platform as well. Shibboleth is an authentication and authorization process for web applications and web services. In fact, Shibboleth is one of the most widely used identity management systems having been adopted by academic institutions and other organizations [8]. We also use Shibboleth for the Semesterplaner as well to retrieve a user session to get more data from the KUSSS database.

2.3 Spring and Angular

The Semesterplaner has been developed using Java Spring [9] and Angular [10], which are widely regarded as leading frameworks for web development. The IM utilizes these technologies, and for a smoother integration into myJKU, we are leveraging the same frameworks for the Semesterplaner. Spring provides a simple way to define Application Programming Interface (API) endpoints over Representational State Transfer (REST). We utilize these endpoints to provide the frontend with data from the KUSSS database.

The frontend, which has been written in Angular, retrieves the data from the Spring backend when the student logs in. Angular is a framework dedicated to creating web frontends. In Angular developers can define components, which are reusable elements in HTML code. Developers can combine components to create even more powerful components or subpages. Furthermore, services which provide a common object between components can be created. In essence, Angular provides a way to modularize an application into smaller parts. Although Angular provides much more, the combination of components and services is enough for us to create the Semesterplaner.

3 User Guide

This section provides detailed instructions on how to use the Semesterplaner and its features. An overview of the application is provided in Section 3.1. The subsequent sections cover in-depth information about individual components and explain their usage.

3.1 Overview

The Semesterplaner is accessible via any web browser; however, it is only accessible after authentication and authorization via Shibboleth. Upon providing their credentials in the usual manner, students will gain full access to the Semesterplaner and will be presented with an initial empty view of the Semesterplaner as illustrated in Figure 3.1.

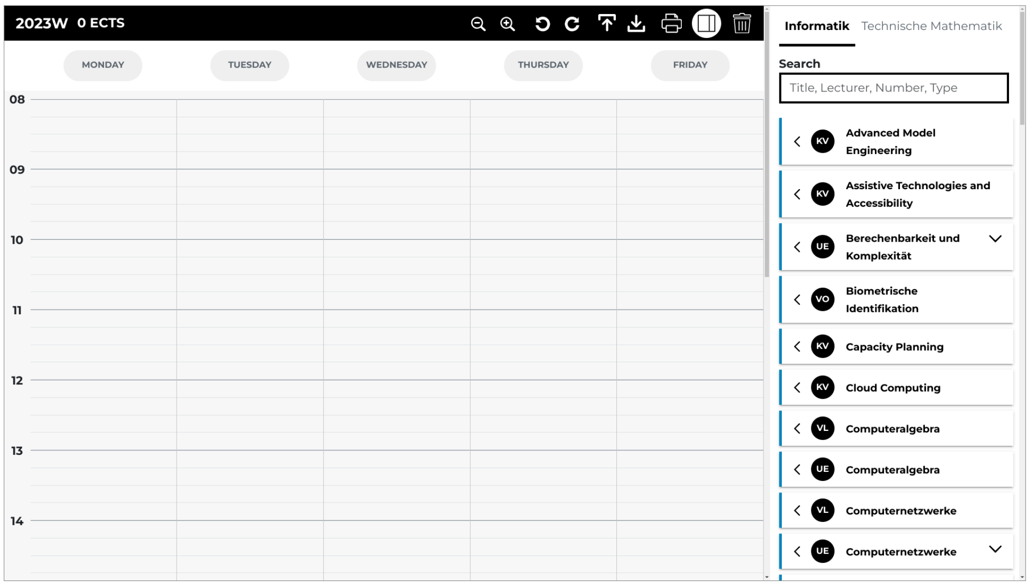


Figure 3.1: The Semesterplaner website after logging in.

The layout of the Semesterplaner closely resembles that of the calendar in myJKU [11]. The styling for the frontend of the Semesterplaner has been designed to match the look of myJKU as closely as possible.

On first inspection, the Semesterplaner can be described as being composed of the following three components:

- At the top, there is a black bar, which we will call *header* from now on. This component will be described in Section 3.2.
- On the right-hand side, there is a list of courses and a search field. This component will be referred to as the *course selection pane* and will be explained in Section 3.3.
- The big panel below the header is the *calendar*. In this component the main planning takes place. The calendar will be further elaborated in Section 3.4.

3.2 Header

After further examination of the header, we can see that it is divided into two sections. The left section is illustrated in Figure 3.2 and displays information relevant for the student. The section on the right contains important controls for the Semesterplaner and is explained in Figure 3.3.

In Figure 3.2, we can see the current semester ①. The semester changes twice a year on February 1st and August 1st. In the event that no courses have been found, the semester will fall back to the previous one. This mechanism ensures that the Semesterplaner can consistently display relevant courses. In addition, we can see an ECTS counter ②, which tracks the number of ECTS selected by the student for the current semester. These ECTS are accumulated from all selected courses, as well as from custom entries, the latter of which will be explained in Section 3.4.4.

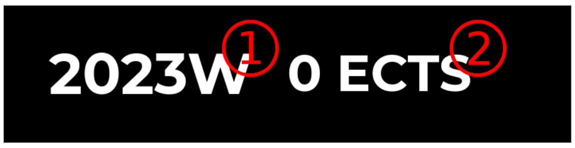


Figure 3.2: The left part of the header.

As previously stated, the right part of the header contains the controls for the Semesterplaner, which are essential for navigation and effective management of the interface. These controls are depicted in Figure 3.3 and include a button for

① zooming out. Zooming out can also be achieved by scrolling down while holding the `Ctrl` key.

② zooming in. Similarly, zooming in can also be achieved by scrolling up while pressing `Ctrl`, allowing for quick and precise adjustments in either direction.

③ undoing actions. This can also be achieved by pressing `Ctrl + Z`.

④ redoing actions. The shortcuts for redoing are `Ctrl + Y` and `Ctrl + Shift + Z`.

⑤ uploading a calendar. An existing calendar can be uploaded from a `.json`-file that has been downloaded before from the Semesterplaner. If the semester does not match the current semester, the file will not be imported. Furthermore, courses which are in the file, but not in the current course selection pane, will not be imported as well.

⑥ downloading the current calendar. When this button is pressed, the browser prompts the user to download the calendar to a `.json`-file. This file will contain the semester, as well as everything that was specified in the calendar view.

⑦ printing the calendar in portable document format (PDF). The Semesterplaner opens the generated PDF in a new tab and includes the calendar view, as well as a list of all selected block courses.

⑧ toggling the visibility of the course selection pane, allowing users to have a bigger calendar view if the course selection pane is not required.

⑨ clearing the calendar. Pressing the clear button brings up a confirmation window if there is something in the calendar view, allowing the user to confirm their intention. Otherwise, the button does nothing and the interface remains unchanged.

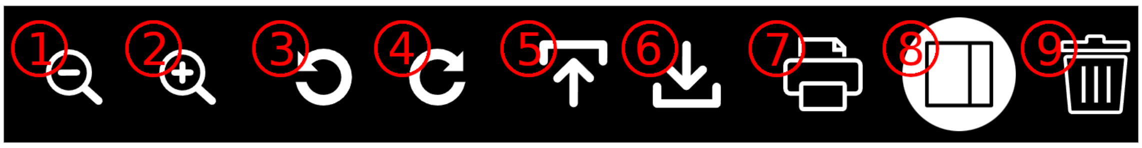
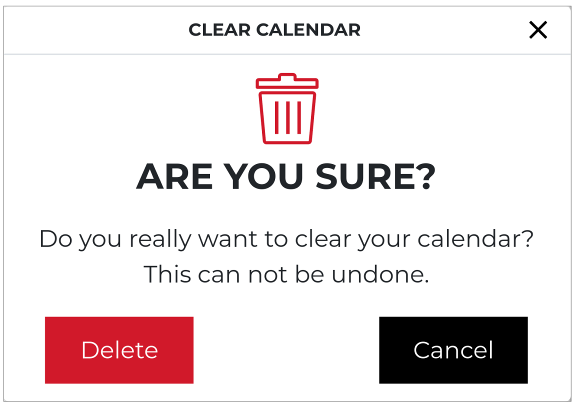


Figure 3.3: The left part of the header.

Upon pressing the clear button, the Semesterplaner prompts the user with the dialog depicted in Figure 3.4. It displays a warning to the user regarding the current action and provides the user with two options. The first option is to click on the red delete button, which permanently clears everything from the calendar. Note that the undo button cannot reverse this action later. The cancel button and the cross in the upper right corner close the dialog. Additionally, it is also possible to close the dialog by clicking outside of it.


CLEAR CALENDAR

ARE YOU SURE?

Do you really want to clear your calendar?
This can not be undone.

Figure 3.4: The delete dialog.

3.3 Course Selection Pane

The course selection component is mainly composed of a course list which is supplemented by a search field and an optional study selection, as illustrated in Figure 3.5. The study selection only becomes visible at the very top, when a student is enrolled in multiple studies. Students are at liberty to switch between studies by clicking the tab with the desired study whenever they wish. Furthermore, students can construct their schedule by selecting courses from any study in which they are enrolled.

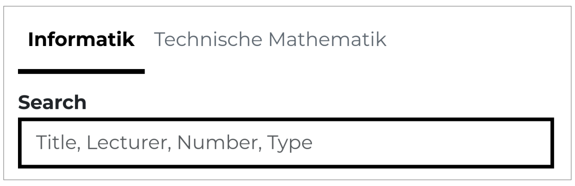


Figure 3.5: The study selection and the search field of the selection pane.

The search component, illustrated in Figure 3.5, filters the courses in the list. Students can search by (1) title, (2) subtitle, (3) course type, (4) lecturer name and (5) course number. If the course number or lecturer matches any course in a *course class* (i. e., a group of courses that belong to the same category and passing one of them is sufficient to complete subject), the entire course class will appear in the list. When students switch their study in the study tab, the Semesterplaner does not remove the filter but rather reapplies the filter to the other study.

A subset of the selection pane is shown in Figure 3.6. The figure reveals three elements that represent the courses for the *Digital Circuits* lecture and lab. Depending on the number of courses that belong to a course class, the Semesterplaner displays the course classes differently. When a course class consists only of one course, the course selection pane displays the minimal required information consisting of the course type ①, the course title ②, an optional subtitle ③, and an insert button ④ to add the course to the calendar.

The panel of a course class with multiple courses includes the same elements. However, the insert button ④ adds all courses of the course class to the calendar. Furthermore, element ⑤ is an icon that informs the student that this course class has multiple courses, and is expandable. When the student clicks on such a course class, it expands in order to show the courses that are initially hidden. Each expanded course class consists of courses ⑥, each of which has a button ⑦ to add it to the calendar, a field for the lecturer's name ⑧, and a field for the time spans of this course ⑨.

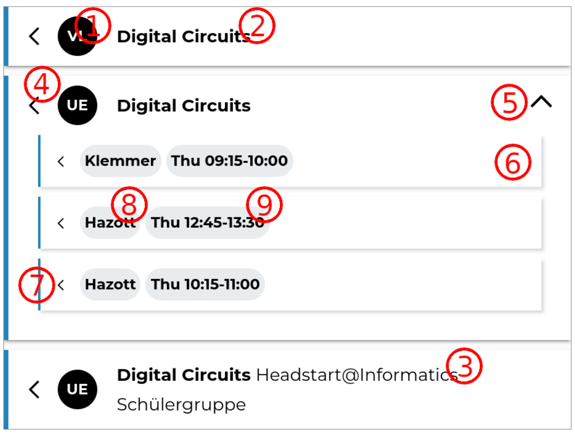


Figure 3.6: The Digital Circuits lecture and courses for the corresponding lab.

Moreover, all courses have a dialog associated with them, which displays more detailed information about the course. The dialog is explained in Section 3.5 and can be accessed either by clicking on the course class if no icon indicates expandability, or by expanding the course class and clicking on a course to enable the dialog view of the course.

3.4 Calendar

The calendar component is the most significant view, as this is where the majority of the actions are performed. Initially, it consists of the weekdays from Monday to Friday, with a time span from 8:00 a.m. to midnight. As explained in Section 3.3, the student can insert courses into the calendar. When the student selects a course that takes place on Saturday or Sunday, the Semesterplaner expands its calendar component to include all days until the corresponding day.

3.4.1 Courses

Among other entry types, the calendar primarily consists of courses, which are added when the student selects a course from the selection. Examples of such courses are shown in Figure 3.7 and Figure 3.8. At the top, the course type ① is displayed, followed by the title and an optional subtitle ②, similar to the course selection pane. If a course class contains multiple courses, the lecturer's name ③ is displayed as well. A small cross icon ④ is positioned in the upper right corner. By clicking this icon, the course can be removed from the calendar and returned to the course selection pane.

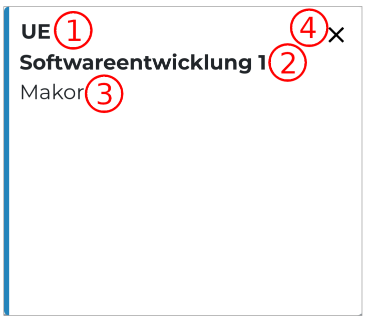


Figure 3.7: The Software Development lab with the lecturer Makor.

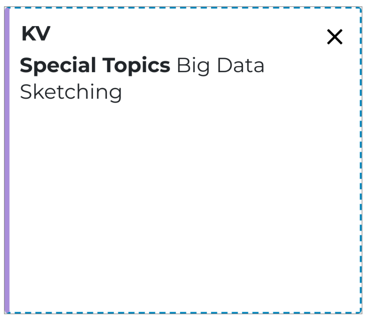


Figure 3.8: The Special Topics - Big Data Sketching course.

Courses have a blue or purple colored bar on the left-hand side. When the bar is purple, as depicted in Figure 3.8, it signifies that this course is the one added most recently. Conversely, if the bar is blue, it only distinguishes courses from custom entries, which will be explained in Section 3.4.4. Analogous to the course selection pane, a click on the course in the calendar opens a dialog for the course which will be addressed in Section 3.5.

Additionally, a dashed border surrounds the course in Figure 3.8, serving as an additional visual indicator. In the absence of a border, the course is designated for weekly sessions. When the border is dashed, it indicates that the course has biweekly sessions. Furthermore, a blue tint signifies that the course is a block course.

3.4.2 Block Courses

Block courses are characterized by their irregular sessions. For that reason, the courses are not placed in the calendar directly when added. Instead, the Semesterplaner places them in a separate pane below the calendar with the option to add them to the calendar. The block course pane is illustrated in Figure 3.9 and is only displayed when the student adds at least one block course from the course selection pane.

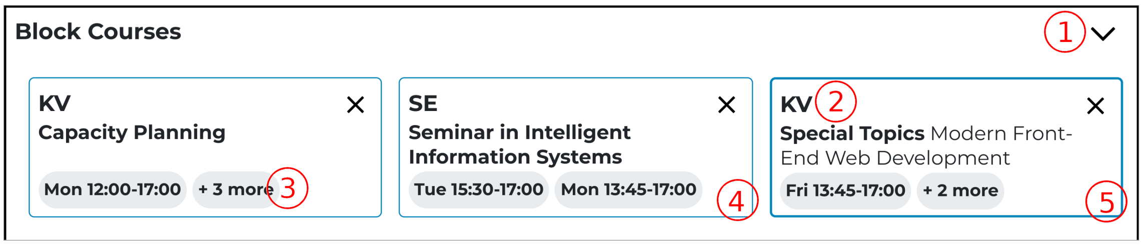


Figure 3.9: A few added block courses.

In the top right corner of the block courses pane there is a button ① to minimize the pane. The body of the pane contains a list of all added block courses. Analogous to the panes in the calendar, the type, title, and subtitle ② are displayed. Additionally, a course shows its sessions as well ③. Depending on the quantity, either all sessions are shown, or if there are more than two, the panel shows *+n more*.

Courses from the block course pane can be shown in the calendar using a dialog that is accessible by clicking on the course. The courses have a thin border ④ when they are not shown in the calendar; in contrast, they have a thick border ⑤ when displayed.

3.4.3 Specifying Time Ranges

In order to provide more advanced features for the Semesterplaner, it is important to be able to specify a certain time range in the calendar (e. g., for marking time ranges on which the student is not available). Time ranges can be specified using drag selection. There are two options for doing that.

The more common option is to press the mouse button at the desired start time, drag the mouse to the desired end time, and release the mouse button, or vice versa. This method of drag selection, allows the user to specify time ranges with an accuracy of 15 minute intervals. If the student makes a mistake with the selection, they can click and drag at the top or bottom to adjust the selection to the correct times. Pressing on one day and dragging to the next day will not select both days. Rather, the Semesterplaner keeps track of the cursor position and selects the time ranges where the press started.

The second option is to select an entire day by clicking on the weekday at the top of the calendar. If a day is already selected, clicking on the weekday will deselect that day. Students can select multiple time ranges by holding down `Ctrl` while dragging. To remove time ranges or parts of them, pressing `Shift` while dragging is required.

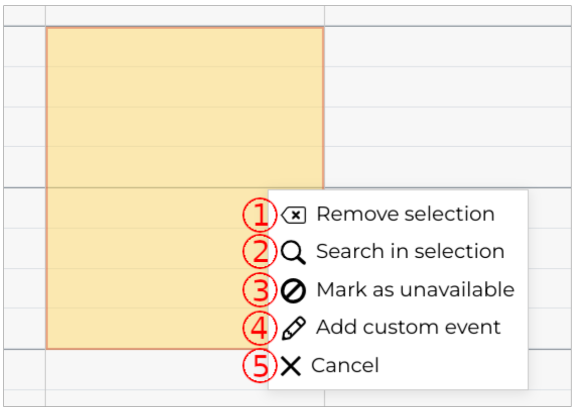


Figure 3.10: Drag selection with follow-up options.

Once students are satisfied with their selection, they may click on it. At this point, the Semesterplaner will provide available options in a context menu, as depicted in Figure 3.10. The options include the ability to remove the selected time range ①. This can also be achieved by right-clicking outside of the selection or by pressing the `Esc` key when no context menu is active. Furthermore, students have the option to search for courses within the specified selection ②, as outlined in Section 3.4.6, and to mark the selection as a time range where the student is unavailable ③, as detailed in Section 3.4.2. Moreover, the actions include the ability to add a custom event ④, as described in Section 3.4.4. The final available action is to cancel ⑤, which only removes the context menu. Note that canceling can also be achieved by clicking anywhere outside of the context menu.

3.4.4 Custom Events

When selecting "*Add custom event*" in the context menu, the student may add an entry with a green bar that appears analogous to a course, as depicted in Figure 3.11. The figure illustrates the custom event at three different stages.

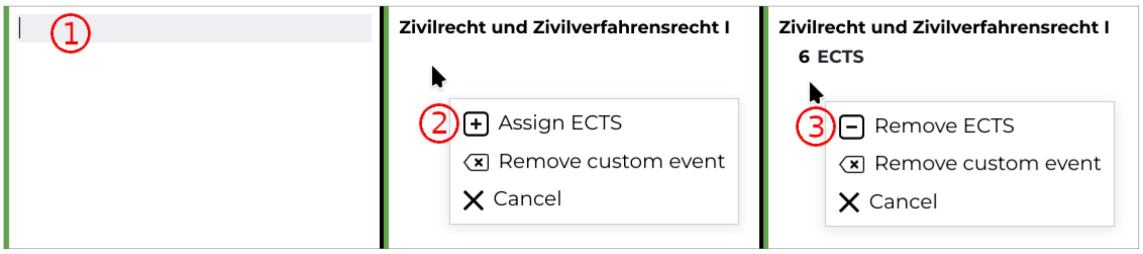


Figure 3.11: Custom event and its options while adding a title and ECTS.

Initially, the entry is blank, as illustrated in the leftmost example. Upon adding a custom event, the student is immediately prompted to enter a title ①. In the case of an error, the text can be edited at any time by clicking on the title. Similar to the drag selection, custom entries have a context menu assigned to them when clicked. As illustrated in the second two examples in Figure 3.11, the context menu can be observed in two different states.

The first available action is to assign ② or remove ③ ECTS to the custom event. This feature is provided to mimic courses from alternative curricula. When ECTS are assigned to a custom event, they are also added to the total ECTS in the ECTS counter in the header. The ECTS for a custom event can be edited analogous to the title.

The second entry in the context menu removes the custom event. If there are multiple custom events, the context menu contains an additional option to remove all custom events. The last option removes the context menu, similarly to the context menu for the selection of time ranges.

3.4.5 Time Restrictions

Another option to choose is "*Mark as unavailable*" for a selected time range. When the student chooses this option, the selected time range will be marked as depicted in Figure 3.12. Furthermore, all courses that overlap the unavailable time range will be marked as unavailable in the course selection pane on the right-hand side, as shown in Figure 3.13.

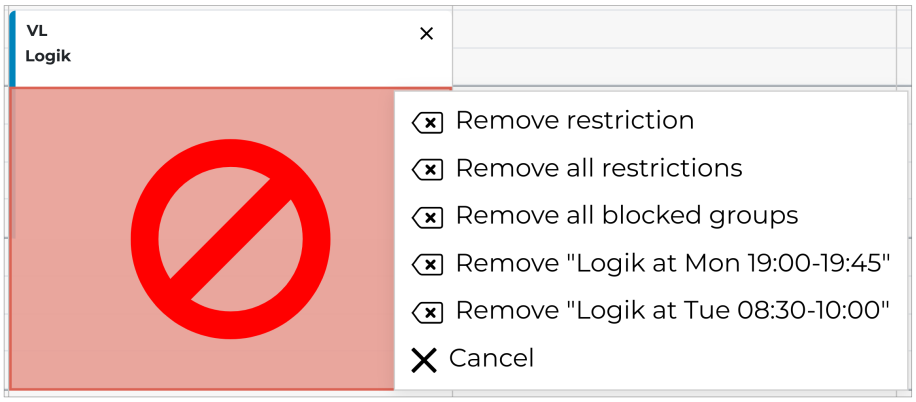

VL
Logik
Remove restriction
Remove all restrictions
Remove all blocked groups
Remove "Logik at Mon 19:00-19:45"
Remove "Logik at Tue 08:30-10:00"
Cancel

Figure 3.12: Time range marked as unavailable and options thereof.

A closer examination of Figure 3.12, reveals that one course is overlapping with the displayed restriction. In this example, we assume that there are multiple restrictions and courses in the Semesterplaner. When clicking on a restriction, a context menu similar to those previously mentioned is displayed.

The first option is to remove the restriction. In this example, the option of removing all restrictions is available as well, because we have assumed that there are multiple restrictions. Depending on the presence of multiple courses in the calendar, there is an option to remove all blocked courses. Additionally, there is an option to remove each course individually. The final option is to cancel the action and close the dialog.

Figure 3.13 shows an example with two (partly) restricted course classes in the selection pane. The first course class has three courses, two of which are restricted. The Semesterplaner replaces the insertion button with a red restriction icon and shades the panels in a light red hue. The utility of the buttons is the same as previously described, while the restriction icons cannot be clicked. If all courses within a course class are restricted, the entire course class is marked as unavailable. When students wish to insert a course anyway, they can do so by opening the dialog and proceeding as explained in Section 3.5.

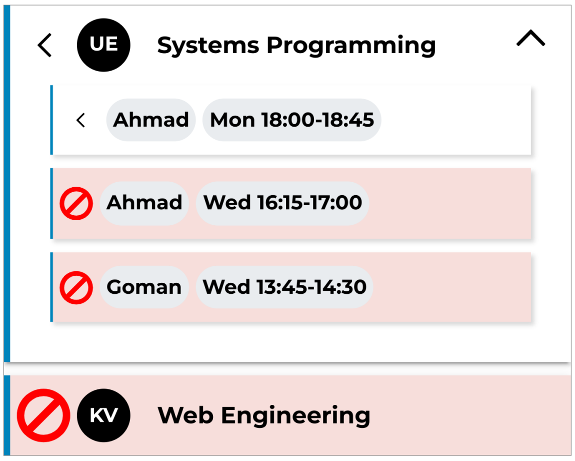


Figure 3.13: Two courses that are (partly) restricted.

3.4.6 Searching for Courses in Selected Time Ranges

Another option for the student is to click "*Search in selection*" in the context menu of the selected time range. When the student chooses this option, the Semesterplaner displays the time range in a blue color. Furthermore, it displays a new element in the course selection pane. At the top, below the search bar, new elements appear that represent every defined time range as shown in Figure 3.14. These elements offer a concise overview of the defined time ranges, allowing students to see their time ranges at one glance.

If a time range has been defined for searching courses within it, the Semesterplaner displays in the selection pane only those courses that fully fit within a defined time range. Courses that partially overlap a time-search element are excluded from the course selection pane as well. As with all other calendar elements, the student may click on time-search elements to get options in the context menu. For these, the options are to hide the context menu and to remove the element which can also be achieved by clicking on the cross, which is positioned in the element beneath the search bar.

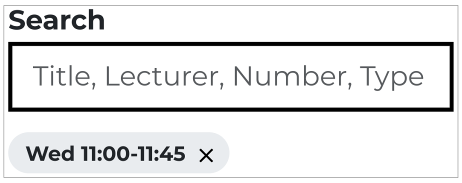
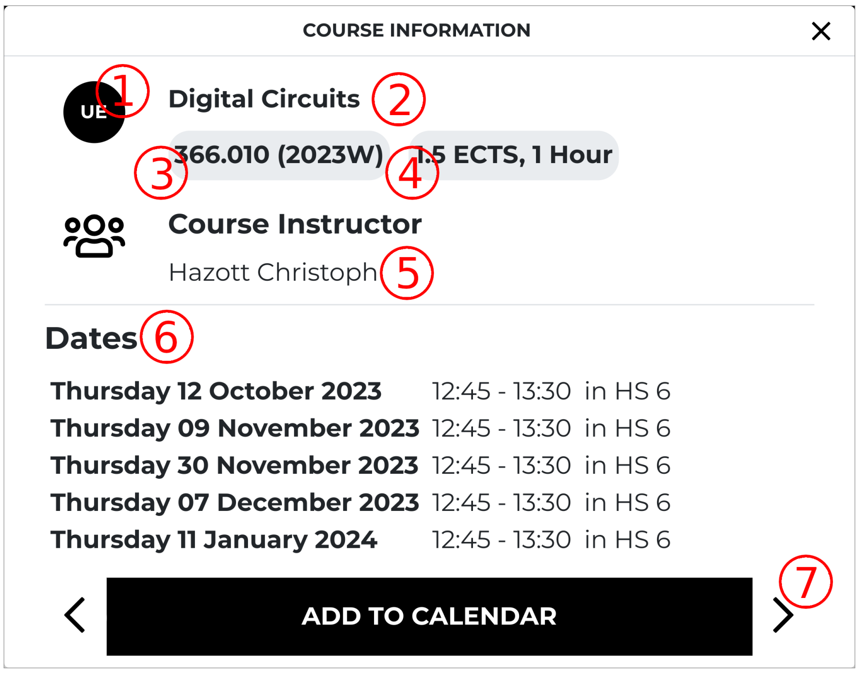


Figure 3.14: A time chip below the selections search bar.

3.5 Course Dialog

Given the limited capacity of the screen space, it is necessary to maintain the smallest possible dimensions for elements and thereby display only critical information in the course selection pane and in the calendar. It is important that the student immediately knows what a specific course is about and when it takes place. In order to display all information of a course, we chose to implement a dialog view for each course as illustrated in Figure 3.15. A dialog is an element that pops up above everything else. This approach allows presentation of all data without interrupting the main workflow and overwhelming the student with excessive information. A student has multiple ways to open a course dialog, that is, by clicking on

- a course class in the course selection pane if the course class consists of one course.
- a course in the course selection pane if the course class consists of multiple courses.
- a course in the calendar.
- a course in the block course pane.


COURSE INFORMATION

UE

Digital Circuits

366.010 (2023W)

1.5 ECTS, 1 Hour

Course Instructor

Hazott Christoph

Dates

Thursday 12 October 2023	12:45 - 13:30 in HS 6
Thursday 09 November 2023	12:45 - 13:30 in HS 6
Thursday 30 November 2023	12:45 - 13:30 in HS 6
Thursday 07 December 2023	12:45 - 13:30 in HS 6
Thursday 11 January 2024	12:45 - 13:30 in HS 6

Figure 3.15: The dialog for the course "Digital Circuits".

The dialog contains all available information related to a course, including the course type ① and the course title ② as well as the optional subtitle as discussed before. Furthermore, the course number and semester ③ as well as the ECTS and weekly hours ④ are displayed. Below that, the instructors for this course ⑤ are listed, followed by a list of all sessions for the selected course in the form of a table ⑥. Each session is characterized by date, start time, end time, and location. Moreover, there are always buttons ⑦ to perform actions related to the course, depending on the current state. The buttons can occur in different varieties, which are illustrated in Figure 3.16.

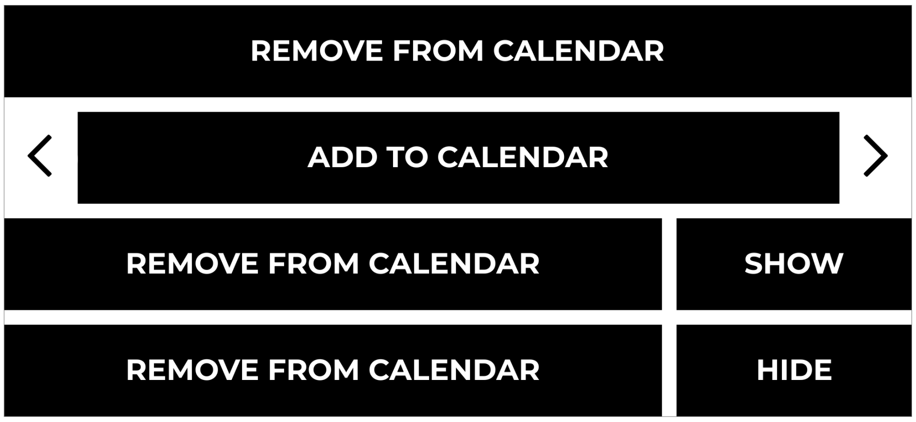
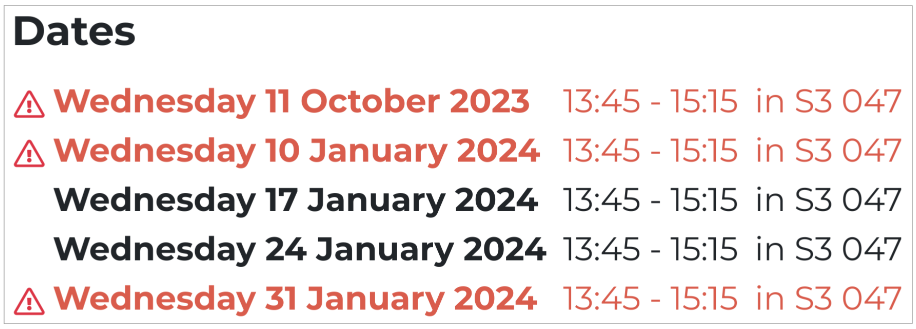


Figure 3.16: The variety of buttons in the course dialog.

Normal courses have just one button with the text "*add to calendar*" or "*remove from calendar*", depending on whether the course is visible in the course selection pane or in the calendar. When the "add to calendar" button is pressed, the Semesterplaner moves the course from the course selection pane to the calendar or vice versa. The button is updated to perform the inverse action when clicked and the dialog does not close. Optionally, there are two additional buttons to the left and right of the primary button when there are multiple courses belonging to a particular course class. These buttons can be used to cycle through the courses of the course class, compare their times, and identify the most suitable course. Furthermore, block courses feature an additional button, labeled "*show*" or "*hide*". This button enables the student to toggle the visibility of the block course in the calendar, as already mentioned in Section 3.4.2. This button is only displayed when the course is already in the calendar. When the current course is in the course selection pane, this button remains hidden until the course is added to the calendar.

3.6 Warnings

The Semesterplaner prevents a substantial amount of errors, as it ensures that students cannot misplace courses in the calendar any more. However, due to design choices, the Semesterplaner only displays the most common times of a course in the calendar. Consequently, there is a possibility that a course overlaps with some other course at a specific time, which is not reflected in the calendar. To address this, the Semesterplaner alerts the student of any overlap in the course dialog. Each session with an overlap is complemented by a small warning icon and is colored red, as illustrated in Figure 3.17.



Dates

⚠ Wednesday 11 October 2023 13:45 - 15:15 in S3 047
⚠ Wednesday 10 January 2024 13:45 - 15:15 in S3 047
Wednesday 17 January 2024 13:45 - 15:15 in S3 047
Wednesday 24 January 2024 13:45 - 15:15 in S3 047
⚠ Wednesday 31 January 2024 13:45 - 15:15 in S3 047

Figure 3.17: Overlaps visualized in the course dialog.

Another potential error is when a student forgets to add a lab to the corresponding lecture. In such a case, the student will observe two warnings, as depicted in Figure 3.18. The course in the calendar and the missing courses in the course selection pane are highlighted in orange together with a warning icon.

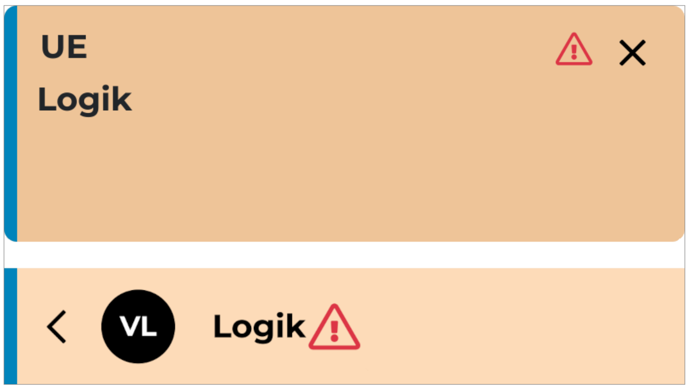


Figure 3.18: Paired courses highlighted if not taken together.

4 Architecture

In this section, the architecture of the Semesterplaner is explained (see Figure 4.1). The Semesterplaner has been constructed following the standard client/server architecture model, with the frontend and backend clearly separated. This architectural decision is aligned with the IM's choice to facilitate seamless integration into their existing web application myJKU.

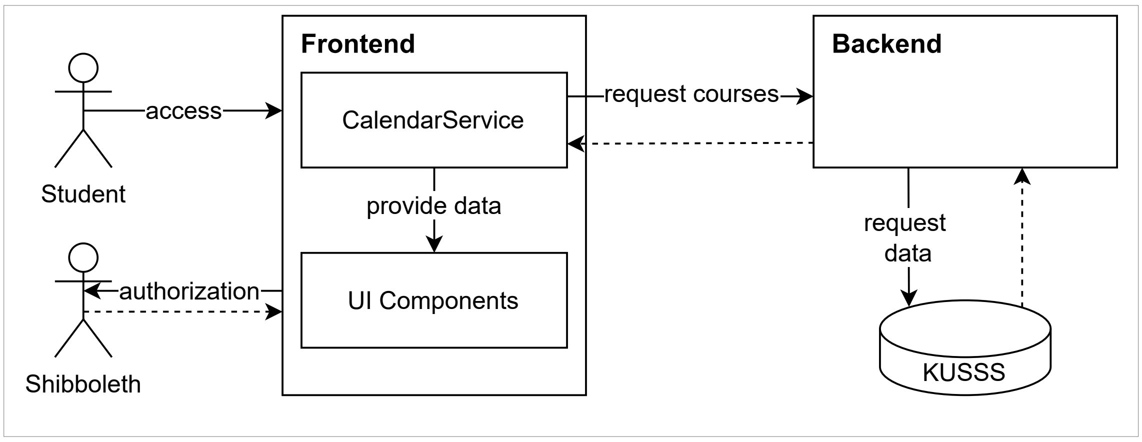


Figure 4.1: The architecture of the Semesterplaner.

In order to utilize the Semesterplaner, students access it via a web browser. The frontend, which serves as the user interface, consists primarily of the calendar service and UI components. The calendar service is responsible for preparing and organizing the data, ensuring that it is structured and ready for display. In addition, the calendar service interacts with the backend to request data about the studies and courses associated with the current user. The UI components present this data to the user in a visually accessible format. The backend is responsible for fetching data from the KUSSS university database and preparing it for presentation in the frontend. To ensure proper access and authentication, the Semesterplaner utilizes the university-provided Shibboleth system.

4.1 Backend Architecture

The backend of the system is a Java Spring application that processes requests from the front end, as illustrated by the dashed arrow in Figure 4.2. The frontend can call the API endpoints provided by the `CourseController` to retrieve studies and courses for a specific student. Subsequently, the `CourseController` requests the necessary data from the `CourseRepository` and forwards it to the frontend.

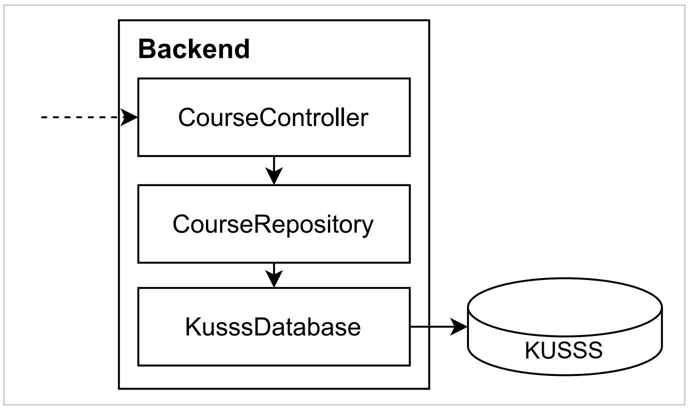


Figure 4.2: The detailed architecture of the backend.

The `CourseRepository` checks if the courses of the requested study are already in its cache. If so, it filters these courses, removing those, that have already been completed by the student or these that got accredited. Otherwise, the `CourseRepository` consults the `KusssDatabase`. As soon as the `CourseRepository` receives the courses of the study, it caches them for potential future use, and proceeds as explained above. The class `KusssDatabase` communicates with the the actual KUSSS database and returns the raw data in the form of a record class. The raw data from the `KusssDatabase` is put into a more suitable form by the `CourseRepository`.

4.2 Frontend Architecture

Upon successful authentication and authorization by Shibboleth, students are granted access to the frontend of the Semesterplaner. Specifically, they see the UI components populated with data from the backend. The view gets the data from the `CalendarService`.

For this, the `CalendarService` builds a data model and provides it to each component as needed. The student can interact with the UI components and these communicate with the `CalendarService` to perform actions on the data model. Furthermore, the `CalendarService` notifies the `HistoryService` of any modifications made by the student, thereby ensuring the preservation of the most recent changes, so that the student can undo and redo changes on demand.

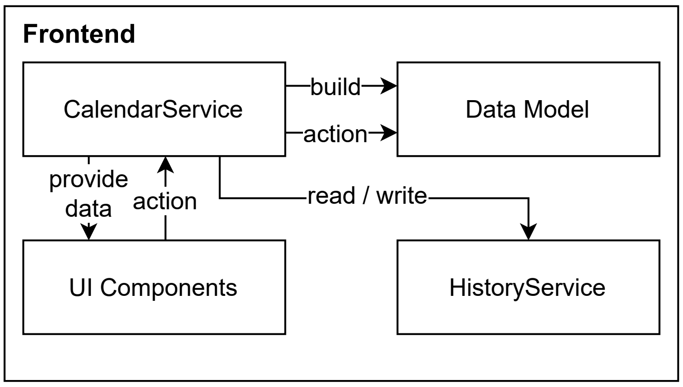


Figure 4.3: The detailed architecture of the backend.

5 Implementation

After having presented an architectural overview, we will go through implementation details of the Semesterplaner. In Section 5.1 we will see how the data from the database is transformed into a more suitable form and Section 5.2 covers the remaining implementation of the backend. The following sections explain the different parts of the frontend.

5.1 Data Preparation

The IM provided us with three record types that the Semesterplaner can access. The first two records (see Listing 5.1 and Listing 5.2), can be queried by matriculation number. The first one denotes a study that the student is taking (incl. matriculation number). The second one returns data that is relevant to identify completed and accredited courses (excl. matriculation number), which are stored in the same table.

```
TYPE Studium_row is RECORD(  // STUDY
    stmat      VARCHAR2(8),   // the students matriculation number
    skzs       VARCHAR2(11),  // study code
    spkkeys    VARCHAR2(7),   // curriculum code
    bez        VARCHAR2(150), // study title
    status     VARCHAR2(15),  // status (finished, ongoing, paused, ...)
    bevorzugt  CHAR(1));      // preferred flag
```

Listing 5.1: Studies of a particular student as received from the KUSSS database.

```
TYPE LVAabsolv_row is RECORD( // COMPLETED
    lvsem   CHAR(5),           // semester of completion
    lvanr   CHAR(6),           // course number
    klaid   VARCHAR2(16),      // course class
    epdat   DATE);             // date of completion
```

Listing 5.2: Completed course of a particular student as received from the KUSSS database.

Furthermore, the record type in Listing 5.3 contains information about sessions of a course. We query these records by `spkkey` (i.e., curriculum code) and `semester` and receive results for the entire study. The `spkkey` can be obtained by querying the studies of a student.

```
TYPE LVATermine_row is RECORD(   // SESSION
    lvsem          CHAR(5),       // semester
    klaid          VARCHAR2(16),  // course class
    lvanr          CHAR(6),       // course number
    lvatyp         CHAR(2),       // course type
    titel          VARCHAR2(200), // course title
    untertitel     VARCHAR2(200), // subtitle
    wochenstunden  NUMBER(4,2) ,  // weekly hours
    ects           NUMBER(5,2),   // ects
    inhaltsgleich  CHAR(1),       // content identical
    rhythmus       VARCHAR2(20),  // rhythm / type of dates
    leiter         VARCHAR2(50),  // course lecturer
    datum          DATE,          // session date
    startzeit      CHAR(5),       // session start time
    endzeit        CHAR(5),       // session end time
    raum           VARCHAR(8));   // session room
```

Listing 5.3: Session of a course as recieved from the KUSSS database.

The `KusssDatabase` transforms all records into Java classes as illustrated in Figure 5.1. The records denoting the studies of a student are transformed into a `List<Study>` and the records denoting the courses of a student are converted into a `List<CourseClass>`.

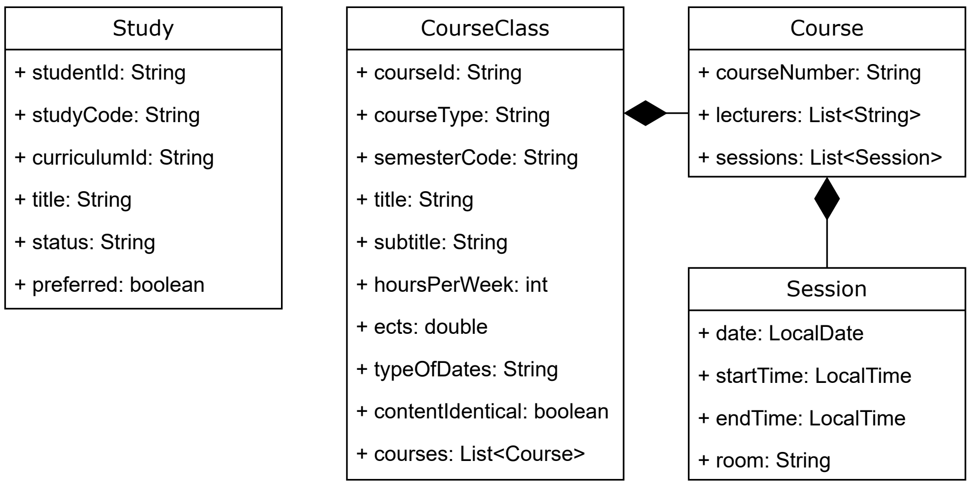


Figure 5.1: The UML diagram of the data transformed by the `KusssDatabase` class.

5.2 Data Processing and API Design

After transforming the data into Java classes, the rest of the backend can work with those four classes. As explained in Section 4.1, the `CourseController` provides an API and forwards the parameters to the `CourseRepository`. At the moment, the Semesterplaner needs two API endpoints in its backend. One returns the studies of a student, while the other returns the filtered courses of a study. A shortened example for an endpoint is `/semplan/studies` that can be seen in Listing 5.4. Since the API endpoints have a similar structure, we will explain the concept on this example.

```
1  @GetMapping("/semplan/studies")
2  ResponseEntity<List<Study>> getStudies(
3      @RequestHeader Map<String, String> headers) {
4      String stuNumStr = headers.get("x-userid");
5  
6      if(/*invalid student id*/) {
7          return ResponseEntity.badRequest().build();
8      }
9  
10     int studentNumber = Integer.parseInt(stuNumStr.substring(1));
11     List<Study> studies = courseRepository.getStudies(studentNumber);
12     return ResponseEntity.ok(studies);
13 }
```

Listing 5.4: Study endpoint in the `CourseController`.

The annotation in line 1 defines the name and enables the endpoint. The method returns a `ResponseEntity<List<Study>>` that Spring converts into JavaScript Object Notation (JSON). This JSON is then sent to the requester, that is the frontend in our case. In Spring we can get request headers by defining method parameters with the correct annotation. Additionally, we can get path- and query parameters in a similar way.

The responsibility of the endpoint is to first get all required headers. Then it validates that the headers are correct. The student number is especially important. The student number is set by Shibboleth in the frontend. Because employees of the university might try to login as well, we must invalidate those users by only allowing student numbers starting with the letter *k*. Lastly, we call the corresponding method in the `CourseRepository` with sanitized parameters and return the requested data.

In Listing 5.5, we can see the corresponding method that is called in Listing 5.4. In this example, the studies are queried from the database and then filtered by their status. In order to provide only the relevant studies, the Semesterplaner returns those studies that have the status *registered* (German: gemeldet).

```
List<Study> getStudies(int studentNumber) {
    return databaseAccess.getStudies(studentNumber)
        .stream()
        .filter(study -> study.status().equals("gemeldet"))
        .toList();
}
```

Listing 5.5: Method to get the studies of a student in the `CourseRepository`.

The second, more complex method that can be called from the `CourseController` is shown in Listing 5.6. Here we first calculate a cache key from the semester and curriculum id and check if the courses of this study and this semester are already in the cache. If so, we store the courses into the `courses` variable. Otherwise, we fetch the courses from the database, store them into the `courses` variable and cache them. Lastly, we remove the courses that a student already passed and return the list to the API endpoint.

```
List<Course> getCourses(int currId, String semester, int studentNumber) {
    final List<Course> courses;
    String key = cacheKey(currId, semester);

    if (cache.containsKey(key)) {
        courses = cache.get(key);
    } else {
        courses = databaseAccess.getCourses(currId, semester);
        cache.put(key, courses);
    }

    return removePassedCourses(studentNumber, courses);
}
```

Listing 5.6: Method to get the courses for a study of a student in the `CourseRepository`.

In order to remove the already passed and accredited courses, we consult the table from Listing 5.2 that contains records for these kind of courses. If we find a course with the `contentIdentical` flag being set and if one of the tables entries in Listing 5.2 matches the `courseId` (i. e. course class) of the course, it is removed.

5.3 Extended Frontend Data-Model

Now that the backend provided the data, we can fetch it from the frontend. Nevertheless, after fetching the courses, the Semesterplaner adds a few fields to each class so that their values do not have to be calculated over and over again. The added fields are printed in bold in Figure 5.2. The `Study` class includes an array that holds `CourseClass` objects for this study, which makes switching between studies easier. The `CourseClass` class now contains so-called paired course classes, which are courses classes that logically belong together and where the Semesterplaner warns the student if these courses are not taken jointly as explained in Section 3.6. The `Course` class has additional fields that include

- a `selected` flag indicating if the course appears in the calendar or selection pane.
- a `blocked` flag that denotes whether there is a time range marked as unavailable that overlaps the course. Consequently, the course can be effectively displayed as such without having to loop the time ranges every time it needs to be re-rendered.
- a reference to the parent `CourseClass` for better readability and maintainability.
- the time ranges of the sessions sorted by quantity. This array is calculated and stored in advance to reduce unnecessary recalculations when inserted and removed.
- a `latest` flag that triggers animation when the course is added to the calendar.

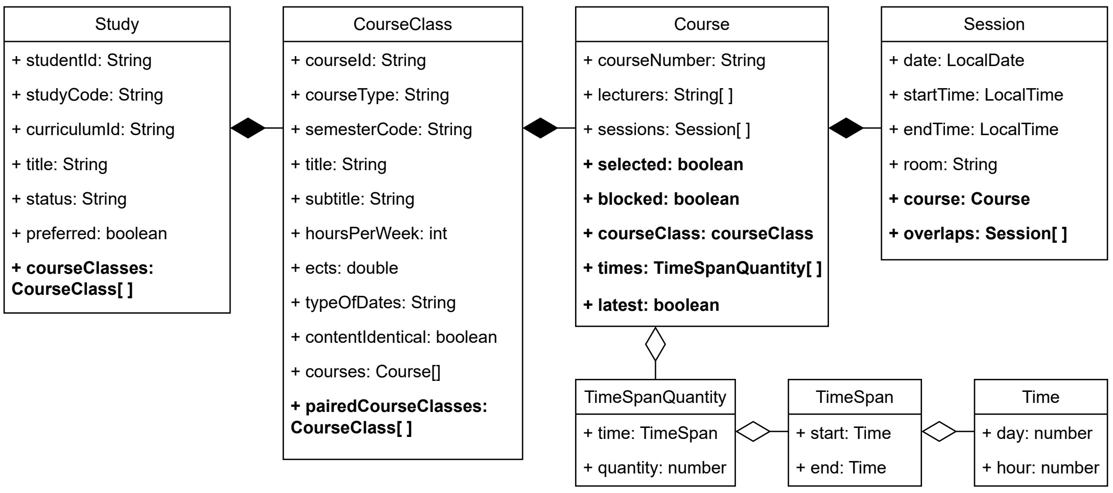


Figure 5.2: The UML diagram of the data adopted to include additional fields for the frontend.

Similarly to the `Course` class, the `Session` class also has a reference to the `Course` class they belong to. Furthermore, `Session` objects have an array to store overlapping `session` objects. These overlapping sessions are recalculated each time a course is added or removed to the calendar. In addition, there are a few classes that make calculations easier. Those include a `Time` class that contains the day of the week and the hour of the day. Additionally, we combine two `Time` objects to create a `TimeSpan` that itself is included in the `TimeSpanQuantity` class used to count the occurrences of sessions that take place on the same day and time in a course.

5.4 Calendar Service

As described in Section 5.3 the frontend fetches data from the backend and transforms it into a new data structure tailored for the frontend. This transformation includes calculating the paired courses. We chose to heuristically pair courses if they share title and subtitle and one of them is of type VL or VO and the other one is of type UE. Alongside this transformation, the calendar service executes tasks initiated by various components. The calendar service acts as the glue that binds different components together, facilitating smooth communication between the components. By serving as a central hub, it not only manages data flow but also ensures that each component gets its data in the required format. In this section we explain the structure of the calendar service and the following sections explain individual components and features that use the calendar service.

5.4.1 Observables

Angular makes extensive use of the `rxjs`-library [12], which is a library for composing asynchronous and event-based programs by using observable sequences. RxJS provides the core type `Observable`, satellite types (`Observer`, `Scheduler`, `Subject`), and operators inspired by array methods (map, filter, reduce, every, ...) to allow handling asynchronous events as collections. The calendar service also makes use of the `Observable` class by creating a `BehaviorSubject` and calling `.asObservable()` on it. When we call `.next(...)` on a `BehaviorSubject` the changes are detected by the `Observable` and propagated to each subscriber. In our case, the subscribers are mostly UI components.

5.4.2 Structure

We designed the calendar service in such a way, that it includes private subjects of values that build the core of the calendar service. On top there are public `Observables` and methods to alter their values. Different components can subscribe to the `Observables` and get notified when a value changes, e. g., in order to change an UI component. Sometimes, the methods change the value of multiple subjects. A small example containing a single subject, observable, and method is given in Figure 5.3.

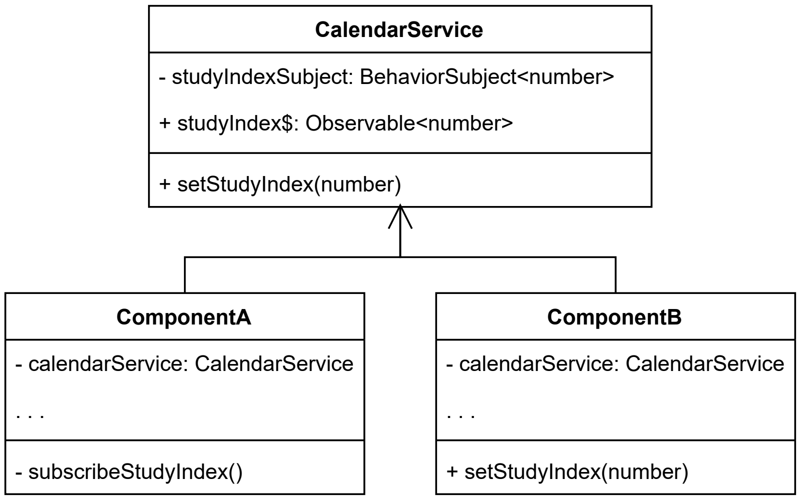

CalendarService
- studyIndexSubject: BehaviorSubject<number>
+ studyIndex$: Observable<number>
+ setStudyIndex(number)
ComponentA
- calendarService: CalendarService
. . .
- subscribeStudyIndex()
ComponentB
- calendarService: CalendarService
. . .
+ setStudyIndex(number)

Figure 5.3: A partial view of the calendar service.

In this example we store the index of the study that the student selects. For that we declare the private field `studyIndexSubject` and create an observable `studyIndex$` from it by calling `.asObservable()` on the subject. By convention, we append a dollar sign to the name of each observable, to easily tell them apart from other fields. Furthermore, we create the method `setStudyIndex()` that only updates the `studyIndexSubject` in this case. These methods tend to be more complex, as careful updating of variables is required, to keep data consistent. In addition, we have multiple components that use the calendar service. Each component, depending on its purpose, subscribes to an observable or calls a method to update something in the calendar service.

5.4.3 Persistence

Another responsibility of the calendar service is to persist data between sessions of the student. That way, students can access their plan any time to make changes or share it with their peers. In order to achieve that, the calendar service facilitates the local storage of the browser. The `localStorage` property in web development allows developers to access a `Storage` object. The stored data is saved across browser sessions [13]. The calendar service facilitates two methods of the `localStorage` property, i. e., `localStorage.setItem('key', 'value')` to store the value for the specified key and `localStorage.getItem('key')` to retrieve the value on reload of the page. The values are mostly JSON collections, so that they can easily be retrieved again.

5.5 Calendar

The component with the most amount of features is the calendar component. Not because it has many features itself, but because it contains many sub-components. As we can see in Figure 5.4, it includes the header that we for simplicity described as an independent component in Section 3.2. The implementation of the header is further elaborated in Section 5.5.4 On the left-hand side, we have a timeline that is not elaborated further, since it only displays the hours to the user and adapts to the zoom level. In the center are the day components with different entry types including (1) courses, (2) the drag-selection, (3)

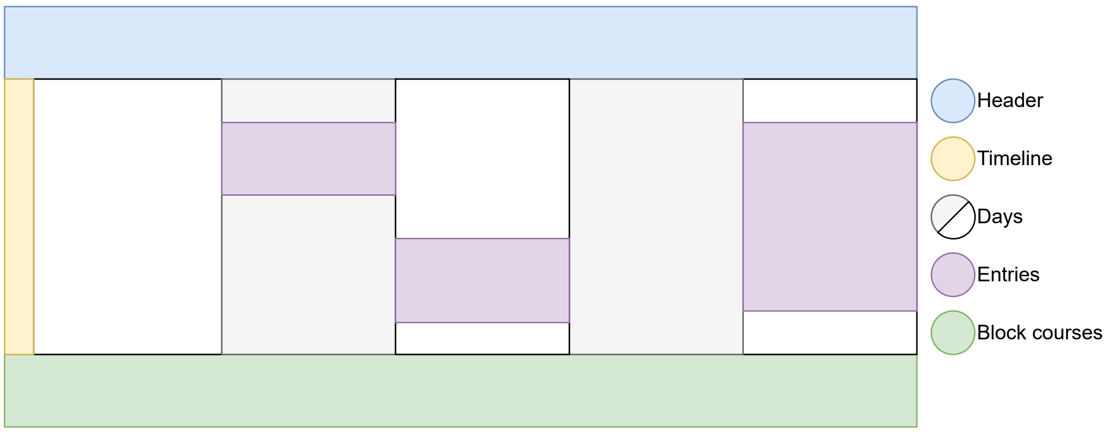


Figure 5.4: A schema of the calendar view.

search-markings, (4) unavailable-markings, and (5) custom entries. The day component is explained in Section 5.5.2 and the entry types are explained in Section 5.5.3. On the bottom we can see the optional block course view that is further explained in Section 5.5.5.

5.5.1 Components

Before taking a look at the various components, we first need to know how a component has to be implemented in Angular. Using the command `ng generate component foo` Angular generates a folder *foo* that represents the component. This folder contains a file for HTML, Cascading Style Sheets (CSS), and TypeScript code each. These files can be edited to define the structure, looks, and functionality of the component [14]. The component can be used with the `<app-foo>`-tag by default. In this thesis we do not elaborate the styling except partially for the entries where we chose a custom approach.

5.5.2 Days

The largest sub-component of the calendar is the body that contains the days of the week. To efficiently display courses that are selected for a day, we introduce an array of observables `selectedCourses$: Observable<Course[]>[]` in the calendar service. The `selectedCourses$` property is an array of length 7, containing an observable for the selected courses from Monday to Sunday in this order. We chose this data structure to store the selected courses, because this makes recalculations of the UI much faster. The calculations are faster because only the day that contains the course sends an event and thus only the relevant part of the calendar is recalculated.

The calendar body usually displays the days from Monday until Friday. Optionally, if there is a course on the weekend, we include the days until that day. To know how many days to display, we initialize the field `weekdays` to the value 4. This variable is the index of the day of the week that the body should still show. Furthermore, to update the field, we subscribe to the last two elements of `selectedCourses$` that correspond to the two days of the weekend. As soon as an observable triggers, we recalculate the `weekdays` to the update number of days. Each day is a separate component and we create it by passing a parameter with the corresponding index for the day of the week. This index is required to subscribe to the correct courses from the data structure.

UI elements To create a pattern with quarter-hourly tiles, the Semesterplaner uses the markup from Listing 5.7. The UI elements therein use the fields `hours`, `pixelPerHour` and `minimumHour`. The `hours` field is an array containing the numbers from 0 to 23 representing the hours and is used to loop over the array in a specified slice. Furthermore, the two other fields are updated on the change of their corresponding observable `pixelPerHour$` and `minimumHour$` from the calendar service.

```
<div *ngFor="let hour of hours | slice: minimumHour : 24"
     [class]="'size-' + this.pixelPerHour">
  <div *ngFor="let i of ['F', 'O', 'U', 'R']"
        class="calendar-tile"
        (mousedown)="tileDown($event)"
        (mousemove)="tileMove($event)"
        (mouseup)="tileUp($event)">
  </div>
</div>
```

Listing 5.7: Creating the tiles for a day in the calendar.

The field `pixelPerHour` is responsible for the height in pixels of a single hour in the calendar. To provide a smooth experience, the calendar service has an array of predefined sizes and keeps track of the current size with an index as can be seen in Listing 5.8. Furthermore, the calendar service provides two methods that allow the user to increase and decrease the size. These two methods are used by the buttons in the toolbar header.

```
private sizes = [64, 96, 128, 160, 192, 224, 256];
private sizeIndex = 2;
private pixelPerHourSubject = new BehaviorSubject<number>(/*128*/);
pixelPerHour$ = this.pixelPerHourSubject.asObservable();

increaseSize() {
    if (this.sizeIndex + 1 >= this.sizes.length) {
      return;
    }
    this.pixelPerHourSubject.next(this.sizes[++this.sizeIndex]);
    this.storeSize();
}
```

Listing 5.8: Part of the calendar service that is responsible for the calendar height.

The day component uses the `minimumHour` field to know from which hour to start the timeline. For this to work, the calendar service sends an update event, when the student adds or removes a course from the calendar. To get the correct minimum hour we calculate the minimum of all selected courses and round down to the nearest integer. When all courses start after 8:00, the calendar still displays that time as the minimum hour.

Time range elements As explained in Section 3.4.3 the student may specify a time range by clicking and dragging in the calendar. To detect the actions by the student, every tile listens to the following events as can be seen in Listing 5.7:

- `mousedown`: When the user presses the mouse on a tile, it results in a `mousedown` event. When this event is fired, we check for modifiers such as the control button (for additive mode) and the shift button (for subtractive mode). Furthermore, we forward the data, i. e., the modifier and a `Time` object to the calendar service. The `Time` object consists of the day of the week and the time as a number, e. g. Monday, 10:15 is converted to a `Time` objects with the values `day=0` and `hour=10.25`. This representation makes calculations for overlaps and rendering easier, because we can later create `TimeSpan` objects, which have a method for calculating overlaps between them. The calendar service stores the data as the current `selectionAnchor` and `selectionCursor`. The `selectionAnchor` serves as a fixed point for the time range while the `selectionCursor` serves as a movable point that is updated when the two following events fire.
- `mousemove`: This event is triggered, when the user moves the mouse over a tile. When this event is fired, we update the hour of the `selectionCursor` to match the current mouse cursor position. Moving the mouse will not change the day of the time range, so that the students are not limited to keeping the cursor in the narrow tunnel that is limited by the width of the day.
- `mouseup`: When the user releases the mouse, a `mouseup` event is triggered. Similarly to the `mousemove` event, we update the `selectionCursor`, but we perform additional actions to store the time range properly in an array of time ranges. This array contains all time ranges that are visible in the calendar. The performed calculations include a comparison of the `selectionAnchor` and `selectionCursor` to determine which of these two is earlier, so that a `TimeSpan` object can be generated. Depending on the modifier, the `TimeSpan` object is differently integrated into the existing set of time ranges. In the default case without modifiers, the existing time ranges are replaced

by the new one. The control modifier adds a new element to the existing time ranges in the array. The `Timespan` class includes a method to detect overlaps between them, and if an overlap with an existing time ranges is detected, the affected time range elements are merged into one time range element as can be seen in Figure 5.5. If the shift modifier is enabled, the new time range is subtracted from the affected time ranges in the array.

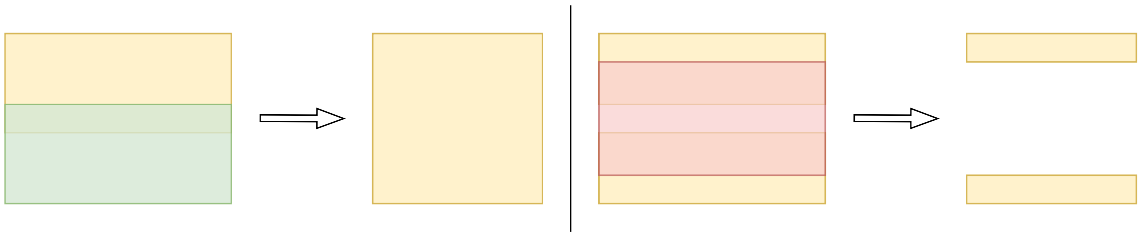


Figure 5.5: Time ranges in addition mode and subtraction mode.

The day component itself subscribes to the existing time ranges that is a `TimeSpan[]` as well as to the `selectionCursor` and `selectionAnchor`. That way the day can filter the corresponding elements and render the selection for this day. The details of rendering selections are similar to other entries and thus are covered in Section 5.5.3.

The major difference to other elements is that selection elements have handles at the top and bottom. The student may click those handles to resize the selection. To achieve that, the element is removed from the selection and the end of the clicked handle is set as the selection cursor and the other one is the selection anchor. From here we can continue as explained earlier with the difference of being in edit mode that is very similar to additive mode which we get when pressing the control key.

5.5.3 Entries

The remaining sub-components of the day component are various entries. All entry types can be created by either selecting a time range as and performing an action or by adding a course from the selection pane at the right-hand side of the calendar. All entry types are stored in the calendar service in separate data structures. Just as for the other properties, the calendar service provides observables for the entry types in `currentSelection$`, `blockedEntries$`, `searchTime$`, `customEntries$` and as we already know `selectedCourses$`. Each day component subscribes to all data structures, except for the selected courses. The day components only subscribe to the corresponding day to avoid redundant updates and re-renderings.

Simple Entries The current selection is an array with the `TimeSpan` objects similar to the blocked entries and the search entries. When the user chooses to convert the current selection to another type of entry, all of the `TimeSpan` objects are moved to the corresponding data structure.

The two simple cases are the block entries and the search entries. For those two entry types, the elements are moved to the corresponding array. Before moving an entry, the Semesterplaner performs a check for overlaps and merges the entries if applicable. To merge the entries, the Semesterplaner loops over all entries in the array and checks whether they overlap with the current entry to be added. If so, the entry is stored in a temporary array and removed from the original one. At the end, we take the minimum start time of all entries as the new start time and similarly for the end time. The new entry is then finally added to the array and persisted in the local storage.

To render these three entry types, we filter the entries to only contain the ones for the corresponding day. Then we convert the start time to a pixel offset relative to the start of the calendar with the formula `pixelPerHour * (entry.start.hour - minimumHour)`. The height of the entry is calculated in a similar fashion by first calculating the duration of the time span and multiplying it by `pixelPerHour`. The remaining considerations involve applying appropriate styling.

```
class CustomEntry {
  constructor(
    public timeSpans: TimeSpan[], // = current selection
    public title: string,         // = ""
    public showEcts: boolean,     // = false
    public ects: number,          // = 0
    public newEntry: boolean      // = true
  ) {}
}
```

Listing 5.9: The structure of the custom entry.

Complex Entries The remaining entry types are custom events and selected courses. The custom events are stored a bit differently from the previous entry types. When the student creates a custom event, the entries are not merged as time search and blockings. Instead, we convert all elements of the selection to a `CustomEntry` object that has a structure as listed in Listing 5.9.

In addition to copying the time range selection into the `timeSpans` array of the custom entry, we initialize the `title` as an empty title and prompt the student to immediately change it by focusing on the title element in the UI. Furthermore, we initialize the flag to show the ECTS with false and the ECTS value to 0. The student can change the `showEcts` flag in the context menu when clicking on the custom entry. As for to the title, the student can change the value of the `ects` field, by typing a numeric value into the corresponding field. The `newEntry` flag is used internally to know which is the last custom event added by the student. Using this flag, the Semesterplaner knows to which entry it should focus the student into so that they enter the title. This value is immediately set to `false` when the element loses focus in the UI so that the student is not prompted to the element over and over again. If the title and ECTS are set to the same value, the Semesterplaner merges those two `CustomEntry` objects into one.

Rendering custom entries and selected courses is not as trivial as the other entry types. Instead of filling the entire width for the specified time span, we have to be able to display multiple courses and custom events next to each other. To achieve this, we store the entries for the calendar in a special data structure as specified in Listing 5.10.

```
calendarEntries!: {
    inner: Course | CustomEntry;
    timeSpan: TimeSpan;
    start: (pixelPerHour: number) => number;
    height: (pixelPerHour: number) => number;
    width: number;
    offset: number;
  }[];
```

Listing 5.10: Calendar entries in a custom data structure for proper rendering.

The `calendarEntries` store the type of the inner entry, so that we can display the right panel in the UI. Additionally, we redundantly store the `timeSpan` of the inner entry, so that we can easier access it in the HTML and CSS for markup and styling. Furthermore, we provide two functions to calculate the start and height of the entry. We opted for functions here, so that we can easily recalculate these values if the student chooses to resize the calendar. The width and offset are calculated in a more complex manner. The values represent the relative width of the entry in the day and the offset specifies the relative offset from the left border of the day.

Rendering Courses and Custom Events To render the entries, we have to calculate the width and offset of the left border for each entry. To achieve this, we adapt the algorithm from Jarderot [15]. The idea of the algorithm is to keep track of the entries in a two-dimensional jagged array representing the columns of a calendar day. The algorithm consists of the following four steps.

First the custom entries and selected courses are merged into an array of `Event` objects. The reason for merging is that the courses and custom events should not overlap. Rather, they should be handled as being the same type in the calendar alignment. These `Event` objects store a `left` and `right` field, where we keep track of how much relative space there is to the left and to the right of the entry. Furthermore, we also store a `TimeSpan` object for each entry, to make calculations for overlaps easier. In addition we store the start and end times of the time range for easier subsequent access. Since the algorithm of Jarderot assumes that the events are sorted, we sort them first by the start time and if they start at the same time, we sort the affected courses by their ending times.

The calculation of the values `left` and `right` is split into two iterations. The first iteration is the original algorithm of Jarderot and the second iteration is an improvement of the algorithm. The addition has the goal of filling the remaining space evenly if possible instead of giving all courses except the last one the same width. In the original algorithm, the last event fills the space up to the right border.

In the first iteration of the algorithm, we loop over all events and insert them. We insert each event by looping over all existing columns and checking whether the current event overlaps with the last event in the column. For checking overlaps, we ignore the edges, e. g. if event A starts at 15:00 and event B ends at 15:00 this does not count as an overlap. We perform this check until we find a fitting column for the entry. If we cannot find a column during the iteration, we append a new column at the end of the array and insert the current event to be the first element in the column.

After putting all events into their respective columns, we can calculate the `left` and `right` values by first calculating the column span of each event. The column span for an event is calculated by counting how many columns to the right we can expand the event without overlapping with any other event. Finally, the `left` value for an event is calculated by dividing its column index by the total number of columns and similarly, the `right` value is calculated by dividing the sum of the column index and the column span of the event by the total number of events.

In the second iteration we group all events by start and end time. This means, that all elements with same start and end time are in a pool and are inspected together. We search for the smallest left value and the highest right value of the entries in each pool. Then we check if the events in the pool are continuous, i. e., whether the whole range from left to right is filled with entries. If the range is continuous, we evenly space the events by replacing their left and right values. We calculate the length of the continuous block and divide by the number of entries. Starting at the previously smallest left location, we can now incrementally set the left and right value by looping over the entries and incrementing the evenly spaced width.

5.5.4 Header

As discussed in Section 3.2, the Semesterplaner has a header with a toolbar. Furthermore, the header also contains a head for each weekday with a button that selects or deselects the time range as explained in Section 3.4.3.

The first element in the header is the current semester. To calculate the semester that shall be displayed to the student, we perform a small process. First we define the border dates of a semester. The winter semester starts on the first of February and the summer semester starts on the first of August. The Semesterplaner calculates the current semester according to these border dates and tries to fetch the courses for the assumed current semester. If the Semesterplaner does not get any courses form the backend, it falls back to the previous semester and repeats the fetching step. This process guarantees that we can find courses for the student to work with.

The next element in the header is the ECTS counter. To display the cumulative ECTS, the header subscribes to the `ects$` field in the calendar service. This field collects the ECTS of all courses added by the student. In order to calculate the value, the calendar service itself combines two `Observables`, i. e., `allSelectedCourses$` and `customEntries$`, that are both be explained in Section 5.5.3. We filter the selected courses to have one course of each course class in our collection. After filtering, we sum up all the ECTS of the courses and custom entries. The right hand side of the header provides a few options to the student. The functionality of the nine buttons is explained in the following paragraphs.

For zooming in and out, the calendar service provides two methods that are called when the student presses a zoom button. One is the `increaseSize()` method from Listing 5.8 and the inverse operation is implemented similarly. First, we check in the methods if resizing is possible (i. e. if we are at the edges of the array that contains the available sizes), and if so we perform the operation and save the index to the local storage.

The student may want to undo and redo actions as well. To achieve this, we forward the intent of the student to the calendar service. The calendar service in return consults the history service that is explained in detail in Section 5.7.

Uploading and downloading the calendar is implemented in a similar fashion to storing to the local storage. In fact, the JSON that the student receives when pressing download, contains the same fields as the local storage. As a result, the download is implemented trivially by checking the file for validity and copying the fields from the JSON to the local storage and reloading the data structures.

The implementation to clear the calendar first contains a check, whether any of the entry collections is non-empty. If this is the case, we prompt the student to confirm, and if they confirm the action, we clear all data structures, reset the `selected`, `show` and `blocked` flags for each course, and delete the history.

5.5.5 Block Courses

To show the block courses, we only need to subscribe to the selected courses from the calendar service. When subscribing, we perform a `flatmap` operation to get all selected courses into a single array. We create a `filter` for the resulting array by checking which courses' parent course class has the `typeOfDates` property set to `Block`. The remaining implementation of this component is responsible for proper rendering.

5.6 Course Selection Pane

The course selection pane is composed of a few sub-components as well. More precisely, the course selection pane hosts (1) study tabs that have been explained in Section 5.6.1, (2) the course search explained in Section 5.6.3, (3) elements for the time search that are rather trivial and thus skipped as well as (4) a list of courses detailed in Section 5.6.2.

5.6.1 Study Tabs

The component responsible for switching between studies displays the studies in tabs that the student can click. Internally, this study tab component subscribes to the array of studies from the calendar service. In addition, we have an observable for the index of the selected study that we subscribe to in the component. When clicking on a tab, we update the index in the calendar service. This update immediately propagates to all components that subscribe to the study index and thus the course list is updated to display the relevant courses for the selected study as well. Furthermore, we shorten the title of the study in the tabs, if there are no other studies with the same name. E. g. "Bachelorstudium Informatik" (bachelor studies in computer science) can be shortened to "Informatik" if there is no masters program taken by the student with the same name.

5.6.2 Course List

The list of courses in the selection pane shows the courses of the study currently selected by the student. If only one study is available, we choose that one and do not show an the study selection tabs. Each course is rendered as an independent component, assuming that the filters from Section 5.6.3 do not hide the course. The course panel contains the corresponding `Course` object. Based on this object we can perform important operations in the calendar service.

The most important action that the student can perform in the course selection pane is to add a course or an course class to the calendar. To add a course class to the calendar, we loop over all contained courses and add those into the calendar, which in essence is the same action as adding a single course, just repeatedly. To mark a course as selected, we need to perform a few steps that are redundant, but save computing effort when rendering. The first step is to set the `selected` field to `true`. Additionally, we mark the course in the calendar service as the most recently added course. This has the purpose of drawing attention from the student using a small animation and a distinct color marking as explained in Section 3.4.1.

Moreover, we add the course to the `selectedCourses` on the correct day the of week. This update is immediately propagated to the calendar by the observables. In addition, we calculate the overlaps for each session and store it into the course, so that we do not have to recalculate this property every time it is needed. We also update the overlaps of the foreign course to stay consistent in our data. Lastly, we also update the minimum hour for the calendar if required and store the selected courses to the local storage. To remove a course from the calendar and move it back into the course selection pane, we can perform the inverse actions.

5.6.3 Search

In this section the search box and the time filters are explained. The search box includes a text field, where the student can enter a string to be searched for in each course. This string is sent to the calendar service as the filter string. In the course list, the filter string is used by the `show(courseClass: CourseClass)` method that determines whether a course should be shown or not.

The `show` method takes into account the filter string and search entries. According to the time search filter, a course class is only displayed, if there is at least one course that entirely fits into the time span of a search entry. Additionally, when all courses are selected and shown in the calendar, we do not show the course class in the selection any more. For filtering by the search string, we allow all `CourseClass` objects for which at least one of the fields `courseType`, `title` or `subtitle` includes the value of the filter. Additionally, we allow the course class if any of the `courses` has a reasonable field (e. g. `lecturer` and `courseNumber`) that matches the filter. For the check of the `courseId`, we eliminate all dots to maximize matching for the student.

5.7 History Service

As mentioned in Section 5.5.4, the student can undo and redo actions by clicking the corresponding buttons. The Semesterplaner handles these requests using the history service. The history service is implemented as a custom ring buffer and uses the abstract class `Action`. For each entry type, there is a class that extends `Action`. These `Action` objects hold relevant data to undo and redo the action, e. g. the class `CourseAction` holds the arrays `addedCourses` and `removedCourses`. Assuming that we are in the correct state, we can use those arrays to recover the previous state by doing the inverse action. Furthermore, we can recover the current state, assuming that we are in the previous state. To keep track of the `Action` objects, the history service provides the following four methods.

- `do(Action)`: The calendar service uses this method to write instances of the `Action` class to the history. When adding a course to the calendar, the calendar service calls `this.historyService.do(new CourseAction([course], []))` to persist the action. This method pushes the action into the ring buffer.
- `undo(): Action | redo(): Action`: The calendar service uses these two methods to reconstruct the expected state. Depending on the action returned, the calendar service adds or removes affected entries.
- `clear()`: This method is called when the calendar is cleared and resets the history service to be empty again. We chose to clear the history in this case to reduce complexity of the Semesterplaner.

6 Evaluation

In order to assess the application's suitability for the broader student audience, a small-scale usability test was conducted among peers. The testing was conducted to evaluate the usability, functionality, and overall user satisfaction of Semesterplaner and aimed to identify potential areas for improvement. This chapter provides a detailed examination of the process and the results of this usability test.

6.1 User Testing Methods

The test was conducted with students enrolled in the computer science bachelor's and master's programs. All participants were required to sign a privacy policy agreement, which confirms that they agree that their data would be utilized for the purpose of testing. This agreement was subsequently forwarded to the IM to facilitate access to the testers' data within the staging deployment of the Semesterplaner. It should be noted that the testing was conducted under restricted conditions. The testing was limited to either my personal computer utilizing the Virtual Private Network (VPN) connection that is provided by the IM, or at the testers device at the JKU campus. This restriction is due to the fact that, despite having Shibboleth authorization and authentication the testing domains for the Semesterplaner are only available in the local network of the JKU.

The experimental approach was characterized by its informality, guided by the protocol attached in the appendix. The test process involved reconstructing the participants' current semester based on their ongoing courses. Once they finished their main task, we identified features, that have not been used and hinted at using that feature, e. g. "How would you search for other courses after course XYZ?". During this process, the participants were encouraged to articulate their thoughts and make suggestions freely. In the end, we concluded the testing with a few questions.

6.2 Findings

The results of the testing indicate that the current approach is satisfactory, but that there is also room for improvement. During the testing phase, a few bugs were identified, which need to be addressed prior to the release of the Semesterplaner to a broader audience. The testers generally had a positive impression of the Semesterplaner and indicated their intention to utilize it if it were available. All participants stated that the approach of a university-provided Semesterplaner is significantly more convenient than searching their courses in KUSSS and manually entering them into a list or spreadsheet. The Semesterplaner was perceived as practical, helpful, and official, and it fulfills its intended function. However, the question of aesthetics generated varied responses, with some participants finding it appealing and others suggesting modifications to the color scheme.

Regarding intuitiveness, the participants encountered difficulties with the drag selection and time search functions. One participant proposed the implementation of available action options within the calendar header. The majority of participants tried searching for time spans in the search bar instead of selecting the time span. Furthermore, some participants were confused by the warning for the lab and lecture connection and expected to open the relevant course when clicking the warning icon. The testing resulted in 43 unique suggestions for improvement from the four participants. These suggestions can be categorized into four groups. Many suggestions concern the specification of custom events, including, but not limited to

- an optional description for the custom event.
- an option to rename a single event in a group of events. Currently, events that have the same name are grouped together and renamed all at one.
- a button to remove a custom event as already implemented for courses.
- a line between custom events or random colors in the bar on the left so that they are easier to tell apart.
- prompting the student into the ECTS field when assigning ECTS and signaling malformed input therein to the student.
- to make the custom event movable in the calendar.

Suggestions concerning the search include that the time search should also be possible by typing the wanted day or time span in the search box in the course selection pane. Furthermore, the text search should not filter the courses, but accumulate the results on the top and display the remaining courses below. In addition, filtering by ECTS, overarching topic and semester is a desirable feature.

Suggestions for the drag selection include selecting a range by just clicking the start and end time while holding the shift key. Many participants experienced difficulties with the selection process and proposed that the selection be removed upon left-clicking, rather than creating a new selection immediately. Furthermore, some suggested to start a new selection when starting at an existing entry. Moreover, actions for the current selection should also be accessible from the calendar header as already mentioned earlier. The final group of suggestions contains proposals for enhancing the quality of life. Noteworthy ideas include

- notes for courses in the calendar. With a note, students may add additional information such as their preference or peers that also take this course.
- a hover tool tip with the full course name in the calendar and in the course selection pane. That way the full name can be easily inspected without needing to open the course dialog.
- an option to display the ECTS in the calendar panel.
- a hint whether there are multiple time spans for a course and a marker for overlaps in the calendar view panels. Currently, overlaps can only be detected when opening the course dialog. Another option is to display a representative for each session to see all time spans of a course.
- a view for ranking the courses, in terms of which to register for first.
- displaying all courses in the selection pane, even if no session exists. Some courses do not have sessions initially and are thus not shown in the Semesterplaner.
- predefined semesters by the union of students. Those can be designed according to the curriculum. Students would then just have to load a semester and can start editing the calendar to their needs.
- adding links for the registration in KUSSS.

7 Future Work

As we have seen, the Semesterplaner has considerable potential. However, the present implementation exhibits deficiencies, as it is not universally applicable to all studies. Other studies have far more complex curricula restrictions that include dependencies on other courses, the time of their completion, and much more. Despite the complexity, an idea might be to extend the Semesterplaner to support all studies of the JKU that do not already provide a predefined schedule for a semester.

Furthermore, it may be worthwhile to add a feature to browse also courses of other curricula and to choose such courses as free electives in the Semesterplaner, instead of inserting placeholder events as outlined in Section 3.4.4. During development this idea was already proposed, however it was considered complex and unnecessary. Nevertheless, as Semesterplaner evolves over time, the potential value of this feature might become more apparent. Furthermore, a course search is already available in myJKU [16] and might be facilitated partially.

During the development of the Semesterplaner, we presented the current development status to other master's students. Following these presentations, the audience offered some suggestions for enhancing the Semesterplaner. One suggestion was to automatically register students that planned their semester using the Semesterplaner. On the one hand, this would cause problems when too many students want to register for the same course. On the other hand, for the case that the course is not overbooked this might be a well-suited feature and worth some attention.

As previously indicated in the evaluation, the integration of links to KUSSS, through which students can register for courses, has the potential to enhance the registration experience. This approach mitigates the need for students to search for courses in KUSSS on the day of registration, when KUSSS is very slow anyway. An alternative solution involves the implementation of a button within the course element in the calendar view, that is only displayed once registration in KUSSS has started. The functionality of this button would be to initiate a registration request for the selected course. This would allow students to view all selected courses in the calendar and register without having to switch between multiple tabs, thereby enhancing the efficiency of the registration process.

The evaluation resulted in additional valuable insights that have the potential for implementation in the future, including the predefined semesters by the student union. This initiative has the potential to reduce the students' workload of planning their semesters, especially, when they are just starting their studies. Furthermore, students who do not study according to the recommended semester plan usually have a few main courses that usually take place in one semester according to the recommended plan and thus some effort can be saved there. For students at the beginning of their studies a feature to warn about the StEOP might be an important addition as well.

In retrospect, it would be more beneficial to retrieve the data on a course basis and then fetch the sessions. This approach would ensure the display of all courses and courses, even in the absence of scheduled sessions. Furthermore, this would have the side effect of shorter loading times, since course class and course data is not sent multiple times and the sessions can be loaded on demand.

The search component would benefit from enhancements, such as drop-downs for various fields. This might make the search easier by e. g. searching for a course type with a specific lecturer. An implementation of this change could be achieved by repurposing the time range elements below the search button, in which we could also include search terms. Additionally, as seen in the evaluation, the search bar should also manage search by time, since many test participants intuitively tried to search for time spans there.

8 Conclusion

The development of the Semesterplaner represents a substantial advancement for students at the JKU, as it provides an official tool to assist students in planning their semesters. The IM's incorporation of a user-centric interface for course search, selection, and visualization is a notable feature, as it eliminates the inefficiencies and errors often associated with a manual scheduling process. Furthermore, the tool aids in the detection of conflicts, displays overlapping time slots, and reminds users to take courses that are intended to be taken simultaneously. With the Semesterplaner, students can approach their planning with greater confidence and efficiency, minimizing risk of subsequent rescheduling in case of scheduling conflicts during the planning process.

Key contributions of the Semesterplaner are the integration of advanced functionalities, such as filtering courses by time, title, lecturer, etc. as well as visualizing arbitrary schedules, and incorporating warnings for incomplete course pairings. Even though the Semesterplaner technically supports any study and lists all available courses, it lacks the capability to recognize dependencies between these courses, and thus does not provide warnings in this regard.

By leveraging technologies like Angular for the frontend and Spring Boot for the backend, the Semesterplaner not only aligns with existing university infrastructure but also allows integration into the myJKU platform. The use of Shibboleth for authentication ensures a secure and seamless user experience, while maintaining university standards.

Chapter 3 of this thesis provides a comprehensive overview of the interface and features, illustrating the Semesterplaner's application for students. Key functionalities such as the toolbar, course selection and the drag selection of time spans in the calendar are explained in detail.

Through user testing, the tool demonstrated its potential to significantly reduce the time and effort required for semester planning. Feedback highlighted the mostly intuitive design of the Semesterplaner and the ease with which students can visualize and adjust their schedules. Nevertheless, there remain opportunities for further enhancement. It was found that the feature to search courses by time spans using the selection is not as intuitive as initially assumed. Suggestions from the feedback include a few changes to further expand user experience. Further suggestions and ideas consist of making the tool available to other studies that a student may take, predefined semesters by the student union and more. Such improvements would not only broaden the adoption of the tool but also ease the lives of numerous students at JKU.

In conclusion, the Semesterplaner represents an innovative solution to a long-standing problem. By providing students with an efficient and reliable scheduling tool, this project contributes to a smoother and more seamless academic journey. It also establishes a model for subsequent development by the IM.

Appendix

Date:__.01.2025

Protokoll			
What did the user want to do?	What happened?	What should happen?	Severity (mild 1 - 5 severe)

Questions after the Testing

What is your overall impression?

Was it intuitive in general?

What was confusing?

Did you achieve what you wanted?

What do you miss about the Semesterplaner?

Would you use the Semesterplaner in the upcoming Planning phase if it was available?

Any suggestions?

Did you like the aesthetics and design of the Semesterplaner?

Did you understand the purpose of each step when using a feature?

How do you like the Semesterplaner in comparison to your traditional planning method?

Bibliography

[1] Christian Fröller. *Ein Web-basiertes Programm zur Erstellung persönlicher Stundenpläne*. ger. 2005 (cit. on p. 2).

[2] Alexander Burghuber. *A Tool for Creating Personalized Semester Schedules*. 2024 (cit. on pp. 2, 4).

[3] *Course Registration | JKU Linz*. URL: `https://www.jku.at/en/degree-programs/students/start-your-studies/course-registration/` (visited on 12/28/2024) (cit. on p. 3).

[4] *Studies Introduction and Orientation Phase (StEOP) | JKU Linz*. URL: `https://www.jku.at/en/degree-programs/students/studies-introduction-and-orientation-phase-steop/` (visited on 12/28/2024) (cit. on p. 4).

[5] *And Suddenly There are Three! Introducing the New Student App, myJKU | JKU Linz*. 2022. URL: `https://www.jku.at/en/news-events/news/detail/news/und-ploetzlich-sind-es-drei-studi-applikation-myjku-gestartet/` (visited on 02/13/2025) (cit. on p. 4).

[6] *myJKU | JKU Linz*. URL: `https://www.jku.at/en/degree-programs/students/myjku/` (visited on 12/12/2024) (cit. on p. 4).

[7] *Shibboleth | Library*. URL: `https://www.jku.at/en/library/find-materials/additional-tools/off-campus-access/shibboleth/` (visited on 12/28/2024) (cit. on p. 4).

[8] *The Shibboleth Project - Shibboleth Consortium*. URL: `https://www.shibboleth.net/about-us/the-shibboleth-project/` (visited on 02/01/2025) (cit. on p. 4).

[9] *Spring | Home*. URL: `https://spring.io/` (visited on 02/13/2025) (cit. on p. 4).

[10] *Home • Angular*. URL: `https://angular.dev/` (visited on 02/13/2025) (cit. on p. 4).

[11] *myJKU | Calendar*. URL: `https://my.jku.at/calendar` (visited on 02/01/2025) (cit. on p. 7).

[12] *RxJS - Introduction*. URL: `https://rxjs.dev/guide/overview` (visited on 01/13/2025) (cit. on p. 28).

[13] *Window: localStorage property - Web APIs | MDN*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage` (visited on 01/16/2025) (cit. on p. 30).

[14] *Angular - Angular components overview*. URL: `https://v17.angular.io/guide/component-overview` (visited on 01/16/2025) (cit. on p. 31).

[15] Markus Jarderot. *javascript - Visualization of calendar events. Algorithm to layout events with maximum width - Stack Overflow*. URL: `https://stackoverflow.com/questions/11311410/visualization-of-calendar-events-algorithm-to-layout-events-with-maximum-width` (visited on 01/19/2025) (cit. on p. 37).

[16] *myJKU | Course-Search*. URL: `https://my.jku.at/course/search` (visited on 01/26/2025) (cit. on p. 46).

List of Figures

List of Listings

Statutory Declaration

I, Adrian Vinojčić, hereby declare that the thesis submitted is my own, unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.

This printed thesis is identical to the electronic version submitted.

Linz, February 23, 2025