

Lecture Connect

ein interaktives Lehreplanungstool

Elias Walder
Johannes Karrer



Bachelorarbeit
Betreuer: Philipp Zech, PhD
17. Dezember 2024

Abstract

Die vorliegende Bachelorarbeit beschreibt die Entwicklung von **Lecture Connect (LeCo)**, einem webbasierten Lehreplanungstool zur automatisierten und interaktiven Erstellung von Stundenplänen. Ausgangspunkt war die bisherige manuelle Planung, die durch mangelnde Übersichtlichkeit, fehlende Filtermöglichkeiten und aufwendige Kollisionserkennung ineffizient und fehleranfällig war.

Die Lösung wurde als moderne Webanwendung konzipiert und implementiert. Das Backend wurde mit **Spring Boot** entwickelt und beinhaltet einen Zuweisungsalgorithmus, der Kurse unter Berücksichtigung von Hard- und Soft-Constraints wie Raumkapazität, zeitlichen Überschneidungen und technischen Anforderungen automatisch zuweist. Das Frontend basiert auf **Angular** und bietet eine benutzerfreundliche Oberfläche mit Kalenderansichten, Drag-and-Drop-Funktionen und diversen Export-Möglichkeiten.

Zusätzlich wurden noch Features wie besondere Filteroptionen und ein semi-automatischer Zuweisungsmodus implementiert, die über die festgelegten Anforderungen hinausgehen. Die Anwendung wird über ein Docker-Container-Setup zur Verfügung gestellt, um eine einfache Wartbarkeit und Skalierbarkeit zu gewährleisten.

Abschließend bietet LeCo nicht nur eine erhebliche Effizienzsteigerung bei der Lehreplanung, sondern auch erweiterte Anpassungsmöglichkeiten für individuelle Anforderungen der NutzerInnen.

Inhaltsverzeichnis

Abstract	i
Inhaltsverzeichnis	iii
Abbildungsverzeichnis	vii
Tabellenverzeichnis	ix
Erklärung	xi
1 Einführung	1
2 Hintergrund und verwandte Arbeiten	3
3 Anforderungen	5
3.1 Backend	5
3.2 Frontend	6
4 Lösungsvorschlag	7
4.1 Use Case Diagramm	7
4.2 Backend	8
4.3 Frontend	8
4.4 Zusätzliche Funktionen	8
4.5 Deployment	9
5 Ablaufbeschreibung	11
5.1 Login	12
5.2 Header	12
5.3 Home	12
5.3.1 Letzte Änderungen	14
5.3.2 Filtern und Hervorheben von Kalenderevents	15
5.3.3 Lens Mode	15
5.4 Wizard und Pre-Wizard	16
5.4.1 Erstellen eines neuen TimeTable-Objekts	16
5.4.2 Vorauswahl der Daten	17
5.4.3 Manuelle Wahl der Daten	17
5.4.4 Lokales Speichern	18
5.5 Editor	18

6	Backend	21
6.1	Einführung	21
6.1.1	Technologiestack	21
6.2	Modellierung	22
6.2.1	Hauptklassen	22
6.2.2	Hilfsklassen	24
6.2.3	Enums	24
6.3	Security und Authentifizierung	25
6.3.1	JWT-basierte Authentifizierung	25
6.3.2	BenutzerInnen	26
6.3.3	Sicherheitsmechanismen	26
6.3.4	Authentifizierungsprozess	26
6.4	Daten	27
6.4.1	Datenbank und Zugriffsschicht	27
6.4.2	Manuelle Datenerfassung	27
6.5	Zuweisungsalgorithmus	27
6.5.1	Vorwort	28
6.5.2	Anforderungen	28
6.5.3	Datenstrukturen	29
6.5.4	Scheduler	31
6.5.5	Ablauf	32
6.5.6	Effizienz und Optimierungen	35
6.5.7	Erweiterungen	36
7	Frontend	37
7.1	Angular	37
7.2	struktureller Aufbau	38
7.3	Components	38
7.4	Services und Converter	41
7.4.1	Converter	41
7.4.2	Data Transfer Object (DTO) Pattern	42
7.4.3	Services	42
7.5	Models	46
7.5.1	CourseSession	46
7.5.2	Course	46
7.5.3	RoomTable	47
7.5.4	TmpTimeTable	47
7.5.5	TimeTable	47
7.6	Login	48
7.6.1	Json Web Token	49
7.7	Home	50
7.7.1	Constructor	50
7.7.2	Laden eines konkreten Tables	51
7.7.3	Aktualisieren aller Kurse	52
7.8	Wizard	53
7.9	Editor	55

8	Evaluierung und Diskussion	57
8.1	Backend	57
8.2	Frontend	57
9	Zusammenfassung und Ausblick	59
	Literaturverzeichnis	61

Abbildungsverzeichnis

4.1	LeCo - Use-Case-Diagramm	7
5.1	Komponentendiagramm	11
5.2	Login-Komponente	12
5.3	Header von Standard-BenutzerInnen	12
5.4	Header von Admin-BenutzerInnen	12
5.5	Home-Ansicht	13
5.6	Beispiel Change Log	14
5.7	Lens Mode	16
5.8	Create Dialog	16
5.9	Pre-Wizard Auswahlmöglichkeiten	17
5.10	Wizard Steps	17
5.11	Editor Ansicht	18
5.12	Liste der nicht enthaltenen Kurse	19
6.1	Klassendiagramm	22
6.2	Struktur AvailabilityMatrix	30
6.3	Überblick Ablauf	32
7.1	Auth Guard Admin Sequenzdiagramm	38
7.2	Ordner Struktur	38
7.3	Komponentendiagramm	39
7.4	Auth Guard Admin Sequenzdiagramm	39
7.5	Ordner Struktur	41
7.6	Data Transfer Object Pattern, Quelle: Dto Pattern Website	42
7.7	Klassendiagram Services und Converter	43
7.8	TableShareService Sequenz Diagramm	44
7.9	CourseSessionDTO Model	46
7.10	CourseDTO Model	46
7.11	TmpTimeTable Model	47
7.12	Lebenszyklus eines TmpTimeTable	47
7.13	TimeTable Model	48
7.14	AuthGuard	49
7.15	Beispiel JWT Token	49
7.16	Ablauf Token Austausch	50
7.17	Construtor Ablauf	51
7.18	Laden eines spezifischen Tables	52
7.19	Aktualisieren der Kalender Daten	52
7.20	Wizard Komponenten Zusammenhänge	54

7.21 Editor Komponenten Diagramm	55
8.1 CORS Fehler	58

Tabellenverzeichnis

5.1	Buttons	13
5.2	Übersicht ChangeTypes	15
5.3	Tabelle der Labels und Farben	15

Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Signiert: Datum:

Signiert: Datum:

1 Einführung

Bisher wurde am Institut für Informatik die Lehreplanung immer manuell durchgeführt. Für die zuständigen Personen, insbesondere unsere Auftraggeberin Cornelia Vidovic, hatte dies unter anderem folgende Nachteile:

- *Mangelnde Übersichtlichkeit:* Da die Erstellung der Pläne mithilfe von Excel-Listen durchgeführt wurde, war es nicht möglich, alle zugewiesenen Kurse übersichtlich darzustellen.
- *Überschneidungen:* Zeitliche Überschneidungen von Kursen konnten nur manuell überprüft werden, wodurch vor allem das Verschieben bestimmter Kurse unnötig erschwert wurde.
- *Keine Filtermöglichkeiten:* Da die Kurse in den Excel-Listen einfach als Text eingetragen wurden, war es aufgrund fehlender Strukturierung der Daten nicht möglich, sie beispielsweise nach Semester oder Studienrichtung zu filtern.

Daher war es seitens der Universität ein großes Anliegen, diesen Prozess zu automatisieren und in ein benutzerfreundliches Tool auszulagern, das die Nachteile der bisherigen Methode kompensiert. Es wurde eine Bachelorarbeit für zwei Personen ausgeschrieben, wobei die verwendeten Technologien frei gewählt werden konnten. Wir haben beschlossen, uns dieser Aufgabe anzunehmen und in enger Zusammenarbeit mit unserer Auftraggeberin **Lecture Connect**, kurz **LeCo** entwickelt, ein modernes und automatisiertes Lehreplanungstool.

Wir haben uns für folgende Aufgabenteilung entschieden:

- Elias Walder:
 - Design und Implementierung der Frontend-Komponenten
- Johannes Karrer:
 - Implementierung der Backend-Komponenten
 - Entwicklung des Zuweisungsalgorithmus
- Gemeinsam:
 - Softwarekonzept
 - Anforderungen und grobes Design
 - Deployment der Software

In den kommenden Kapiteln werden die einzelnen Teile dieser Arbeit nacheinander vorgestellt, beginnend mit den festgelegten Anforderungen an die Software und einer Übersicht über die bereitgestellte Lösung, gefolgt von einer allgemeinen Beschreibung des Ablaufs in Kapitel 5. Anschließend wird in den Kapiteln 6 und 7 detailliert auf die individuellen Teile Backend und Frontend eingegangen. In Kapitel 8 werden die Ergebnisse der Arbeit evaluiert und diskutiert, bevor Kapitel 9 mit einer Zusammenfassung sowie einem kurzen Ausblick auf zukünftige Erweiterungen abschließt.

2 Hintergrund und verwandte Arbeiten

Da das Problem der Raumplanung weit verbreitet ist, gibt es bereits diverse Lösungsansätze mit unterschiedlichen Herangehensweisen. Die wichtigsten werden nachfolgend kurz vorgestellt:

Lösungsansatz mit bipartiten Graphen [Lach and Zorn]

Ein Lösungsansatz wurde von Gerald Lach und Erhard Zorn an der TU Berlin entwickelt. Ihr Algorithmus teilt den Lösungsprozess in zwei separate Schritte: Zuerst werden Zeitkonflikte gelöst, indem den Kursen konfliktfreie Zeitfenster zugewiesen werden. Danach erfolgt die Zuweisung der Räume, wobei Raumkonflikte durch die Modellierung als bipartite Graphen adressiert werden. Die Implementierung verwendet Techniken der ganzzahligen Programmierung, einer mathematischen Methode zur Problemlösung. Dabei liegt ein besonderes Augenmerk auf der effizienten Behandlung großer Probleminstanzen.

Heuristische Lösungsansätze

Viele Algorithmen zur Bearbeitung des Kurs-Raum-Zuweisungsproblems basieren auf heuristischen Ansätzen, darunter die Arbeit von Meyers und Orlin [Meyers and Orlin (2006)], die Very Large-Scale Neighborhood Search (VLSN) verwenden, um die Zeitpläne schrittweise zu verbessern, ohne dabei vollständige Untersuchungen durchzuführen. Ein Kernaspekt dabei ist das zyklische Vertauschen geordneter Teilmengen von Kursen aus disjunktiven Zeitslots. Durch Heuristiken wird der Suchraum dabei effektiv eingeschränkt.

SAT-basierter Ansatz [Nieuwenhuis and Acha (2010)]

Ein weiterer interessanter Ansatz basiert auf der Verwendung von SAT- und MaxSAT-Solvern. Der hier beschriebene Ansatz kodiert das Problem in SAT. Harte Constraints werden dabei durch SAT-Formeln repräsentiert, während weiche Constraints durch MaxSAT oder Weighted-MaxSAT behandelt werden. Bei Letzterem müssen nicht alle Bedingungen, sondern nur möglichst viele erfüllt sein, damit eine Lösung gefunden werden kann, was sich für Soft Constraints sehr gut eignet.

Trotz des breiten Angebots an bereits vorhandenen Lösungen haben wir uns dafür entschieden, einen eigenen Algorithmus zu implementieren, der nicht auf vorhergegangene Arbeiten zurückgreift. Dies begründet sich damit, dass wir einen maßgeschneiderten Algorithmus entwickeln wollten, der die Anforderungen des Instituts für Informatik bestmöglich abdeckt.

3 Anforderungen

Dieses Kapitel beschreibt die funktionalen und nicht-funktionalen Anforderungen an die entwickelte Anwendung, basierend auf den Vorgaben, die zu Beginn des Projekts definiert wurden. Die Anforderungen werden in die beiden Hauptbereiche Backend und Frontend unterteilt, um die jeweiligen Zielsetzungen und Herausforderungen klar zu strukturieren.

3.1 Backend

Das Backend bildet den Kern der Anwendung und übernimmt sämtliche serverseitigen Aufgaben, darunter die Verwaltung der Daten, die Umsetzung des Zuweisungsalgorithmus und die Bereitstellung von APIs für das Frontend. Dadurch ergeben sich folgende Anforderungen:

- **Eingabe von Daten:**

- Unterstützung für den Import von Kurs- und Raumdaten aus Excel-Dateien.
- Kursdaten umfassen Attribute wie Name, Dauer, Bedarf, Vortragende, Computer-Notwendigkeit (Flag), Timing Constraints (Liste von nicht verfügbaren Zeitfenstern) sowie die Anzahl der Gruppen.
- Raumdaten umfassen Attribute wie Raum-ID, Kapazität, Verfügbarkeit von Computern (Flag) und Timing Constraints (Liste von nicht verfügbaren Zeitfenstern).

- **Zuweisungsalgorithmus:**

- Automatische Zuweisung von Kursen zu Räumen basierend auf diversen Hard und Soft Constraints (siehe Abschnitt 6.5.2).
- Möglichkeit zur Überprüfung bestehender Stundenpläne auf Kollisionen (Collision Check), einschließlich Raumkapazität, Timing Constraints und Überschneidungen innerhalb desselben Studiengangs und Semesters.

- **Speicherung und Verwaltung:**

- Persistente Speicherung von Stundenplänen in der Datenbank, wobei jeder Stundenplan eindeutig einem Semester und Jahr zugeordnet ist.
- Unterstützung für die Erstellung eines neuen Stundenplans basierend auf bestehenden Daten.
- Nachvollziehbarkeit aller Änderungen am Stundenplan durch die NutzerInnen.

- **Benutzer- und Sicherheitsanforderungen:**

- Unterstützung für eine Single-User-Nutzung, um parallelen Zugriff zu vermeiden.
- Rollenbasiertes Zugriffsmanagement mit Unterstützung für verschiedene Benutzerrollen (z. B. AdministratorInnen).

3.2 Frontend

Das Frontend dient als Benutzeroberfläche für die Interaktion mit der Anwendung und stellt sicher, dass die Funktionalitäten des Backends intuitiv und effizient genutzt werden können. Die Anforderungen an das Frontend umfassen:

- **Benutzerfreundlichkeit:**

- Bereitstellung einer klar strukturierten und intuitiven Weboberfläche, die die Navigation durch die Anwendung erleichtert.
- Visualisierung des Stundenplans in einer übersichtlichen Kalenderansicht.
- Interaktive Bearbeitungsmöglichkeiten für den Stundenplan, z. B. das Verschieben von Kursen per Drag-and-Drop oder das direkte Zuweisen von Kursen zu Räumen.
- Unterstützung für die Fixierung von Kursen, um diese von weiteren Änderungen auszuschließen.

- **Datenimport und -export:**

- Möglichkeit zum Hochladen von Excel-Dateien für Kurs- und Raumdaten.
- Export des finalen Stundenplans als PDF, um die Ergebnisse für externe Zwecke bereitstellen zu können.

- **Wartbarkeit und Skalierbarkeit:**

- Sicherstellung, dass die Webanwendung im Netzwerk der Universität leicht zugänglich und wartbar ist.
- Modularer Aufbau des Frontends, um zukünftige Erweiterungen oder Anpassungen zu erleichtern.

4 Lösungsvorschlag

Der Lösungsvorschlag für das entwickelte Lehrplanungstool *Lecture Connect* baut auf den in Kapitel 3 definierten Anforderungen auf und erweitert sie zusätzlich um einige nützliche Funktionen.

4.1 Use Case Diagramm

Das in Abbildung 4.1 dargestellte Use-Case-Diagramm illustriert die Interaktionen zwischen den NutzerInnen und den Hauptkomponenten des Systems, *Frontend* und *Backend*. Während das Frontend die Benutzerschnittstelle bereitstellt und Interaktionen ermöglicht, liegt die Verantwortung für Datenverarbeitung und Geschäftslogik im Backend.

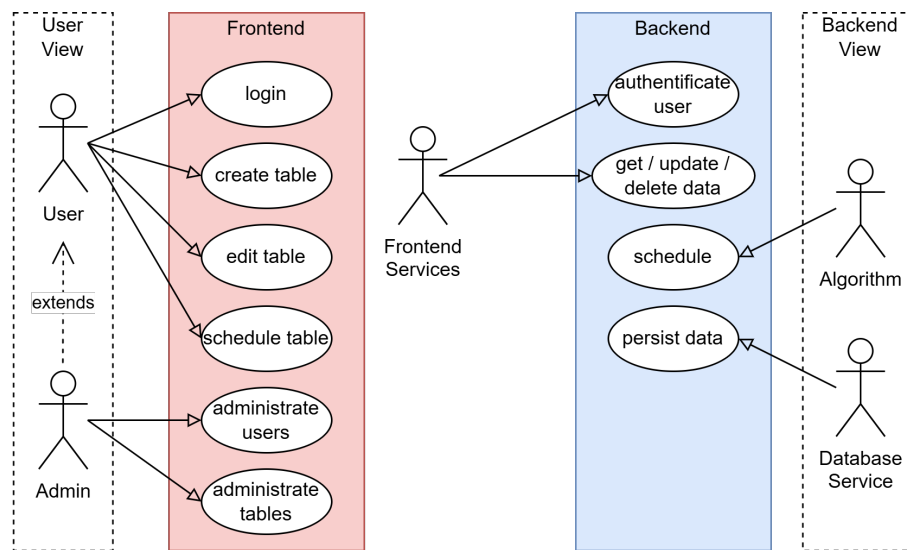


Abbildung 4.1: LeCo - Use-Case-Diagramm

Das Frontend bietet verschiedene Anwendungsfälle, die durch Nutzeraktionen initiiert werden. Ein zentraler Anwendungsfall ist der *Login*, bei dem sich NutzerInnen authentifizieren, um Zugriff auf geschützte Funktionen zu erhalten. Nach erfolgreicher Anmeldung können autorisierte NutzerInnen neue *TimeTable*-Objekte erstellen (*Create Table*) oder bestehende bearbeiten (*Edit Table*). Eine wichtige Funktionalität ist außerdem die Möglichkeit, erstellte Tabellen automatisch zuzuweisen (*Schedule Table*). Für AdministratorInnen stehen zusätzliche Optionen zur Verfügung, wie die Verwaltung von BenutzerInnen (*Administrate Users*) und TimeTables (*Administrate Tables*), wodurch eine detaillierte Kontrolle über Systemressourcen und NutzerInnenrechte gewährleistet wird.

Das Backend bildet die technische Grundlage und stellt sicher, dass alle durch das Frontend angestoßenen Aktionen korrekt verarbeitet werden. Es übernimmt die *Authentifizierung* der

NutzerInnen, überprüft Zugangsdaten und gewährt nur autorisierten Personen Zugriff auf geschützte Funktionen. Zudem bietet es Funktionen zur Verwaltung von Daten, einschließlich der Erstellung, Aktualisierung, Löschung und Abfrage von Ressourcen wie Kursen, Räumen und `TimeTable`-Objekten. Das Backend unterstützt außerdem die Planung und Optimierung von Zuweisungen, die im Frontend initiiert werden (*Schedule*), und gewährleistet, dass alle relevanten Daten sicher und persistent gespeichert werden (*Persist Data*). Diese Architektur ermöglicht ein nahtloses Zusammenspiel zwischen Benutzeroberfläche und den zentralen Backend-Funktionalitäten.

4.2 Backend

Das Backend wurde mit dem Java-Framework Spring Boot entwickelt. Der Fokus liegt auf einer robusten Datenverarbeitung und der Implementierung eines Zuweisungsalgorithmus, der Kurse unter Berücksichtigung von Hard- und Soft-Constraints effizient Räumen und Zeiten zuweist. Die wichtigsten Backend-Funktionen umfassen:

- **Zuweisungsalgorithmus:** Automatische und manuelle Zuweisung von Kursen basierend auf Constraints wie Raumkapazität, zeitlichen Überschneidungen und spezifischen Anforderungen (z. B. Verfügbarkeit von Rechnern).
- **Collision Check:** Möglichkeit, manuell zugewiesene Kurse jederzeit auf Kollisionen zu prüfen.
- **Persistenz:** Speicherung von Stundenplänen, Kursen und Räumen in einer MySQL-Datenbank.
- **Sicherheitsmechanismen:** JWT-basierte Authentifizierung und rollenbasiertes Zugriffsmanagement.

4.3 Frontend

Das Frontend, basierend auf Angular, bietet eine interaktive Benutzeroberfläche, die die Planung und Anpassung von Stundenplänen erleichtert. Die wichtigsten Features des Frontends sind:

- **Kalenderansicht:** Übersicht aller zugewiesenen Kurse in einem Kalender.
- **Kalenderbearbeitung:** Raumbasierte Visualisierung der Stundenpläne mit Drag & Drop-Funktionalität für Kurszuweisungen.
- **Stundenplanerstellung:** Schrittweise Auswahl und Konfiguration der Kurse und Räume.
- **Exportmöglichkeiten:** Erzeugung von PDFs für die Stundenpläne, sowohl für den gesamten `TimeTable` als auch raumspezifisch.
- **Änderungshistorie:** Darstellung der Änderungshistorie für eine lückenlose Nachvollziehbarkeit.

4.4 Zusätzliche Funktionen

Zusätzlich zu den grundlegenden Anforderungen bietet das Tool weitere innovative Funktionen:

- **Semi-Automatic Assignment:** Individuell anpassbare Kombination aus automatischer und manueller Zuweisung.

- **Filteroptionen:** Anpassung der Kalenderansicht nach Studiengängen, Semestern oder spezifischen Kursen.

4.5 Deployment

Die Anwendung wird über einen eigenen Server der Universität Innsbruck bereitgestellt. Für das Deployment wird Docker verwendet, um eine einfache Bereitstellung und Verwaltung der Anwendung zu gewährleisten. Dabei wird eine `docker-compose`-Datei genutzt, die das automatische Erstellen und Starten von drei Containern ermöglicht:

- **Frontend-Container:** Dieser Container enthält die Angular-Anwendung und wird von einem nginx-Server ausgeliefert, um eine schnelle und zuverlässige Bereitstellung zu gewährleisten.
- **Backend-Container:** Der Backend-Container basiert auf einer Spring-Boot-Anwendung und ist so konfiguriert, dass er über eine definierte Portfreigabe auf HTTP-Anfragen reagiert.
- **Datenbank-Container:** Der Datenbank-Container verfügt über eine `mySQL`-Datenbank und ist für die persistente Speicherung der Anwendungsdaten zuständig.

Eine nähere Beschreibung der genannten Features findet sich in den nachfolgenden Kapiteln. Dieser Lösungsvorschlag deckt sich größtenteils mit den beschriebenen Anforderungen, allerdings wurde im Entwicklungsverlauf die Anforderung des Dateiimports aufgrund fehlender Datenquellen verworfen. Die Absenz dieser Funktionalität konnte aber durch manuelle Datenerfassung kompensiert werden (siehe Abschnitt 6.4).

5 Ablaufbeschreibung

Das in Abbildung 5.1 dargestellte Komponentendiagramm zeigt die wichtigsten Komponenten von **Lecture Connect**:

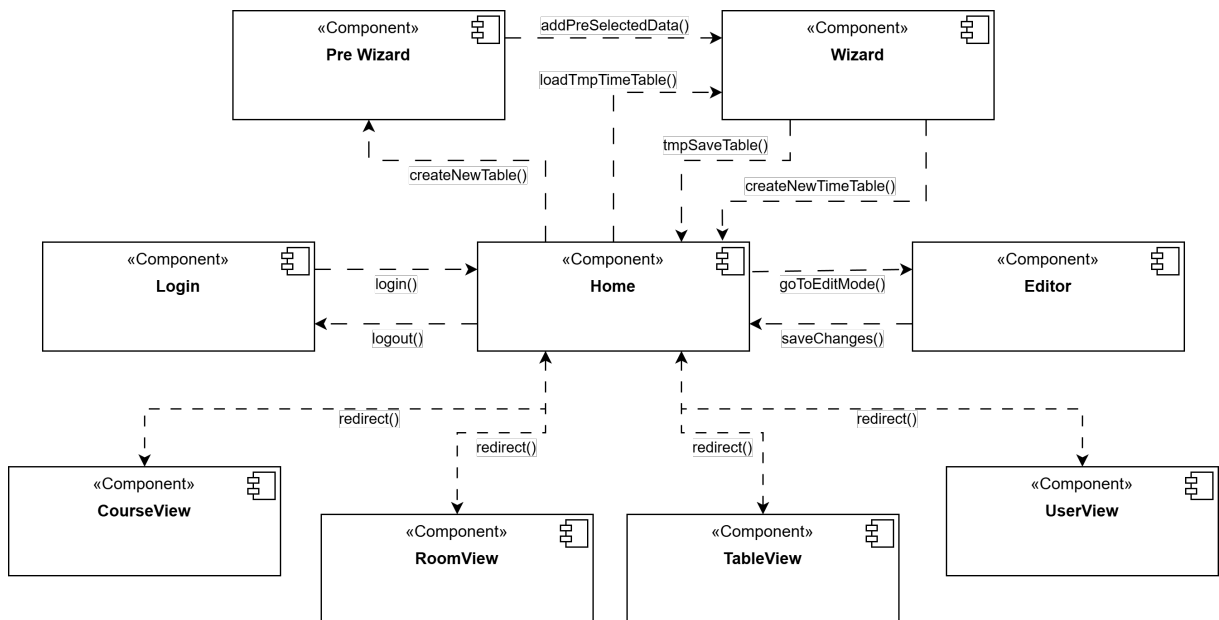


Abbildung 5.1: Komponentendiagramm

Nach erfolgreicher Authentifizierung in der Login-Komponente werden BenutzerInnen in die Home-Komponente weitergeleitet, die eine Übersicht des gerade aktiven TimeTables in einer Kalenderansicht bietet. Von dort können sie in die Editor-Komponente wechseln, in der es möglich ist, Kurse manuell zu verschieben oder zu bearbeiten.

Die Wizard-Komponente führt den NutzerInnen in vier Schritten durch die Erstellung eines neuen Zeitplans, wobei die benötigten Kurse und Räume sowie weitere Details festgelegt werden. Die Pre-Wizard-Komponente unterstützt diesen Prozess, indem sie eine automatische Vorauswahl von Daten erlaubt.

Die übrigen Komponenten CourseView, RoomView, TableView und UserView dienen jeweils der Bearbeitung spezifischer Entitäten in einer Tabellenansicht. Die folgenden Abschnitte werden nochmal genauer auf die einzelnen Komponenten eingehen, wobei die technischen Details in Kapitel 7 beschrieben sind.

5.1 Login

In Abbildung 5.2 ist die Login-Ansicht dargestellt. Die vom NutzerInnen eingegebenen Daten werden für die Authentifizierung an das Backend gesendet. Wenn der Benutzername in der Datenbank existiert und das Passwort mit dem hinterlegten Passwort übereinstimmt, wird für die sich anmeldende Person ein JWT-Token erstellt und sie wird an die Home-Komponente weitergeleitet. Schlägt die Authentifizierung fehl, wird vom Backend eine entsprechende Meldung zurückgegeben. Eine nähere Beschreibung dazu findet sich in den Kapiteln 6.3 sowie 7.6.

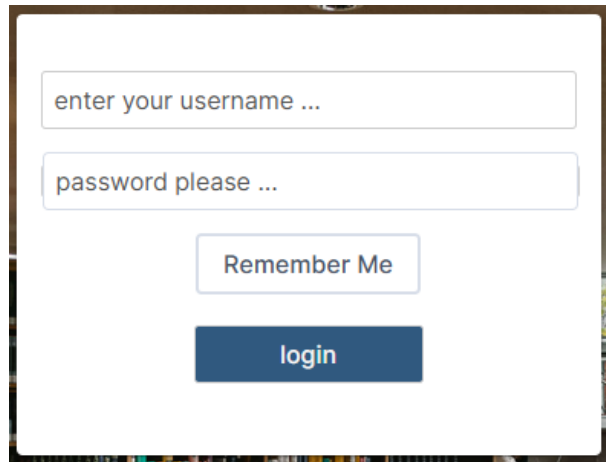


Abbildung 5.2: Login-Komponente

5.2 Header

In Abbildung 5.3 wird der Header von NutzerInnen mit der Rolle `USER` angezeigt. Die Home-Komponente befindet sich im Tab `CONNECTOR`, während `COURSES` und `ROOMS` die in der Datenbank gespeicherten Kurse und Räume in Tabellenform anzeigt. Dort können die entsprechenden Entitäten hinzugefügt, bearbeitet oder gelöscht werden.



Abbildung 5.3: Header von Standard-BenutzerInnen

BenutzerInnen mit der Rolle `ADMIN` verfügen zudem über zwei zusätzliche Tabs `TABLE MANAGEMENT` mit einer Übersicht der vorhandenen TimeTable-Objekte und `USERS`, einer tabellarischen Ansicht aller vorhandenen BenutzerInnen (siehe Abbildung 5.4).



Abbildung 5.4: Header von Admin-BenutzerInnen

Über den Tab `LOGOUT` können sich BenutzerInnen zudem von der Anwendung ausloggen, wodurch der hinterlegte JWT-Token gelöscht wird.

5.3 Home

Die Home-Komponente ist die Hauptansicht von **Lecture Connect**. Von hier aus lassen sich alle anderen Frontend-Komponenten mit einem Klick ansteuern, um eine möglichst intuitive Nutzererfahrung zu ermöglichen. Abbildung 5.5 zeigt eine Darstellung der Home-Ansicht mit dem ausgewählten TimeTable `RealisticTest WS 2024`. In deren Zentrum befindet sich eine eingebettete Kalender-Komponente, die alle zugewiesenen Kurse des derzeit ausgewählten Zeitplans darstellt.

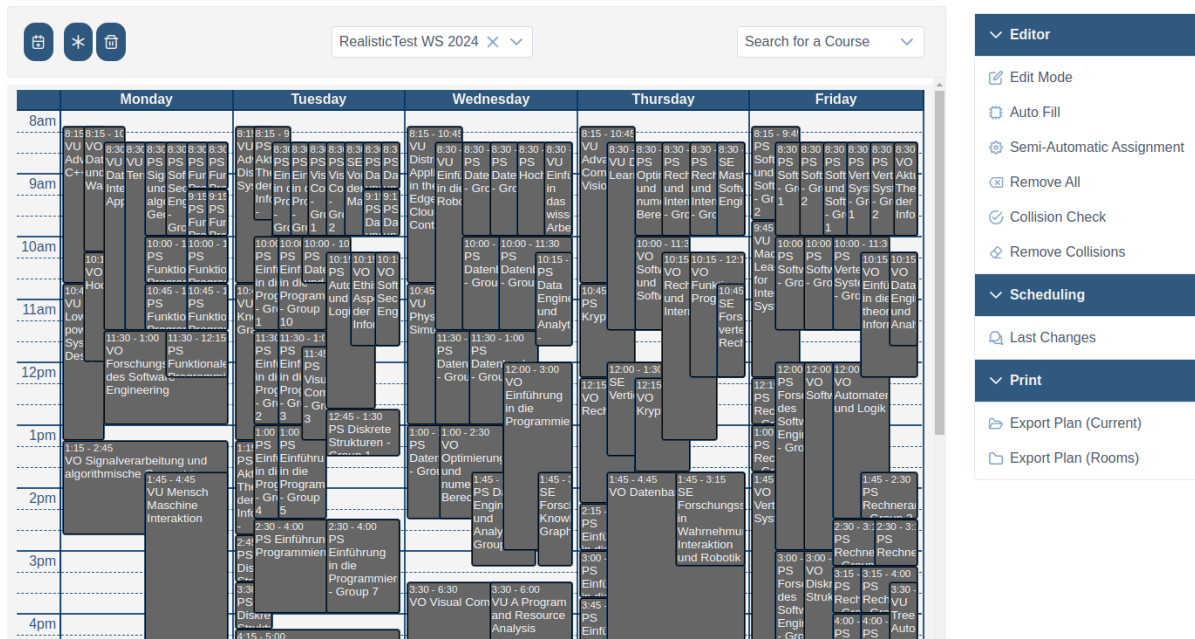


Abbildung 5.5: Home-Ansicht

Im linken, oberen Rand befinden sich drei Buttons, deren Funktionen in Bezug auf die Wizard-Komponente in Tabelle 5.1 näher erklärt wird.

Button	Beschreibung
	Dieser Button initialisiert die Erstellung eines neuen TimeTable-Objekts und öffnet den in Abbildung 5.8 dargestellten <i>Create Dialog</i> .
	Wurde bereits mit der Erstellung eines neuen Zeitplans begonnen, diese aber noch nicht abgeschlossen, können die zwischengespeicherten Daten mithilfe dieses Buttons geladen werden.
	Dieser Button löscht alle während der Erstellung zwischengespeicherten Daten.

Tabelle 5.1: Buttons

Das Menü auf der rechten Seite bietet zudem viele nützliche Funktionen, die im späteren Verlauf dieser Arbeit nochmal näher erklärt werden.

- **Edit Mode:** Leitet den/die BenutzerIn für die manuelle Kursbearbeitung zur Editor-Komponente weiter (siehe Kapitel 5.5).
- **Auto Fill:** Initialisiert die automatische Zuweisung aller nicht zugewiesenen Kurse des TimeTable-Objekts unter Berücksichtigung diverser Hard und Soft Constraints. Der genaue Ablauf des dafür zuständigen Algorithmus sowie weitere Implementierungsdetails werden in Abschnitt 6.5 beschrieben.
- **Semi-Automatic Assignment:** Erlaubt BenutzerInnen eine Kombination aus automatischer und manueller Zuweisung (siehe Kapitel 6.5.7).
- **Remove All:** Entfernt alle Kurszuweisungen, die nicht explizit im Editor fixiert wurden.

- **Collision Check:** Führt eine Kollisionsprüfung aus, um bereits zugewiesene Kurse auf zeitliche Überschneidungen zu überprüfen. Näheres dazu in Abschnitt 6.5.7.
- **Remove Collisions:** Entfernt die Zuweisung aller Kurse, bei denen Kollisionen auftreten.
- **Last Changes:** Zeigt eine tabellarische Ansicht aller durch die UserInnen vorgenommenen Änderungen. Details dazu finden sich in Abschnitt 5.3.1.
- **Export Plan (Current):** Erstellt eine PDF-Datei des gerade angezeigten Kalenders einschließlich aller aktiven Filter. Die diversen Filteroptionen werden in Abschnitt 5.3.2 näher beschrieben.
- **Export Plan (Rooms):** Erstellt eine PDF-Datei mit allen im TimeTable-Objekt befindlichen Räumen, die mindestens einen Kurs zugewiesen haben.

5.3.1 Letzte Änderungen

Um Änderungen an TimeTable-Objekte langfristig nachvollziehbar zu machen, wird jede Bearbeitung seitens der UserInnen dokumentiert. Dabei besteht jeder Eintrag dieses Change Logs aus einem **ChangeType**, einer kurzen Beschreibung samt betroffener Kurse, dem Benutzernamen der Person, die die Änderung vorgenommen hat und dem Zeitstempel der Bearbeitung.

Abbildung 5.6 zeigt beispielhaft die Ansicht des Change Logs eines TimeTable-Objekts. Zuerst wurde die automatische Zuweisung gestartet. Danach wurden noch zwei weitere Änderungen vorgenommen, einmal eine Kursverschiebung innerhalb desselben Raumes und danach die Fixierung eines anderen Kurses.

Table Changes
×

Type	Description	Made by	Date
Select Type ▼			
FIX_COURSE	Course VO Einführung in die Programmierung was fixed in room null at WEDNESDAY, 12:00 - 15:00 Uhr	admin	2024-12-16T17:18:26
MOVE_COURSE	Course VO Funktionale Programmierung was moved from THURSDAY, 10:15 - 12:15 Uhr to FRIDAY, 12:15 - 14:15 Uhr	admin	2024-12-16T17:18:14
APPLY_ALGORITHM	Assignment algorithm was processed successfully.	admin	2024-12-16T15:49:13

Abbildung 5.6: Beispiel Change Log

In Tabelle 5.2 findet sich eine Auflistung aller definierten **ChangeTypes**.

Durch die Kombination von Change Log und Collision Check können BenutzerInnen einerseits nachverfolgen, welche Änderungen in einem bestimmten TimeTable vorgenommen wurden, andererseits stellt es sicher, dass mögliche Probleme aufgrund manueller Bearbeitung umgehend behoben werden können.

ChangeType	Bedeutung
CREATE_TABLE	Das TimeTable-Objekt wurde erstellt.
ASSIGN_COURSE	Ein Kurs wurde einem Raum zugewiesen.
MOVE_COURSE	Ein Kurs wurde von einem Zeitslot auf einen anderen verschoben.
FIX_COURSE	Ein Kurs wurde in einem bestimmten Raum zu einer bestimmten Zeit fixiert.
UNASSIGN_COURSE	Die Zuweisung eines Kurses wurde entfernt.
ADD_COURSE	Ein neuer Kurs wurde hinzugefügt.
REMOVE_COURSE	Ein Kurs wurde entfernt.
APPLY_ALGORITHM	Die automatische Zuweisung wurde durchgeführt.
CLEAR_TABLE	Die Zuweisungen aller nicht fixierten Kurse wurden entfernt.

Tabelle 5.2: Übersicht ChangeTypes

5.3.2 Filtern und Hervorheben von Kalenderevents

Über das Rechtsklick-Menü der Kalender-Komponente stehen AnwenderInnen einige zusätzliche Möglichkeiten für die Darstellung der angezeigten Kurse zur Verfügung.

Einerseits kann eine beliebige Kombination der folgenden Filteroptionen auf die angezeigten Kurse angewendet werden:

- Studiengang: Nur Kurse des ausgewählten Studiengangs werden angezeigt.
- Semester: Nur Kurse des jeweiligen Semesters werden angezeigt.
- Kurstyp: Filtert die Kurse nach Kurstypen wie Vorlesungen oder Proseminaren.

Diese Filter können zusammen mit der Export-Funktionalität genutzt werden, um beispielsweise den Plan für alle Vorlesungen des ersten Semesters des Bachelorstudiums Informatik als PDF-Datei zu exportieren.

Andererseits können bestimmte Gruppen auch farblich hervorgehoben werden. In der Tabelle 5.3 wird jeder Kurstyp mit seiner entsprechenden Farbmarkierung gezeigt. Durch die unterschiedliche Farbgebung können auch mehrere Kurstypen gleichzeitig herausgestellt werden.

Kurstyp	Farbe
VO	Terrakotta (#C36049) ■
VU	Altrosa (#985F53) ■
PS	Korallenrot (#ED5432) ■
SE	Taupe (#6E544E) ■
SL	Schokoladenbraun (#433C3B) ■
PR	Dunkles Mahagoni (#332927) ■

Tabelle 5.3: Tabelle der Labels und Farben

Alle angewandten Filter und Hervorhebungen können zudem über das Rechtsklick-Menü auch wieder rückgängig gemacht werden.

5.3.3 Lens Mode

Um in der Home-Ansicht mehr Informationen über die angezeigten Kurse zu erhalten, wurde der Lens Mode eingeführt. Er kann über das Rechtsklick-Menü aktiviert beziehungsweise deaktiviert

werden und dient als Detailansicht eines zugewiesenen Kurses. Zusätzlich hebt er durch farbliche Markierung alle zum ausgewählten Kurs gehörigen Gruppen oder Splits hervor.

Ins Abbildung 5.7 sieht man ein Beispiel. Der Kurs **PS Datenbanksysteme - Group 1** wurde angeklickt und erscheint in rot ■. Alle weiteren Proseminargruppen desselben Kurses werden außerdem in orange ■ hervorgehoben. So erhält man schnell einen Überblick über die Verteilung der einzelnen Gruppen.

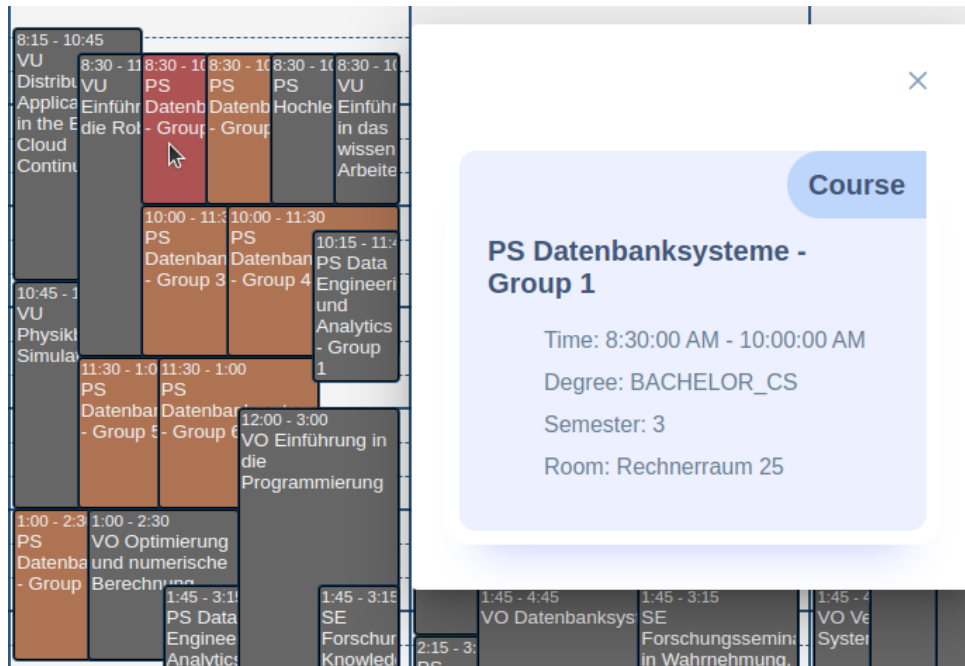


Abbildung 5.7: Lens Mode

5.4 Wizard und Pre-Wizard

5.4.1 Erstellen eines neuen TimeTable-Objekts

Für die Erstellung eines neuen Zeitplans, beziehungsweise die Manipulation während des Wizard-Prozesses zwischengespeicherter Daten, stehen BenutzerInnen die in Tabelle 5.1 dargestellten Buttons zur Verfügung.

Abbildung 5.8: Create Dialog

Wird ein neuer Zeitplan erstellt, erscheint im ersten Schritt das in Abbildung 5.8 dargestellte Dialogfeld. Hier werden Semester, Jahr und ein individueller Name festgelegt. Besteht kein Eintrag mit der gleichen Kombination dieser Daten, werden NutzerInnen zum Pre-Wizard weitergeleitet.

5.4.2 Vorauswahl der Daten

Um den Aufwand für die Zeitplanerstellung im Wizard zu verringern, wurde der Pre-Wizard als unterstützende Komponente hinzugefügt. So kann bereits eine Vorauswahl an Daten getroffen werden.

The image displays four sequential screens of the Pre-Wizard interface, each with a title and a question:

- Preselect Data:** "Do you want to preselect data?" with "yes" and "no" buttons.
- Semester Courses:** "Do you want to only include the courses of the selected semester?" with "yes" and "no" buttons.
- Course Degree(s):** "Do you want to select the degree of courses?" with three radio button options: "Bachelor Computer Science" (selected), "Master Computer Science", and "Master Software Engineering".
- Rooms:** "Do you want to include all Rooms?" with "yes" and "no" buttons.

Abbildung 5.9: Pre-Wizard Auswahlmöglichkeiten

In Abbildung 5.9 werden die jeweiligen Optionen angezeigt. NutzerInnen werden zuerst gefragt, ob sie überhaupt eine Vorauswahl treffen wollen. Falls ja, können sie sich dafür entscheiden, bereits alle Kurse des in Abbildung 5.8 festgelegten Semesters automatisch auszuwählen. Im nächsten Schritt werden die Studiengänge ausgewählt, deren Kurse in der Vorauswahl berücksichtigt werden sollen. Im letzten Schritt können UserInnen zusätzlich noch alle in der Datenbank befindlichen Räume zum TimeTable hinzufügen.

5.4.3 Manuelle Wahl der Daten

Ist die Vorauswahl der Daten abgeschlossen, gelangt der/die UserIn zur Wizard-Komponente. Wie in Abbildung 5.10 dargestellt, ist der Ablauf in vier Schritte unterteilt:



Abbildung 5.10: Wizard Steps

1. *Choose Courses:* Auswahl der Kurse, die im TimeTable enthalten sein sollen.
2. *Define Details:* Festlegen der Kursdetails (Anzahl an Gruppen, Aufteilen der Gesamtdauer auf mehrere Zeiteinheiten).
3. *Choose Rooms:* Auswahl der Räume, die für die Kurszuweisung zur Verfügung stehen.
4. *Room Constraints:* Markieren der reservierten beziehungsweise blockierten Zeiten jedes ausgewählten Raumes.

Wurde der Pre-Wizard für die Vorauswahl der Daten in Anspruch genommen, sind die betreffenden Kurse und Räume in Schritt 1 und 3 bereits ausgewählt. Durch Drücken des **Finish**-Buttons in Schritt 4 wird die Erstellung abgeschlossen und die BenutzerInnen gelangen wieder in die Home-Ansicht.

5.4.4 Lokales Speichern

Der Wizard bietet die Möglichkeit, Daten während des Erstellungsprozesses lokal zwischenspeichern. Dies ermöglicht eine Unterbrechung der Bearbeitung, ohne dass Daten verloren gehen. Wird der Wizard zwischendurch geschlossen, wird der `TmpTimeTable` im LocalStorage gespeichert. In der Home-Komponente besteht anschließend die Möglichkeit, den Wizard wieder aufzurufen und den temporär gespeicherten Table weiterzubearbeiten (siehe Kapitel 5.4.1). Nach Abschluss der Bearbeitung werden die finalen Daten an das Backend gesendet, wo aus dem `TmpTimeTable` ein vollständiger `TimeTable` mit den zugehörigen `CourseSessions` und `RoomTables` generiert wird.

Beim Versuch, den Wizard zu verlassen, ohne die Änderungen zu speichern, werden BenutzerInnen durch den `AuthGuardWizardClose` unterbrochen. Mithilfe des `CanDeactivate`-Interfaces erscheint ein Popup, das sie auf nicht gespeicherte Änderungen hinweist. Entscheiden sie sich dennoch, die Seite zu verlassen, kann es zu Datenverlust kommen. Um dies zu minimieren, wird der `TmpTimeTable` nach jedem abgeschlossenen Schritt im LocalStorage aktualisiert, sodass die bis dahin eingegebenen Daten jederzeit verfügbar bleiben.

5.5 Editor

Während der Algorithmus die Zuweisung der Kurse automatisiert, kann deren Zuweisung im Editor manuell bearbeitet werden. Anstelle eines globalen Kalenders wird für jeden Raum ein eigener Kalender angezeigt, der neben den zugewiesenen Kursen auch die zeitlichen Einschränkungen des jeweiligen Raumes abbildet.

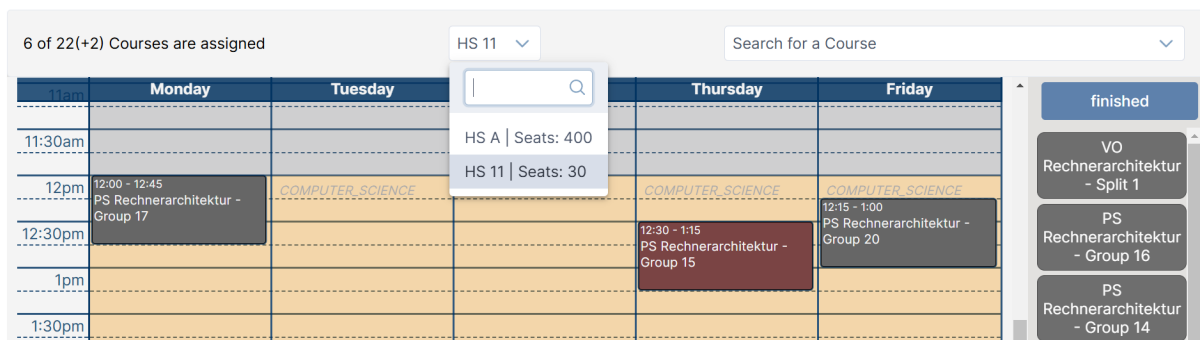


Abbildung 5.11: Editor Ansicht

Abbildung 5.11 zeigt die Editor-Ansicht eines `TimeTable`-Objekts mit 2 Räumen *HS 11* und *HS A*. Sie lässt sich grob in drei Bereiche unterteilen, welche nacheinander kurz vorgestellt werden:

1. Kopfzeile

Die Kopfzeile teilt sich ebenfalls in drei Abschnitte auf, die alle eine unterschiedliche Funktion erfüllen:

- Auf der linken Seite werden Informationen über die Anzahl der zugewiesenen Kurse sowie die Gesamtanzahl an Kursen im `TimeTable` angezeigt. Wurden nach der Erstellung über den Wizard noch weitere Kurse hinzugefügt, wird deren Anzahl in Klammern angegeben (in diesem Fall +2).
- In der Mitte kann mithilfe eines Auswahlfeldes zwischen den im `TimeTable` enthaltenen Räumen gewechselt werden. Als Zusatzinformation wird neben dem Namen auch noch die

Kapazität jedes Raumes angezeigt.

- Auf der rechten Seite befindet sich ein weiteres Auswahlfeld, mit dem man den Ort der Zuweisung eines bestimmten Kurses herausfinden kann. Wählt man dabei einen Kurs aus, der noch nicht zugewiesen ist, erscheint eine entsprechende Meldung und der Kurs wird an die erste Stelle der Liste noch nicht zugewiesener Kurse verschoben. Befindet sich der ausgewählte Kurs bereits im derzeit angezeigten Raum, erhält man ebenfalls eine Meldung. Ansonsten wechselt die Ansicht direkt in den Raum mit der Zuweisung des gesuchten Kurses.

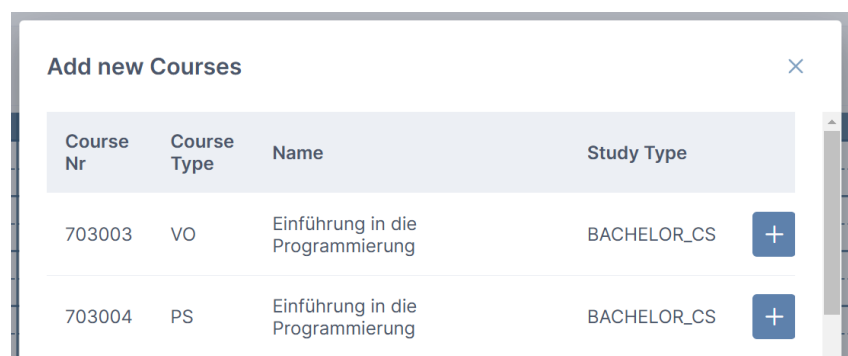
2. Liste der nicht zugewiesenen Kurse

In Abbildung 5.11 wird rechts neben dem Kalender die Liste der nicht zugewiesenen Kurse angezeigt. Per Drag & Drop können sie dem gerade angezeigten Raum zugewiesen werden. Diese sowie alle Änderungen innerhalb der Kalenderansicht werden erst dann gespeichert, wenn der **finished**-Button gedrückt wird. Versuchen BenutzerInnen trotz nicht gespeicherter Änderungen, die Ansicht zu verlassen, erscheint ein Popup mit einem entsprechenden Hinweis.

3. Kalenderansicht

Die Kalenderansicht enthält alle Kurse, die dem momentan angezeigten Raum zugewiesen sind. Außerdem werden die für Informatik reservierten Zeiten in gelb dargestellt, während die blockierten Zeiten grau markiert sind. Innerhalb des Kalenders können Kurse per Drag & Drop verschoben werden. Zusätzlich gibt es noch ein Kontextmenü, das durch Rechtsklick auf einen der zugewiesenen Kurse aufgerufen wird. Dieses bietet folgende Optionen:

- **add new course**: Öffnet den in Abbildung 5.12 angezeigten Dialog. So können Kurse, die bisher nicht im TimeTable enthalten waren, neu hinzugefügt werden.



Course Nr	Course Type	Name	Study Type
703003	VO	Einführung in die Programmierung	BACHELOR_CS
703004	PS	Einführung in die Programmierung	BACHELOR_CS

Abbildung 5.12: Liste der nicht enthaltenen Kurse

- **fix course**: Fixiert den ausgewählten Kurs, sodass er nicht mehr manuell verschoben werden kann. Ist er bereits fixiert, wird stattdessen die Option **free course** angezeigt. Fixierte Kurse sind im Kalender rot markiert (siehe Abbildung 5.11).
- **unassign course**: Entfernt den ausgewählten Kurs aus der Kalenderansicht und fügt ihn der Liste nicht zugewiesener Kurse hinzu.
- **remove group**: Entfernt bei Kursen mit mehreren Gruppen die Gruppe mit der höchsten Nummer aus dem TimeTable.
- **add group**: Fügt dem TimeTable eine neue Gruppe des ausgewählten Kurses hinzu.

6 Backend

6.1 Einführung

Dieses Kapitel soll einen groben Überblick über den Aufbau des Backends geben, einschließlich der wichtigsten Klassen und Komponenten. Der Fokus liegt dabei vor allem auf dem Zuweisungsalgorithmus, der den Kern dieser Arbeit darstellt.

6.1.1 Technologiestack

Der Technologiestack im Backend umfasst folgende Komponenten:

- **Java Spring Boot:** Das Backend basiert auf dem Java-Framework Spring Boot, das eine schnelle und effiziente Entwicklung von Webanwendungen ermöglicht. Es kommen verschiedene Module zum Einsatz, darunter **Spring Security** für die Implementierung von Authentifizierungs- und Autorisierungsmechanismen, **Spring Web** für die Bereitstellung von REST-APIs, **Spring Data JPA** für die Datenbankzugriffe sowie **Spring Boot Test** für das Schreiben und Ausführen von Unit- und Integrationstests.
- **H2-Datenbank:** Für die Entwicklungsumgebung wird die leichtgewichtige H2-Datenbank verwendet. Sie erlaubt es, Daten direkt im Speicher zu verwalten, was schnelle Tests und eine einfache Integration in die Entwicklungsumgebung ermöglicht.
- **MySQL:** In der Produktionsumgebung wird eine MySQL-Datenbank eingesetzt. Diese Datenbank bietet eine solide Grundlage für die Speicherung und Verwaltung der Anwendungsdaten im Live-Betrieb.
- **Lombok:** Zur Reduktion von Boilerplate-Code wird die Java-Bibliothek Lombok genutzt. Sie generiert automatisch Getter- und Setter-Methoden sowie weitere Hilfsmethoden wie `toString()`, `equals()` und `hashCode()`, wodurch die Lesbarkeit und Wartbarkeit des Codes verbessert werden.
- **slf4j:** Für das Logging wird `slf4j` (Simple Logging Facade for Java) verwendet. Dies sorgt für eine einheitliche Handhabung von Log-Ausgaben.
- **jsonwebtoken:** Zur Implementierung von Authentifizierungsmechanismen wird die Bibliothek `jsonwebtoken` (JWT) eingesetzt. Sie ermöglicht die Erstellung und Validierung von JSON Web Tokens (JWTs), die sicherstellen, dass BenutzerInnenzugriffe verifiziert und geschützt sind.

Alle genannten Komponenten werden über Maven als Abhängigkeiten eingebunden. Die Verwaltung und Versionierung der Bibliotheken erfolgt zentral über die `pom.xml`-Datei, wodurch eine einfache Integration und Aktualisierung der verwendeten Technologien gewährleistet wird.

6.2 Modellierung

Ziel der Modellierung war es, ein robustes und erweiterbares System zu schaffen, das sowohl für die BenutzerInnen als auch für den Zuweisungsalgorithmus alle relevanten Informationen bereitstellt. Zentrale Klassen wie **TimeTable**, **Course**, **CourseSession**, **Room** und **RoomTable** bilden dabei das Fundament und definieren die grundlegenden Datenstrukturen. Ergänzt werden sie durch eine Reihe von Hilfsklassen und Enums, die im späteren Verlauf des Abschnitts vorgestellt werden.

6.2.1 Hauptklassen

In Abbildung 6.1 werden die wichtigsten Klassen der Anwendung dargestellt. Die Klassen **Userx** und **UserRole** wurden bewusst weggelassen, da sie nicht für den zentralen Zuweisungsalgorithmus relevant sind und ohnehin in Abschnitt 6.3.2 näher erläutert werden.

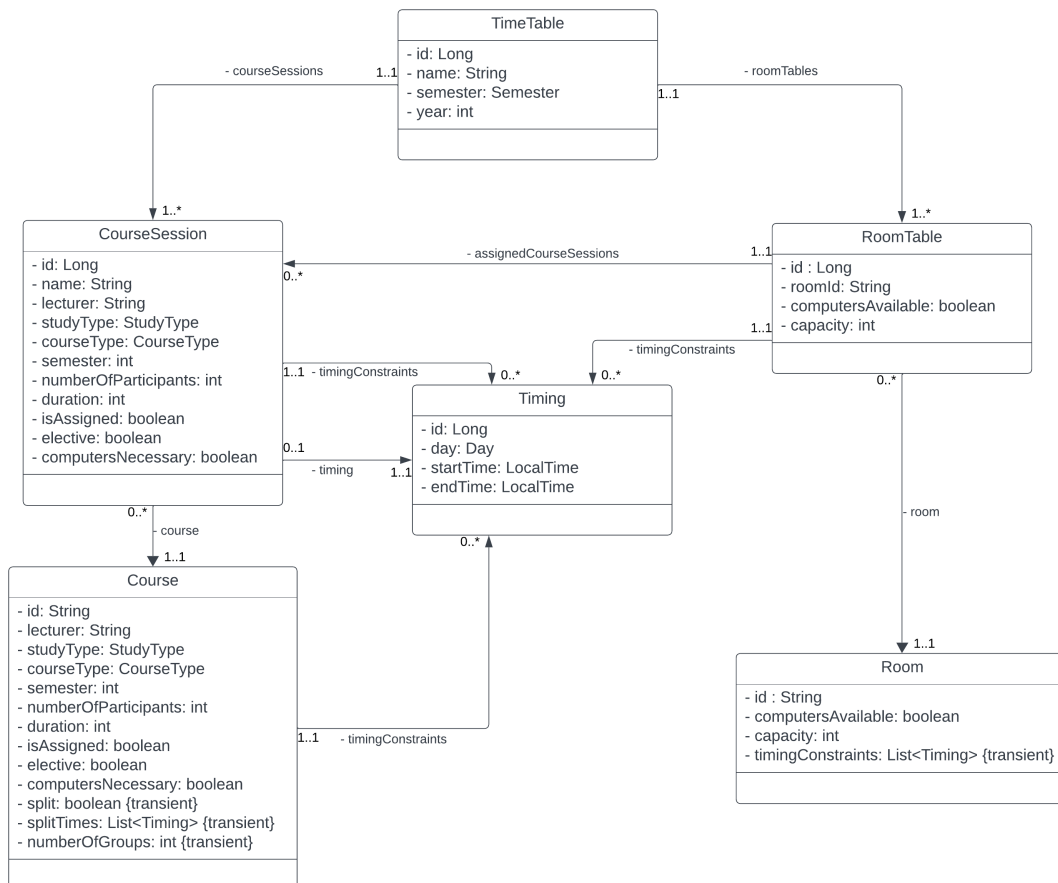


Abbildung 6.1: Klassendiagramm

Nachfolgend werden die abgebildeten Klassen sowie ihre wichtigsten Eigenschaften und Beziehungen erläutert:

- **TimeTable**: Diese Klasse repräsentiert den gesamten Kalender für ein bestimmtes Semester und Jahr. Sie bildet die oberste Ebene des Modells und enthält:
 - Eine Liste von **CourseSession**-Objekten, die die Kurse und deren Zuweisungen repräsentieren.

- Eine Liste von `RoomTable`-Objekten, die die Räume und ihre Verfügbarkeiten abbilden.
- **Course:** Die Klasse `Course` dient als Vorlage für die Erstellung eines oder mehrerer `CourseSession`-Objekte für einen `TimeTable`. Sie enthält die allgemeinen Informationen über einen Kurs (z. B. Name, Dauer, Typ) und zusätzliche Attribute, die für die Initialisierung der `CourseSessions` wichtig sind. Insbesondere die mit `transient` markierten Attribute spielen bei der Erstellung eines neuen `TimeTable` eine Rolle:
 - `split`: Gibt an, ob die Gesamtdauer des Kurses in mehrere kleinere Einheiten aufgeteilt werden soll (z. B. eine Vorlesung mit einer Gesamtdauer von 180 Minuten in zwei Einheiten von 120 und 60 Minuten).
 - `splitTimes`: Eine Liste von Integern, welche die verschiedenen Splits repräsentieren.
 - `numberOfGroups`: Gibt an, wie viele Gruppen ein Kurs hat, falls es sich um einen Gruppenkurs handelt.
- **CourseSession:** Diese Klasse stellt eine spezifische Instanz eines `Course`-Objekts dar, die einem `TimeTable` zugeordnet ist. Sie enthält zusätzliche Attribute wie die Zuweisungsinformationen (z. B. `timing` und `isAssigned`) sowie eine Referenz auf das zugehörige `Course`-Objekt. Durch diese Struktur kann ein Kurs in mehreren Sessions und unterschiedlichen Zeitplänen auftreten.
- **Room:** Die Klasse `Room` dient als Vorlage für die Erstellung eines `RoomTable`-Objekts bei der Initialisierung eines neuen `TimeTable`. Das mit `transient` markierte Attribut `timingConstraints` speichert die Grundbelegung des Raums, also die Zeitfenster, die für bestimmte Studiengänge (z. B. Informatik) reserviert oder blockiert sind.
- **RoomTable:** Diese Klasse repräsentiert den Kalender eines bestimmten Raums innerhalb eines `TimeTable`. Sie enthält Informationen zur Kapazität, zur Verfügbarkeit von Computern (`computersAvailable`) sowie eine Liste aller Zuweisungen (`assignedCourseSessions`) und Zeitbeschränkungen (`timingConstraints`).
- **Timing:** Die Klasse `Timing` repräsentiert eine spezifische Zeitspanne und wird in verschiedenen Kontexten verwendet:
 1. Als *Timing Constraint* eines Raumes, um festzulegen, welche Zeiten für Informatik oder andere Studienrichtungen reserviert oder blockiert sind.
 2. Als *Timing Constraint* eines Kurses, um Zeiträume zu definieren, in denen ein Kurs nicht stattfinden darf.
 3. Als *Timing* einer `CourseSession`, um den zugewiesenen Zeitrahmen eines spezifischen Kurses zu speichern.

Ein `Timing`-Objekt umfasst den Tag (`day`) sowie die Start- und Endzeit (`startTime`, `endTime`).

Das Klassendiagramm zeigt, dass sowohl die Klassen `CourseSession` als auch `RoomTable` in einigen Attributen mit ihren Vorlagen `Course` und `Room` übereinstimmen. Diese Redundanzen sind eine bewusste Designentscheidung, um sicherzustellen, dass ein `TimeTable` eigenständig existieren kann, ohne von den ursprünglichen Kursen oder Räumen abhängig zu sein. So wird die Datenintegrität gewahrt, selbst wenn Kurse oder Räume außerhalb eines spezifischen `TimeTable` gelöscht oder geändert werden.

6.2.2 Hilfsklassen

GlobalTableChange

Die Klasse `GlobalTableChange` dient der Nachverfolgung aller Änderungen, die von `BenutzerInnen` an `TimeTable`-Objekten vorgenommen wurden. Sie stellt sicher, dass Änderungen wie das Hinzufügen oder Entfernen von Kursen und Räumen sowie Anpassungen der Zuweisungen jederzeit nachvollziehbar bleiben. Jede Instanz der Klasse repräsentiert eine spezifische Änderung und enthält folgende Attribute:

- **description:** Eine kurze Beschreibung der vorgenommenen Änderung.
- **changeType:** Der Typ der Änderung, der als `ChangeType`-Enum gespeichert wird.
- **timeTable:** Der Name des betroffenen `TimeTable`.
- **date:** Das Datum und die Uhrzeit, zu der die Änderung vorgenommen wurde.
- **changeUser:** Der/die `BenutzerIn`, der/die die Änderungen vorgenommen hat.

TimeStampedEntity

Die abstrakte Klasse `TimeStampedEntity` dient als Basisklasse, um eine konsistente Speicherung von Erstellungs- und Aktualisierungszeitpunkten in den Datenbankentitäten zu gewährleisten. Durch die Nutzung dieser Klasse können alle abgeleiteten Entitäten automatisch die Attribute `createdAt` und `updatedAt` verwenden, ohne dass deren Verwaltung individuell implementiert werden muss.

Die in der Klasse verwendeten Methoden `onCreate()` und `onUpdate()` sind mit den JPA-Lifecycle-Events `@PrePersist` und `@PreUpdate` verknüpft, um sicherzustellen, dass die Zeitstempel bei jeder relevanten Operation automatisch gesetzt oder aktualisiert werden. Durch diese Implementierung wird eine einheitliche und effiziente Nachverfolgbarkeit von Änderungen in der Datenbank erreicht.

6.2.3 Enums

Die folgenden Enums werden in der Anwendung verwendet, um wichtige Eigenschaften und Konfigurationen standardisiert darzustellen:

- **ChangeType:** Dieses Enum beschreibt die möglichen Typen von Änderungen an einem `TimeTable`. Dazu gehören beispielsweise:
 - `CREATE_TABLE`: Erstellung eines neuen `TimeTable`.
 - `ASSIGN_COURSE`: Zuweisung eines Kurses zu einem Raum.
 - `APPLY_ALGORITHM`: Anwendung des Zuweisungsalgorithmus.
 - `CLEAR_TABLE`: Zurücksetzen eines `TimeTable`.
- **CourseType:** Dieses Enum kategorisiert die verschiedenen Arten von Kursen. Der `CourseType` ist unter anderem bei der Erstellung eines neuen `TimeTable`-Objekts wichtig, da die unterschiedlichen Typen spezifische Eigenschaften aufweisen:
 - `VO`: Vorlesung (kann in mehrere Splits aufgeteilt werden)
 - `VU`: Vorlesung mit Übung (kann in mehrere Splits aufgeteilt werden)

- PS: Proseminar (kann mehrere Gruppen haben)
- SE: Seminar (kann mehrere Gruppen haben)
- SL: Studienleistung (kann mehrere Gruppen haben)
- PR: Praktikum (kann mehrere Gruppen haben)
- **Day:** Dieses Enum repräsentiert die Wochentage, an denen Kurse stattfinden können.
- **Semester:** Dieses Enum unterscheidet zwischen den beiden Semestertypen Sommersemester und Wintersemester.
- **StudyType:** Dieses Enum beschreibt die Studiengänge, für die ein Kurs angeboten wird:
 - BACHELOR_CS: Bachelorstudium Informatik
 - MASTER_CS: Masterstudium Informatik
 - MASTER_SWE: Masterstudium Software Engineering
- **TimingType:** Dieses Enum wird verwendet, um die zeitlichen Einschränkungen eines Raumes zu beschreiben:
 - COMPUTER_SCIENCE: Zeitfenster, die für Informatik reserviert sind.
 - BLOCKED: Zeitfenster, die für andere Studiengänge blockiert sind.

Diese Hilfsklassen und Enums tragen dazu bei, die Struktur der Anwendung klar und wartbar zu gestalten, indem sie standardisierte und wiederverwendbare Datenformate bereitstellen.

6.3 Security und Authentifizierung

Die Anwendung verwendet eine **JWT-basierte Authentifizierung**, um die Sicherheit und den Schutz der BenutzerInnenressourcen zu gewährleisten. Dieses Kapitel beschreibt die implementierten Sicherheitsmechanismen sowie den Authentifizierungsprozess.

6.3.1 JWT-basierte Authentifizierung

JSON Web Tokens (JWT) werden als Authentifizierungsmethode verwendet, um eine zustandslose Architektur zu ermöglichen. Der Token enthält verschlüsselte Informationen über den/die BenutzerIn, wie den Benutzernamen und die Rollen, und wird nach einer erfolgreichen Anmeldung generiert.

Der Token wird bei jeder Anfrage an geschützte Endpunkte im HTTP-Header **Authorization** mitgeführt. Die Struktur des Tokens folgt dem Standardformat:

`header.payload.signature`

Erstellung des Tokens

Die Generierung des JWT erfolgt im `JwtService`. Nach erfolgreicher Verifizierung der BenutzerInnen-Anmeldedaten im `AuthenticationController` wird der Token wie folgt erstellt:

- **Header:** Spezifiziert den Algorithmus (z. B. HS256) und den Typ des Tokens.
- **Payload:** Enthält Informationen wie `username`, `roles` und weitere Metadaten.
- **Signature:** Eine signierte Hashsumme, die mit einem geheimen Schlüssel (**SECRET**) erzeugt wird, um die Integrität und Authentizität des Tokens sicherzustellen.

Validierung des Tokens

Der `JwtAuthenticationFilter` ist für die Überprüfung des Tokens bei eingehenden Anfragen verantwortlich. Der Prozess umfasst:

1. Extraktion des Tokens aus dem `Authorization`-Header.
2. Validierung der Signatur und Prüfung des Ablaufdatums.
3. Ableitung des Benutzernamens aus dem Token und Laden der BenutzerInneninformationen mithilfe eines `CustomUserDetailsService`.

Nach erfolgreicher Validierung wird der/die BenutzerIn in den `SecurityContext` gesetzt, um ihm/ihr Zugriff auf geschützte Ressourcen zu gewähren.

6.3.2 BenutzerInnen

Die Anwendung verwendet eine datenbankgestützte BenutzerInnenverwaltung, implementiert durch den `UserService`. Für differenzierte Zugriffsrechte wurden zwei Rollen (`USER` und `ADMIN`) definiert, wobei UserInnen mit der Rolle `ADMIN` zusätzlich die Berechtigung zur BenutzerInnenverwaltung haben. Dies umfasst sowohl das Erstellen neuer, als auch das Editieren und Löschen bestehender NutzerInnen.

6.3.3 Sicherheitsmechanismen

Die Sicherheitsmechanismen in der Anwendung umfassen:

- **Passwortverschlüsselung:** Alle Passwörter werden mit einem starken Hashing-Algorithmus (BCrypt) verschlüsselt, bevor sie gespeichert werden.
- **Rollenbasierte Zugriffskontrolle:** Zugriff auf bestimmte Endpunkte wird anhand der Benutzerrollen eingeschränkt, wie im `WebSecurityConfig` definiert.
- **CSRF-Schutz:** Da eine JWT-basierte Authentifizierung verwendet wird, wurde der CSRF-Schutz deaktiviert (`csrf().disable()`), da Tokens von Natur aus gegen CSRF-Angriffe resistent sind.
- **CORS-Konfiguration:** Der Zugriff auf die API wird durch eine Cross-Origin Resource Sharing (CORS)-Konfiguration gesteuert, um den sicheren Zugriff von verschiedenen Quellen zu ermöglichen.

6.3.4 Authentifizierungsprozess

Der Authentifizierungsprozess läuft wie folgt ab:

1. Ein/eine BenutzerIn sendet eine Anfrage an den `/auth/login`-Endpunkt mit seinem/ihrer Benutzernamen und Passwort.
2. Der `AuthenticationController` überprüft die Anmeldedaten mithilfe des `UserService`.
3. Nach erfolgreicher Authentifizierung wird ein JWT generiert und an den/die BenutzerIn zurückgegeben.
4. Bei nachfolgenden Anfragen sendet der/die NutzerIn den JWT im `Authorization`-Header mit.

5. Der `JwtAuthenticationFilter` überprüft den Token und authentifiziert den/die BenutzerIn, falls der Token gültig ist.

Die Kombination aus JWT-basierter Authentifizierung und rollenbasierter Zugriffskontrolle bietet eine flexible und sichere Grundlage für die BenutzerInnenverwaltung. Durch die Verwendung von zustandslosen Tokens wird die Skalierbarkeit der Anwendung verbessert, während gleichzeitig sensible Informationen wie Passwörter durch moderne Sicherheitsstandards geschützt werden.

6.4 Daten

Dieser Abschnitt beschreibt den Aufbau der Persistenzschicht von **Lecture Connect** sowie den Prozess der Datenerfassung.

6.4.1 Datenbank und Zugriffsschicht

Die Anwendung nutzt eine relationale Datenbank zur Speicherung und Verwaltung der benötigten Informationen. Die Kommunikation mit der Datenbank erfolgt über die Java Persistence API (JPA) in Kombination mit Repository-Interfaces, die grundlegende Datenbankoperationen wie CRUD (Create, Read, Update, Delete) abstrahieren. Dies ermöglicht eine einfache und flexible Handhabung der Datenbankoperationen, unabhängig vom verwendeten Datenbankmanagementsystem.

In der Entwicklungsumgebung wurde die H2-Datenbank gewählt, da sie schnelles Testen ohne die Notwendigkeit eines externen Datenbankservers erlaubt. In der produktiven Umgebung wird eine MySQL-Datenbank eingesetzt, um eine skalierbare und persistente Speicherung der Daten zu gewährleisten.

6.4.2 Manuelle Datenerfassung

Ursprünglich war geplant, eine Import-Funktionalität für die Daten von Kursen und Räumen zu implementieren. Diese hätte es ermöglicht, relevante Daten in einem standardisierten Format (z. B. CSV oder Excel) aus externen Quellen zu importieren. Aufgrund der unzureichenden Verfügbarkeit von Datenquellen in geeigneter Form musste jedoch auf eine manuelle Datenerfassung zurückgegriffen werden. So besteht der aktuelle Datensatz aus folgenden Einträgen:

- **Räume:** Es wurden 29 Räume manuell in die Datenbank eingepflegt, darunter 9 Rechnerräume. Somit sind alle relevanten Räumlichkeiten abgedeckt, die in der Praxis genutzt werden.
- **Kurse:** Zusätzlich wurden 95 Kurse manuell erfasst, sowohl aus dem Curriculum Bachelor Informatik, als auch aus dem Master Informatik und dem Master Software Engineering. Diese Kurse repräsentieren den Großteil der in der Praxis durchgeführten Veranstaltungen.

Sowohl bei der Erfassung der Räume als auch der Kurse wurde Fokus auf die Genauigkeit und Richtigkeit der Daten gelegt, um den Aufwand für die UserInnen von Anfang an zu minimieren.

6.5 Zuweisungsalgorithmus

Der Algorithmus zur Zuweisung von Kursen zu Räumen basiert auf einer Mischung aus Greedy- und Backtracking-Ansatz, der diverse Einschränkungen wie Raumkapazitäten, zeitliche Überschneidungen und technische Anforderungen berücksichtigt. Dieser Abschnitt beschreibt detailliert, welche Anforderungen an den Algorithmus gestellt werden und welche Lösungen dafür implementiert wurden.

6.5.1 Vorwort

Die Entwicklung des hier vorgestellten Zuweisungsalgorithmus war ein zentraler Bestandteil dieses Projekts. Es handelt sich dabei um eine komplett eigenständige Lösung, die ohne externe Vorlagen oder Referenzen konzipiert und implementiert wurde. Dies ermöglicht es, die spezifischen Anforderungen und Rahmenbedingungen des Instituts für Informatik bestmöglich abzubilden.

6.5.2 Anforderungen

Der Ausgangspunkt für den Algorithmus ist eine Liste von Kursen, deren Einträge einem Raum aus einer Liste von Räumen zugewiesen werden sollen. Dabei ist das Ziel, alle Kurse so zuzuweisen, dass verschiedene Bedingungen (Constraints) erfüllt sind. Hier werden zwei Arten von Constraints unterschieden:

- **Hard Constraints:** Diese Constraints müssen unbedingt erfüllt werden, damit eine Zuweisung durchgeführt werden kann. Innerhalb des Algorithmus entsprechen Hard Constraints *Filterbedingungen*.

1. Raumkapazität: Die Raumkapazität muss für die TeilnehmerInnenzahl des Kurses angemessen sein. Das bedeutet, dass es auf der einen Seite mindestens einen Sitzplatz pro TeilnehmerIn geben muss, auf der anderen Seite soll die Raumkapazität auch nicht viel größer sein als die Anzahl der TeilnehmerInnen, um den vorhandenen Platz effizient zu nutzen. Daher wurde folgendes Constraint für eine Raumkapazität c und eine TeilnehmerInnenanzahl n festgelegt:

$$c \geq n \wedge c \leq 2n$$

2. Timing Constraints des Raumes: Für jeden Raum gibt es vor Semesterbeginn bereits eine Grundbelegung. Das bedeutet, dass es bestimmte Zeiten gibt, die für andere Studienrichtungen reserviert sind und für den Zuweisungsalgorithmus daher blockiert sein müssen.
3. Timing Constraints des Kurses: Auch für Kurse können im Vorhinein bestimmte Zeiten festgelegt werden, an denen sie nicht zugewiesen werden können.
4. Kurse desselben Semesters: Kurse desselben Semesters dürfen sich zeitlich nicht überschneiden, da Studierende die Möglichkeit haben sollen, alle für ein Semester vorgesehenen Kurse uneingeschränkt zu besuchen. Von dieser Bedingung ausgeschlossen sind 2 Arten von Kursen, nämlich Gruppenkurse und Wahlmodule, für deren Verarbeitung eine spezielle Datenstruktur `ConcurrentCourseLimiter` verwendet wird. Details dazu finden sich in Abschnitt 6.5.3.

- **Soft Constraints:** Diese Constraints sollten bei einer Zuweisung erfüllt sein. Innerhalb des Algorithmus entsprechen Soft Constraints *Sortierbedingungen*.

1. Computer: Kurse, für die ein Computer benötigt wird, sollen in Rechnerräumen stattfinden, während andere Kurse nicht in Rechnerräumen stattfinden sollen. Im Falle mangelnder Kapazitäten können einzelne Kurse aber auch anders zugewiesen werden.
2. Verteilung von Gruppenkursen: Kurse mit mehreren Gruppen wie Proseminare sollen möglichst am selben Tag stattfinden.

3. Bevorzugte Zeiten: In der Grundbelegung jedes Raumes gibt es neben Zeiten, die für andere Studiengänge reserviert sind, auch Zeiten, die explizit für Informatik reserviert sind. Daneben gibt es noch leere Zeitslots, die bei Bedarf zugewiesen werden können. Ziel des Algorithmus ist es, für Informatik reservierte Zeiten immer zu bevorzugen und leere Zeitslots nur zu verwenden, wenn es nicht anders möglich ist.
4. Priorisierung früherer Zeitslots: Generell sollten frühere Zeiten immer stärker bevorzugt werden als spätere, da es weder im Interesse der Studierenden noch der KursleiterInnen ist, Kurse von Vollzeitstudien am Abend abzuhalten.

6.5.3 Datenstrukturen

Um die Effizienz und Flexibilität des Algorithmus bei der Verarbeitung der Daten zu gewährleisten, wurden verschiedene Datenstrukturen entworfen. Die wichtigsten werden nachfolgend im Detail beschrieben:

AvailabilityMatrix

Die **AvailabilityMatrix** ist eine zentrale Datenstruktur, die entwickelt wurde, um die zeitliche und räumliche Verfügbarkeit von Räumen über eine Woche hinweg abzubilden. Sie ermöglicht eine effiziente Verwaltung von Zeitslots und unterstützt das Zuweisen, Löschen und Überprüfen von Belegungen für Kurse.

Die Hauptidee hinter der Verwendung dieser Matrix besteht darin, die komplexen zeitlichen und räumlichen Constraints des Zuweisungsalgorithmus in eine strukturierte, leicht zugängliche Form zu überführen. So lassen sich Belegungen und freie Zeiten strukturiert darstellen und der Algorithmus kann effizient auf die Verfügbarkeit eines Raums zugreifen, ohne bei jeder Zuweisung alle Constraints erneut prüfen zu müssen. Den Kern der Struktur bildet dabei die in Abbildung 6.2 dargestellte Matrix mit folgenden Eigenschaften:

- $n \times 5$ Matrix (n = Anzahl verfügbare Minuten pro Tag / 15)
 - Jede Spalte repräsentiert dabei einen Wochentag (S1: MON, S2: DIE, ...)
 - Jede Zeile repräsentiert 15 Minuten des Tages (Z1: $[t_0, t_0+15)$, Z2: $[t_0+15, t_0+30)$, ...)
 - Jeder Eintrag in der AvailabilityMatrix eines Raumes ist eine Referenz auf die CourseSession, die diesem Raum zu einer bestimmten Zeit zugeordnet ist.

Neben der Speicherung der Zuweisungsinformationen bietet die AvailabilityMatrix zudem folgende Funktionalitäten:

- Erstellung einer Liste aller potenziellen Zuweiskandidaten für einen Kurs.
- Prüfung von Überschneidungen mit anderen Kursen, sowohl auf Raum- als auch auf Semester-Ebene.
- Dynamische Anpassung der verfügbaren Zeit basierend auf Belegungen oder gelöschten Zuweisungen.

Diese Struktur erlaubt es, sowohl einfache als auch komplexe Szenarien abzubilden, und bildet damit das Rückgrat des Zuweisungsalgorithmus.

	MON	TUE	WED	THU	FRI
08:00 - 08:15 Uhr	C ₁₁	C ₁₂	C ₁₃	C ₁₄	C ₁₅
08:15 - 08:30 Uhr	C ₂₁	C ₂₂	C ₂₃	C ₂₄	C ₂₅
08:30 - 08:45 Uhr	C ₃₁	C ₃₂	C ₃₃	C ₃₄	C ₃₅
08:45 - 09:00 Uhr	C ₄₁	C ₄₂	C ₄₃	C ₄₄	C ₄₅
	▪	▪	▪		
	▪	▪	▪		

Abbildung 6.2: Struktur AvailabilityMatrix

Candidate

Die Klasse **Candidate** repräsentiert einen möglichen Zuweisungskandidaten für einen Kurs. Ein **Candidate** beschreibt dabei eine spezifische Kombination aus Raum, Startzeit und Dauer und dient als zentrale Einheit, die vom Algorithmus in der Zuweisungsphase verwendet wird.

Jeder **Candidate** enthält die folgenden Attribute:

- Verweis auf die **AvailabilityMatrix** des Raumes.
- Tag: Der Wochentag, an dem der Kurs in diesem Raum stattfinden würde.
- Slot Index: Der Zeitslot, der den Startzeitpunkt der Zuweisung beschreibt (z. B. 9:15–9:30).
- Dauer: Die Länge des Kurses in Minuten, die für die Berechnung des Endzeitpunkts verwendet wird.
- Bevorzugung: Ein Boolean-Wert, der angibt, ob der Kandidat einer für Informatik reservierten Zeitspanne entspricht oder nicht.

Die **Candidate**-Klasse enthält darüber hinaus mehrere Methoden, die zum Vergleich zweier Kandidaten während der Zuweisungsphase des Algorithmus herangezogen werden können, z. B. ob sie sich im selben Raum, in der selben Zeitspanne oder am selben Tag befinden.

AssignmentStack

Der **AssignmentStack** ist eine Datenstruktur, die in der Zuweisungsphase des Algorithmus verwendet wird, um alle vorläufig zugewiesenen Kurse zu speichern. Jeder Eintrag im Stack besteht aus einer Kombination eines **CourseSession**-Objekts und dem zugehörigen **Candidate**.

Sie bildet damit eine zentrale Komponente des Backtracking-Mechanismus und stellt sicher, dass der Algorithmus bei Fehlversuchen immer in einen konsistenten Zustand zurückkehren kann.

ConcurrentCourseLimiter

Die Klasse **ConcurrentCourseLimiter** ist dafür verantwortlich, die Anzahl der Kurse zu begrenzen, die sich zeitlich überschneiden dürfen. Diese Funktion ist besonders relevant für Grup-

penkurse (z. B. Proseminare) und Wahlfächer, bei denen es erlaubt ist, dass mehrere Kurse zur gleichen Zeit stattfinden, jedoch nur bis zu einer definierten maximalen Anzahl.

Die `ConcurrentCourseLimiter`-Klasse arbeitet mit einem `Map`-Objekt, welches jedem Kurs eine Liste von Zeitslots zuordnet, die bereits belegt sind. Dabei wird die maximale Anzahl an Überschneidungen durch einen Parameter `numberOfAllowedOverlapsPerCourse` definiert.

Die Hauptfunktionalitäten der Klasse umfassen:

- **`initGroup(T key)`**: Initialisiert einen neuen Eintrag für einen Kurs, wenn dieser zum ersten Mal überprüft wird.
- **`isLimitExceeded(T key, Candidate candidate)`**: Prüft, ob die maximale Anzahl an Überschneidungen für den angegebenen Kurs und Zeitslot bereits erreicht ist.
- **`addEntry(T key, Candidate candidate)`**: Fügt einen neuen Eintrag für einen Kurs hinzu, indem die entsprechenden Zeitslots als belegt markiert werden.
- **`removeEntry(T key, Candidate candidate)`**: Entfernt einen Eintrag, falls der Algorithmus beim Backtracking eine vorherige Zuweisung rückgängig machen muss.

Diese Klasse spielt eine entscheidende Rolle bei der Verarbeitung von Kursen, die in Gruppen oder Modulen organisiert sind, da sie sicherstellt, dass Überschneidungen innerhalb definierter Grenzen bleiben. Dadurch können Gruppenkurse und Wahlfächer effizient zugewiesen werden, ohne die Belegungslogik anderer Kursarten zu beeinträchtigen.

AssignmentState

Die Klasse `AssignmentState` besteht aus einer Map mit `CourseSession` als Key und einer Liste von `Candidate`-Objekten als Value. Zusätzlich enthält jeder `AssignmentState` noch einen Index. `AssignmentState` wird verwendet, um im Rahmen des Backtracking-Ansatzes den jeweiligen Zustand nach einer bestimmten Anzahl von Zuweisungen speichern und wiederherstellen zu können.

6.5.4 Scheduler

Um den Algorithmus besser austauschbar zu machen, wurde ein Interface *Scheduler* mit folgenden Methoden definiert:

- **`setTimeTable(TimeTable): void`**
Diese Methode wird verwendet, um den Scheduler mit dem gerade verwendeten `TimeTable`-Objekt zu initialisieren. Dabei wird für jeden Raum eine `AvailabilityMatrix` erstellt, die später für die Zuweisungsberechnung herangezogen wird.
- **`assignUnassignedCourseSessions(): void`**
Diese Methode ist der Startpunkt für den Zuweisungsalgorithmus. Ziel ist die Zuweisung aller nicht zugewiesenen Kurse des gerade aktiven `TimeTable`-Objekts.
- **`collisionCheck(TimeTable): Map<CourseSession, List<CollisionType>`**
Da der Algorithmus nicht nur für die automatische Zuweisung von Kursen zu Räumen, sondern auch für die Überprüfung manuell zugewiesener Kurse auf Kollisionen verwendet wird, muss jeder Scheduler noch zusätzlich eine Methode `collisionCheck()` bieten. Eine nähere Beschreibung dieser Funktionalität findet sich in Abschnitt 6.5.7.

Dieses Interface bildet die Grundlage für eine modulare und flexible Implementierung des Zuweisungsalgorithmus. Durch die klar definierten Methoden kann der Scheduler leicht erweitert oder durch alternative Implementierungen ersetzt werden, ohne die übergeordnete Logik anzupassen.

6.5.5 Ablauf

In Abbildung 6.3 wird der grobe Ablauf der Zuweisung dargestellt. Nach der Initialisierung folgt eine erste Zuweisungsrunde, in der bereits versucht wird, alle Kurse gemäß der festgelegten Constraints zuzuweisen. Kurse, die in dieser Runde nicht zugewiesen werden können, verbleiben für eine zweite Runde, in der mithilfe von Backtracking alternative Zuweisungen getestet werden. Sollten auch nach dieser Runde noch nicht alle Kurse zugewiesen sein, erfolgt eine dritte Runde, in der die Anforderungen an die Raumtypen gelockert werden: Für Kurse, die keine Computer benötigen, können nun auch Räume mit Computern herangezogen werden, und umgekehrt. Dadurch wird die Flexibilität erhöht, um möglichst viele Kurse erfolgreich zuzuweisen, ohne die grundlegenden Constraints zu verletzen.

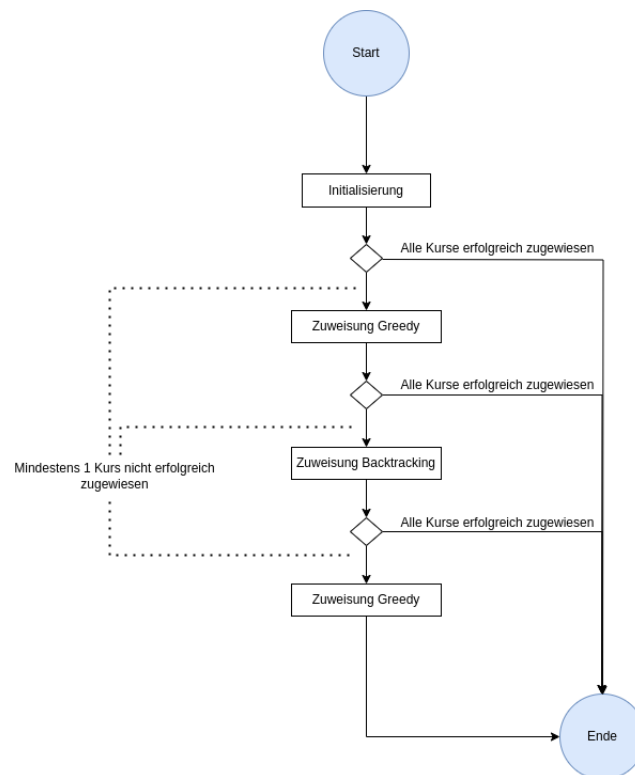


Abbildung 6.3: Überblick Ablauf

In der **Initialisierungsphase** wird für jeden Raum, der dem `TimeTable` zugeordnet ist, eine `AvailabilityMatrix` erstellt. Dabei werden die bereits zugewiesenen Kurse und die Constraints des Raumes eingetragen und einige Vorberechnungen bezüglich der verfügbaren Zeit im Raum vorgenommen.

Die **Zuweisungsphase** lässt sich in drei Teilprozesse unterteilen, wobei sich nur die *Verarbeitungsphase* je nach Ansatz (Greedy oder Backtracking) unterscheidet:

1. *Vorüberprüfungsphase*: In der Vorüberprüfungsphase werden Vorberechnungen durchgeführt, um sicherzustellen, dass eine Zuweisung überhaupt möglich ist. Diese Phase dient dazu, grundlegende Fehler frühzeitig zu erkennen und die Verarbeitungsphase effizienter zu gestalten. Es werden zwei wesentliche Aspekte geprüft:
 - **Zeitverfügbarkeit**: Es wird geprüft, ob die insgesamt benötigte Zeit aller Kurse kleiner oder gleich der insgesamt verfügbaren Zeit aller Räume ist. Dabei werden Räume

entsprechend ihrer Kapazität berücksichtigt, sodass Kurse mit vielen Teilnehmern nur mit ausreichend großen Räumen verglichen werden.

- Raumkapazitäten: Zusätzlich wird überprüft, ob für jede Gruppe von Teilnehmerzahlen genügend Räume mit passender Kapazität vorhanden sind. Hierbei wird sichergestellt, dass sowohl die Mindestkapazität als auch die Obergrenze (Kapazität $\leq 2 \cdot$ Teilnehmeranzahl) eingehalten werden.

Sollte eine der Überprüfungen fehlschlagen, wird der Algorithmus an dieser Stelle abgebrochen, da andernfalls keine sinnvolle Lösung möglich ist.

2. *Kandidatenvorbereitung*: In der Kandidatenvorbereitung werden alle potenziellen Zuweiskandidaten für jeden Kurs berechnet. Dies geschieht mithilfe der Methode `prepareCandidatesForCourseSessions()`, die folgende Schritte ausführt:

- Für jeden Kurs werden alle verfügbaren Räume durchsucht, um Zeitslots zu finden, die dessen Anforderungen (z. B. Raumkapazität, Timing Constraints) erfüllen.
- Daraus wird eine Liste möglicher *Candidate*-Objekte erstellt. Ein *Candidate* repräsentiert dabei eine mögliche Kombination aus Raum, Startzeit und Dauer.
- Die Kandidaten werden nach Präferenzen (z. B. bevorzugte Zeitslots) und anderen Kriterien (z. B. Minimaldistanz zu vorherigen Slots) sortiert. Dabei werden Kandidaten, die stärker mit den Soft Constraints übereinstimmen, priorisiert.

Die Kandidatenvorbereitung bildet die Grundlage für die Verarbeitungsphase, da sie die Menge der möglichen Zuweisungen reduziert und sortiert.

3. *Verarbeitungsphase*: Die Verarbeitungsphase übernimmt die eigentliche Zuweisung der Kurse. Je nach Ansatz wird entweder ohne Backtracking (Greedy) oder mit Backtracking gearbeitet.

Der Ablauf des Greedy-Ansatzes kann stark vereinfacht wie folgt beschrieben werden:

```

1 function processAssignment(map):
2     currentMap = map;
3     currentIndex = 0;
4     failedCourseSessions = empty List
5     while(!currentMap.isEmpty()):
6         currentCourseSession = chooseCourseSessionWithLeastCandidates;
7         if(currentIndex == map.get(currentCourseSession).size()):
8             failedCourseSession.add(currentCourseSession);
9             map.remove(currentCourseSession);
10            currentIndex = 0;
11            continue
12            currentCandidate =
13                map.get(currentCourseSession).get(currentIndex);
14            assignCourseSessionToCandidate(currentCourseSession,
15                currentCandidate);
16            currentMap = filterCandidates(currentMap);
17            if (currentMap has one entry with empty list):
18                unassignLatestCourseSession();
19                index++;
20                continue;
21            finishAssignment();
22            return failedCourseSessions;

```

- (a) **Ausgangspunkt:**
- Eine Map `currentMap`, die alle noch nicht zugewiesenen `CourseSession`-Objekte und deren möglichen Zuweisungskandidaten enthält.
 - Ein Index `currentIndex`, der mit 0 initialisiert wird und angibt, welcher Kandidat als nächstes geprüft wird.
 - Eine Liste `failedCourseSessions`, die zu Beginn leer ist und am Ende alle nicht zugewiesenen Kurse enthält.
- (b) **CourseSession auswählen:** Die `CourseSession` mit den wenigsten möglichen Zuweisungskandidaten wird aus der Map ausgewählt, da Kurse mit wenigen Kandidaten später schwieriger zuzuweisen sind. Dadurch wird die Lösungswahrscheinlichkeit optimiert.
- (c) **Candidate auswählen:** Die Kandidaten einer `CourseSession` werden anhand der *Soft Constraints* (z. B. bevorzugte Zeitslots) sortiert. Der beste Kandidat (gemäß `currentIndex`) wird ausgewählt, solange der Index kleiner ist als die Größe der Kandidatenliste. Wurden alle Kandidaten bereits ausprobiert (`currentIndex == Anzahl der Kandidaten`), wird die `CourseSession` aus der Map entfernt und der Liste `failedCourseSessions` hinzugefügt.
- (d) **Zuweisung:** Die `CourseSession` wird dem Kandidaten zugewiesen, indem beide in den `AssignmentStack` eingetragen und die Zuweisung in der entsprechenden `AvailabilityMatrix` gespeichert werden.
- (e) **Filterung:** Nach jeder Zuweisung wird die Map `currentMap` anhand der *Hard Constraints* gefiltert, wodurch ungültige Kandidaten entfernt werden. Danach werden die verbleibenden Kandidaten erneut anhand der *Soft Constraints* sortiert. Führt die Filterung dazu, dass eine `CourseSession` keine Kandidaten mehr aufweist, wird die letzte Zuweisung rückgängig gemacht (`unassignLatestCourseSession()`) und `currentIndex` um 1 erhöht.
- (f) **Finalisierung:** Die Schritte (b) bis (e) werden so lange wiederholt, bis keine `CourseSessions` mehr in der Map `currentMap` enthalten sind. Die endgültigen Zuweisungen werden persistiert und die Liste `failedCourseSessions` für die nächste Zuweisungsrunde bereitgestellt.

Der Backtracking-Ansatz funktioniert ähnlich wie der Greedy-Ansatz, enthält jedoch zusätzliche Mechanismen, um in den vorherigen Zustand zurückzuspringen, falls keine gültige Zuweisung gefunden wird:

- (a) **Speichern des Zustands:** Vor jeder neuen Zuweisung wird ein `AssignmentState` erstellt. Dieses Objekt speichert die aktuelle Map `currentMap` sowie den Index des zuletzt überprüften Kandidaten. Der `AssignmentState` wird auf einen Stack gepusht, um später darauf zurückgreifen zu können.
- (b) **Rücksprung bei Fehlschlag:** Wenn alle Kandidaten für eine `CourseSession` ausprobiert wurden (`currentIndex >= Anzahl der Kandidaten`), wird kein Eintrag aus `currentMap` entfernt. Stattdessen wird der zuletzt gespeicherte `AssignmentState` aus dem Stack geholt, die letzte Zuweisung rückgängig gemacht, und der `currentIndex` dieses Zustands um 1 erhöht.
- (c) **Vollständige Exploration:** Der Algorithmus führt diesen Prozess so lange fort, bis entweder alle `CourseSessions` erfolgreich zugewiesen wurden oder alle möglichen Kombinationen von Kandidaten durchprobiert sind. Erst dann wird entschieden, ob die Zuweisung vollständig ist beziehungsweise nicht möglich war.

- (d) **Optimale Kandidatenwahl:** Wie beim Greedy-Ansatz wird die CourseSession mit den wenigsten Kandidaten priorisiert, um die Komplexität der Suche zu minimieren. Jedoch werden beim Backtracking zusätzlich alternative Lösungen systematisch geprüft, was die Wahrscheinlichkeit erhöht, eine gültige Zuweisung zu finden.
- (e) **Finalisierung:** Wenn eine gültige Zuweisung gefunden wurde, wird der Algorithmus beendet und die Ergebnisse persistiert. Bei Fehlschlag kehrt der Algorithmus mit einer Liste nicht zugewiesener CourseSessions zurück.

Nach der Verarbeitungsphase werden die zugewiesenen Kurse gespeichert. Nicht zugewiesene Kurse verbleiben in der Liste für die nächste Runde der Zuweisung.

6.5.6 Effizienz und Optimierungen

Die Konzeption und Implementierung des Algorithmus folgte einem iterativen Ansatz, der mehrere Entwicklungsstufen durchlief. Jede dieser Phasen brachte neue Erkenntnisse und führte zu entscheidenden Verbesserungen hinsichtlich Effizienz, Flexibilität und Funktionalität. Es folgt ein kurzer Überblick über die wichtigsten Meilensteine:

- **Erste Version - Einfache Heuristik:** Die erste Version des Algorithmus basierte auf einer heuristischen Vorgehensweise, die ohne Backtracking auskam. In dieser Phase wurde der Fokus auf eine einfache Zuweisung gelegt. Der Algorithmus durchsuchte die verfügbaren Räume sequenziell und wies Kurse direkt zu, sofern keine offensichtlichen Konflikte (z. B. Raumkapazität oder zeitliche Überschneidungen) bestanden. Als Datenstruktur wurde neben AvailabilityMatrix noch eine Queue mit Kandidaten verwendet, die teilweise zufällig bestimmt wurden. Diese erste Version konnte nur triviale Szenarien lösen und scheiterte in Fällen, in denen mehrere konkurrierende Anforderungen erfüllt werden mussten.
- **Zweite Version - Einführung des Backtracking:** Die zweite Version markierte einen entscheidenden Fortschritt durch die Einführung eines Backtracking-Mechanismus. Der Algorithmus konnte nun Lösungen durch systematisches Zurückgehen und erneutes Probieren finden, wenn direkte Zuweisungen nicht möglich waren. Neben der Raumkapazität wurden erstmals zusätzliche Randbedingungen wie bevorzugte Räume und zeitliche Präferenzen berücksichtigt. Die Verwendung von PriorityQueues erlaubte es, vielversprechende Kandidaten zuerst zu berücksichtigen und weniger geeignete Optionen zurückzustellen. Einer der größten Nachteile des reinen Backtracking-Ansatzes war allerdings die mangelnde Flexibilität bei bestimmten Randbedingungen sowie der extreme Zeitaufwand bei realistischen Szenarien mit sehr knappen Zeitbudgets.
- **Finale Version - Kombination aus Greedy- und Backtracking-Ansatz:** Um eine optimale Balance zwischen Effizienz und Genauigkeit zu erreichen, wurde der Backtracking-Ansatz wieder etwas entschärft und um einen Greedy-Ansatz erweitert. Zusätzlich wurde mehr Fokus auf eine komplexe Filter- und Sortierlogik gesetzt, um auch ohne Backtracking bereits die bestmögliche Verarbeitungsreihenfolge der Kurse sowie der Zuweisungskandidaten zu erhalten und so den Berechnungsaufwand zu minimieren. Die Einführung des ConcurrentCourseLimiters erlaubt es zusätzlich, Ausnahmen richtig zu behandeln und die Verwendung des Interfaces Scheduler erhöht die Modularität und Erweiterbarkeit.

Das Ergebnis ist ein Algorithmus, der nicht nur robust und effizient ist, sondern auch eine solide Grundlage für zukünftige Erweiterungen bietet.

6.5.7 Erweiterungen

Neben der automatischen Zuweisung aller nicht zugewiesenen Kurse eines `TimeTable`-Objekts gibt es noch zwei weitere Funktionalitäten, für die Teile des Algorithmus in abgewandelter Form zum Einsatz kommen:

- **Collision Check:** Der Collision Check ermöglicht es, bestehende Zuweisungen im `TimeTable` auf Kollisionen zu überprüfen. Dabei werden vier Arten von Konflikten identifiziert:
 - **Raumkapazität:** Ein Kurs überschreitet die maximale Teilnehmerzahl, die der zugewiesene Raum aufnehmen kann.
 - **Computerverfügbarkeit:** Ein Kurs, der Computer benötigt, wurde einem Raum ohne Computer zugewiesen, oder umgekehrt.
 - **Timing Constraints:** Die Zuweisung des Kurses verletzt zeitliche Einschränkungen, die für den Raum oder den Kurs definiert wurden.
 - **Semesterüberschneidung:** Kurse desselben Semesters überschneiden sich zeitlich, was für Studierende unzulässig ist.

Der Collision Check generiert eine Map, die für jeden kollidierenden Kurs eine Liste der aufgetretenen Konflikttypen (`CollisionType`) enthält. Diese Funktion wird insbesondere genutzt, um manuell erstellte Zuweisungen auf Fehler zu prüfen.

- **Semi-Automatic Assignment:** Diese Funktion erlaubt es NutzerInnen, die Zuweisung von Kursen teilweise selbst zu steuern. Der Algorithmus stellt eine Liste aller noch nicht zugewiesenen Kurse sowie deren mögliche Zuweiskandidaten bereit. UserInnen können dann zwischen zwei Optionen wählen:
 1. **Manuelle Zuweisung:** NutzerInnen wählen für einen Kurs aus der Liste einen spezifischen Kandidaten aus und weisen diesen zu.
 2. **Teilweise automatische Zuweisung:** NutzerInnen markieren mehrere Kurse, die anschließend durch den Algorithmus automatisch zugewiesen werden.

Nach jeder Zuweisung, ob manuell oder automatisch, wird die Liste der möglichen Kandidaten für alle verbleibenden Kurse aktualisiert. Dadurch wird sichergestellt, dass die Constraints weiterhin eingehalten werden. Ein möglicher Workflow könnte so aussehen:

1. Von 10 nicht zugewiesenen Kursen weist ein/eine UserIn einen Kurs manuell einem Kandidaten zu.
2. Für 5 weitere Kurse übergibt er/sie die Zuweisung dem Algorithmus.
3. Die verbleibenden 4 Kurse werden dann erneut nacheinander manuell bearbeitet.

Diese Flexibilität ermöglicht es, individuelle Präferenzen der NutzerInnen mit der Effizienz des Algorithmus zu kombinieren.

Zusammen mit diesen Erweiterungen bietet die Anwendung ein breites Spektrum an Möglichkeiten, die Kurszuweisung nach eigenem Belieben zu steuern, um so zum bestmöglichen Ergebnis zu gelangen.

7 Frontend

In diesem Abschnitt wird die Struktur und Funktionsweise der Frontend Umgebung beschrieben. Beginnend mit einer Erklärung des Frameworks Angular und seinen Konzepten. Um den Aufbau und Ablauf besser nachvollziehen zu können, beschreibt Kapitel 7.5 (Models) die wichtigsten Modelle, etwaige Anwendungen und Zusammenhänge.

Mit dem Kapitel 5(Workflow) wird der Arbeitsprozess aus Sicht des/der BenutzerIn gezeigt. Ebenfalls werden anhand von eingebauten Beispielen angewandte UI/UX Prinzipien gezeigt. Anstelle einer bisher Benutzer Freundlichen Betrachtung behandeln die letzten vier Kapitel die Technischen Details der jeweiligen Komponenten.

7.1 Angular

Entwickelt von Google als Open Source Software, wurde der Vorgänger AngularJS erstmals auf der ng-Europe-Konferenz im Jahr 2014 angekündigt und im Jahr 2016 veröffentlicht. Um der stetigen Entwicklung des Internets mit zu halten, wurden Jahr für Jahr neue Versionen entwickelt. Einer der größten Meilensteine wurde im Jahr 2018 erreicht. Mit Version 7 änderte sich die Programmiersprache von Javascript auf die Sprache Typescript. Im wesentlichen unterscheiden sich die Sprachen in ihrer Typisierung. Während Javascript eine dynamischen Typisierung erlaubt, schränkt Typescript jene auf eine statische Typisierung ein.

Kern Prinzipien:

- **Komposition mit Komponenten:** Um die Entstehung großer Klassen zu vermeiden, bietet Angular die Möglichkeit, Komponenten zu erstellen. Bestehend aus je einer Logik-, Style- und Design-Datei bildet eine Komponente einen Teil der Website ab. Ebenfalls können Komponenten beliebig oft verschachtelt werden, was eine einheitliche Strukturierung von Code und Logik unterstützt. [components]
- **Reactives Design mit Signalen:** Signale bilden eine leichtgewichtige Ummantelung für Variablen. Ergänzend werden bei dem 'Get', 'Set' und 'Update' Module außerhalb der Coding Logik angesprochen. Beispielsweise wird bei einer Änderung des Signal Wertes der DOM (Data Object Model) Eintrag geändert. [signals]
- **Dynamische Interfaces:** Bei einer Referenz des Frontends auf die Angular Logik, wird automatisiert eine Verbindung mit generiert. Anhand dieser Verbindung, werden Änderungen, in der jeweils anderen Komponente mit geändert. Wird ein Zähler auf der Website erhöht, wird der Wert persistent in der Logik gespeichert. Während Signale manuell gesetzt werden, bestehen Dynamische Interfaces automatisch. [interfaces]

Der Kern der Software setzt sich aus vier Komponenten zusammen. In Abbildung 7.1 werden die Zusammenhänge der jeweiligen Komponenten dargestellt. Insgesamt wurden im Frontend 27 Komponenten, 5 Converter und 11 Services eigens erstellt. Hinzu kommen 47 Komponenten und

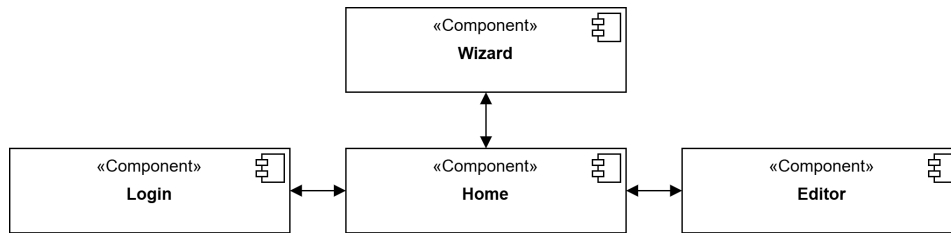


Abbildung 7.1: Auth Guard Admin Sequenzdiagramm

2 weitere Services aus PrimeNG, Fullcalendar und Angular. Das Projekt wurde anhand mehrere Module gegliedert.

7.2 struktureller Aufbau

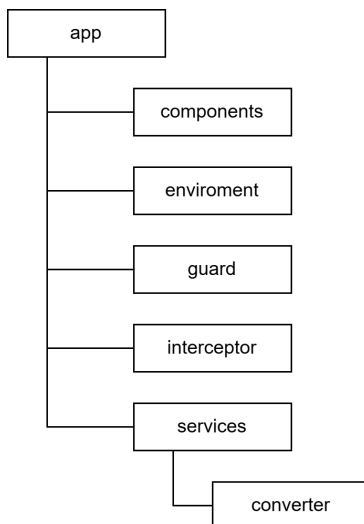


Abbildung 7.2: Ordner Struktur

Basierend auf den unterschiedlichen Befehlen, welche 'ng' zur Verfügung stellt, wurde die Order Struktur 7.2 gewählt. In diesem Projekt wurde zwischen zwei Umgebungen (**enviroment** im Diagramm) unterschieden. Während dem Entwicklungsprozess, wird mit der DEV-Umgebung gearbeitet. Diese unterscheidet sich zur PROD-Umgebung im wesentlichen durch eine Variable. Im DEV-Mode wird der Wert baseUrl auf '/proxy' gesetzt. Damit wird dem Proxy, welcher nur im DEV-Mode inkludiert ist, mitgeteilt, dass es diese Anfragen auf die bei ihm hinterlegte URL weiterleitet. Der PROD-Mode besitzt keinen Proxy weshalb die baseUrl direkt auf die URL des Backends gesetzt wird.

Guard beinhaltet eine Reihe an Funktionen, welche vor dem Aufrufen einer neuen bzw. verlassen einer alten Seite aufgerufen werden. Beispielsweise gibt es den *AuthGuardEditorClose*, welche überprüfen ob nicht gespeicherte Daten im Editor vorliegen. In diesem Fall erscheint ein Popup vor dem umleiten auf eine neue Seite, welches den/die NutzerIn auf seine/ihre ungesicherten Daten hinweist.

Um die Benutzererfahrung (UX) nachvollziehbarer zu gestalten, wurde eine Spinner Komponente gebaut. Diese zeigt einen sich drehenden Kreis welcher den Ladezyklus darstellen sollte. Beim Laden großer Daten kann es zu längeren Wartezeiten kommen. Um die Aufmerksamkeit der BenutzerInnen zu behalten, wurde der Ladebalken eingeführt. Der **Interceptor** überwacht die Anzahl an ausstehenden Anfragen. Ist die Anzahl größer als 0 beziehungsweise es wird auf Daten gewartet, setzt der Interceptor sein *loading* Attribut auf *true*. Ändert sich der Status des Attributs, wird die Komponente angezeigt. Geht der Zähler zurück auf 0, verschwindet die Komponente erneut.

7.3 Components

Basierend auf dem Komponentenprinzip von Angular, wurde das Projekt in unterschiedliche Komponenten aufgeteilt, wobei Jede aus einer Webiste (.html), Styling (.css) und Logik (.ts) Datei besteht. Im Diagramm 7.3 werden alle erstellten Komponenten gezeigt. Nicht aufgelistet sind aus PrimeNG bzw. Fullcalendar importierte Komponenten.

Der Arbeitsprozess sieht vor, dass für jedes Semester ein neuer TimeTable erstellt wird. Um

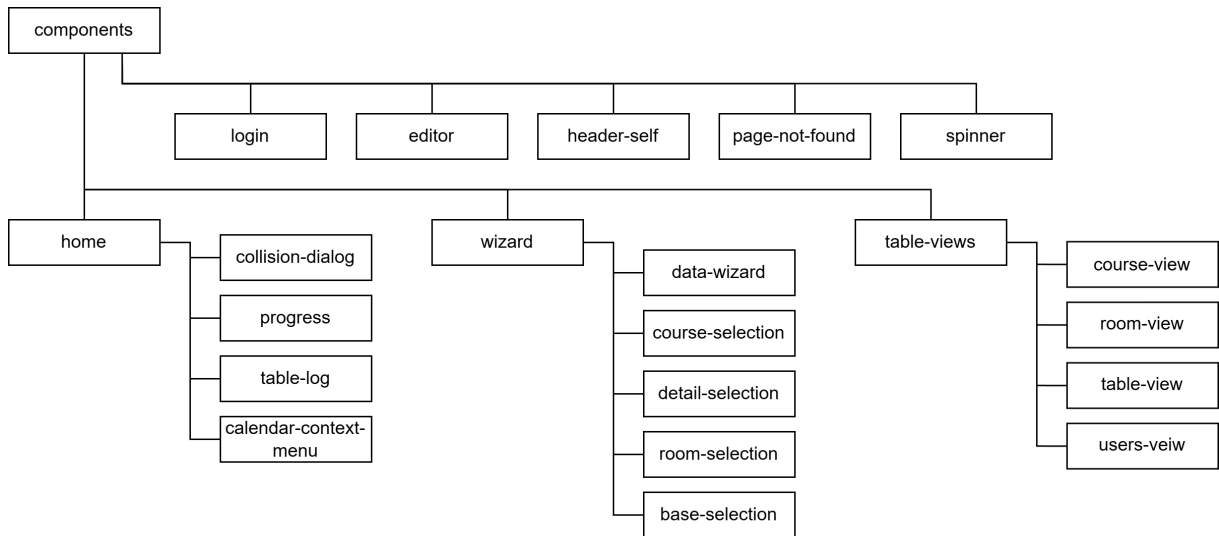


Abbildung 7.3: Komponentendiagramm

veraltete und nicht mehr gebrauchte Daten zu bereinigen, wurde die **table-view** Komponente erstellt. Öffnet man die Seite, wird eine Tabelle aus allen bestehenden TimeTable angezeigt. Beim Aufklappen einer Reihe, wird die jeweilige Liste an RoomTables und CourseSessions geladen. Die Ansicht gibt spezifische Informationen bezüglich der darin enthaltenen Daten. Um den Verlust von wichtigen Daten zu gewährleisten, erscheint dem Benutzer beim Löschen eines TimeTable ein Popup, welches eine erneute Bestätigung anfordert.

Je nach Rolle des/der BenutzerIn, werden im Header Menü (**header-self**) unterschiedliche Optionen angezeigt. Aus Sicherheitsgründen ist es nur BenutzerInnen mit der Administrator Rolle erlaubt, Benutzer, TimeTables zu verwalten. Um den Zugriff zu verhindern, werden Admin Optionen für Default BenutzerInnen ausgeblendet. Versucht ein/e BenutzerIn ohne Administrator Rolle über den Link die jeweilige Admin Oberfläche aufzurufen, wird er von dem *AuthGuardAdmin* blockiert.

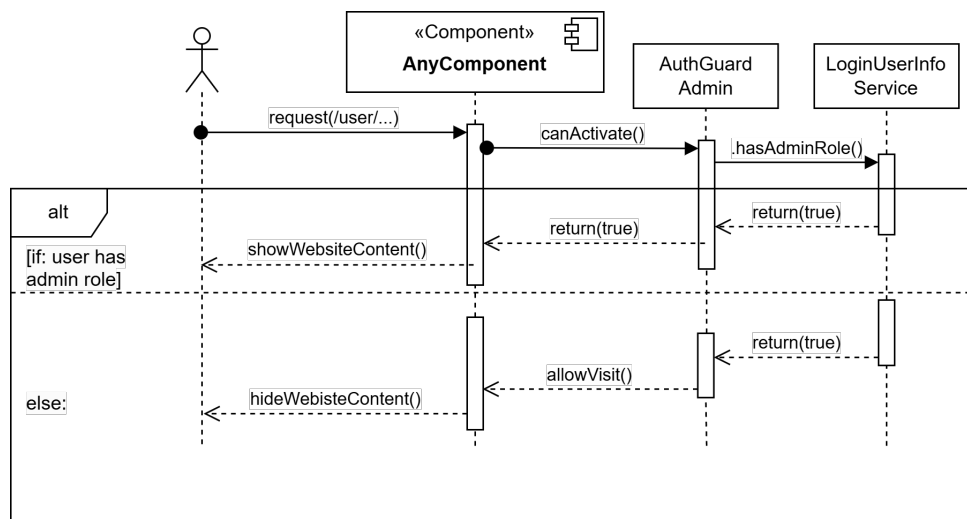


Abbildung 7.4: Auth Guard Admin Sequenzdiagramm

Die Abbildung 7.4 stellt den Ablauf beim Zugriff auf eine neue Admin Seite dar. Bei der Anfrage auf eine beliebige Admin URL, wird die *CanActivate* Methode abgefragt. *CanActivate* gibt einen

Boolean Wert zurück.

Bei einem positiven Wert (true) wird die Anzeige geladen. Ist der Wert hingegen negativ (false), erhält der/die BenutzerIn eine leere Seite, da ihm/ihr die Berechtigungen fehlen. AuthGuardAdmin überprüft anhand der `.hasAdminRole()` Methode, welche im `LoginUserService` enthalten ist, ob der/die entsprechende BenutzerIn die Berechtigungen hat.

Als Standard für die meisten Websites gibt es auch bei LeCo eine Page-not-found Seite. Diese ist in der gleichnamigen **page-not-found** Komponente verbaut. Diese Seite weist den/die BenutzerIn auf eine nicht vorhandene Seite hin und gibt ihm/ihr die Möglichkeit auf die Login Seite zurück zu kehren. Die über bleibenden Komponenten, Login, Home, Wizard und Editor, werden in den Kapiteln 7.6 bis 7.9 beschrieben.

Lebenszyklus einer Komponente

In Angular beschreibt der Component Lifecycle die Phasen, die eine Komponente von ihrer Erstellung bis zu ihrer Zerstörung durchläuft. Entwickler können über spezielle Lifecycle-Hooks auf Ereignisse in diesen Phasen reagieren. Diese Hooks werden als Methoden in bestimmten Interfaces bereitgestellt, die Angular definiert.

Bei der Initialisierung einer Komponente können Änderungen an den Eingabe-Eigenschaften (@Input) auftreten. Diese werden durch das `OnChanges` Interface und dessen Methode `.ngOnChanges()` behandelt, die aufgerufen wird, wenn Eingabewerte geändert werden. Dies ermöglicht es der Komponente, direkt auf Änderungen zu reagieren, noch bevor andere Initialisierungsphasen erreicht werden. Vor allem bei leichtgewichtigen Komponenten wie diverse Popups ist dies öfter der Fall. Bewegt ein/eine BenutzerIn den Maus Zeiger über mehrere Hover Felder, kommt es oft vor, dass ein neuer Datensatz geladen wird, bevor der alte überhaupt angezeigt wurde.

Nach Abschluss des Konstruktors wird die Methode `.ngOnInit()` des `OnInit` Interfaces ausgeführt. Dies markiert den Punkt, an dem die Komponente vollständig initialisiert ist und die eigentliche Ansicht noch nicht gerendert wurde. Spezifisch für die Vorbereitung der Darstellung von Komponenten ist dieser Hook ideal. Etwa um zusammenhängende Daten zu laden oder andere Initialisierungen vorzunehmen. Beispielsweise lädt die Home Komponente die Auswahl der Funktionen, wie 'Apply Algorithm', in der `ngOnInit()` Methode anstelle des Konstruktors. Die Auswahl Optionen werden erst bei der Render Phase benötigt.

Während der Laufzeit können über die Methode `.ngDoCheck()` des `DoCheck`-Interfaces Änderungen erfasst werden. Mechanismen zur Überprüfung der Änderungen sind besonders nützlich, für eine performantere Applikation. Die Logik der Erkennung von Änderungen und das aktualisieren der Seite sind getrennt. Daher wird meist in diesem Block die Aktualisierung der Seite manuell gestartet anstelle auf die automatische Überprüfung zu warten.

Ähnlich funktioniert die Überwachung der Ansicht einer Komponente. Das `AfterViewInit`-Interface und seine Methode `.ngAfterViewInit()` werden ausgeführt, nachdem die Ansicht der Komponente und die ihrer Kind Komponenten vollständig initialisiert sind. Dies ist der ideale Zeitpunkt, um DOM-Manipulationen oder View-Child-Abfragen durchzuführen.

Wird eine Komponente nicht mehr benötigt, wird der von der Komponente selbst belegte Speicher wieder frei gegeben. Vor der endgültigen Entfernung einer Komponente, wird der Hook `.ngOnDestroy()` des `OnDestroy`-Interfaces aufgerufen. Spezifische Aufräumarbeiten, wie das Abmelden von Observables oder das Entfernen von Event-Listnern, werden in diesem Schritt erledigt. Dies dient dazu, um Speicherlecks zu vermeiden und somit die Performance dauerhaft aufrecht zu erhalten. [lifecycle]

7.4 Services und Converter

Services haben gegenüber Komponenten den Vorteil, dass sie dem Singleton Pattern entsprechen. Daher gibt es für jeden Service nur eine Instanz innerhalb der gesamten Anwendung. Diese Instanz ermöglicht es, Daten konsistent zu speichern und zwischen verschiedenen Komponenten zu teilen. Dadurch können redundante Kopien der Daten vermieden werden. Durch die Nutzung von Services können Zustands- und Geschäfts Logik unabhängig von den Komponenten verwaltet werden, wodurch die Wiederverwendbarkeit und Wartbarkeit des Codes erheblich verbessert wird. Zudem fördern Services die Entkopplung von Komponenten, da diese lediglich auf den Service zugreifen müssen, ohne direkt miteinander zu kommunizieren. In diesem Abschnitt wird näher auf die wichtigsten Services eingegangen, einschließlich ihrer Implementierung.

7.4.1 Converter

Converter sind ebenfalls vom Service Typ, unterscheiden sich jedoch in ihrer Funktionalität. Ein Converter erhält als Eingabe eine Instanz des Datentyp A und wandelt ihn in eine Instanz des Typs B um. Diese Beziehung ist in einigen Beispielen Bidirektional. Beispielsweise gibt es einen Int to Float Converter, welcher Integer (Gleitkommazahlen) in Floats (Fließkommazahlen) umwandeln kann. Dadurch ist es problemlos möglich eine Integer 4 als eine Float 4 darzustellen. Versuchen wir die Implementierung umzukehren, stoßen wir auf Probleme. Wird der Wert von 4 auf 3.6 reduziert, kann Dieser nicht ohne Datenverlust in einen Integer konvertiert werden.

Vier der Fünf Converter erfüllen eine konkrete Funktionalität. Jeder Converter implementiert das *DtoConverter* Interface, welches die Methoden *toDTO()* und *fromDTO()* beinhaltet. Damit entfällt die Logik, um Eigenschaften welche lediglich im Frontend benötigt werden, zu speichern. Ein Beispiel dafür ist die Farbe eines spezifischen Kurses im Kalender. Der Wert ist standardmäßig im Frontend hinterlegt, da er für jeden Kurs der selbe ist.

Der *EventConverter* Service verbaut im Vergleich zu den anderen Converter wesentlich mehr Logik. Anstelle der Umwandlung äquivalenter Modelle, werden *CoursesSession* in *EventInput* Objekte umgewandelt. *EventInput* ist eine in der Fullcalendar Bibliothek vordefinierter Klasse, welche ein Event am Kalender darstellt. Mit einem Fragezeichen dargestellte Variablen der Abbildung 7.5, sind optional. Im Entwicklungsprozess wurde die *CourseSession* definiert bevor die erste Kalender Implementierung eingebaut wurde.

Trotz Unterschiede besitzen die Datentypen ähnliche Attribute wie der ID und dem title-Attribut welches dem Session Namen ähnelt. Weitere Attribute wie *startTime* und *endTime* können aus dem Timing Attribut der *CourseSession* berechnet werden.

Um nur zugewiesene Kurse umzuwandeln, wird die Auswahl an *CourseSession* bereits in der Editor beziehungsweise Home Komponente gefiltert. Je nachdem welche Komponente die Methode beeinflusst es die Werte der *editable* und *backgroundColor* Attribute.

Kursen ist es ausschließlich im Editor erlaubt, verschoben zu werden, daher wird das *editable* Attribut bei der Home Komponente auf *false* gesetzt. Kurse haben die Eigenschaft, fixiert zu werden. Dadurch können sie weder von NutzerInnen, noch vom Algorithmus verschoben werden. Um diese Eigenschaft (*backgroundColor* in der Abbildung [7.5]) bildlich dar zu stellen, werden fixierte Kurse in einem dunklerem Rot Ton (#7a4444) ■ eingefärbt.

«class» EventInput
+ id?: string
+ title?: string
+ description?: string
+ daysOfWeek?: string[]
+ startTime?: string
+ endTime?: string
+ backgroundColor?: string
+ display?: string
+ editable?: boolean
+ durationEditable?: boolean
+ droppable?: boolean
+ extendendProps?: any

Abbildung 7.5: Ordner Struktur

Um eigens definierte Werte im Event zu speichern, gibt es das *extendedProps* Attribut vom Datentyp *any*. Hilfreiche Eigenschaften, wie der Kurstyp, Semester, Studien Grad des Kurses, und die Nummer an Teilnehmern können dort definiert werden. Dies vereinfacht das Filtern da die Suche nach Sub-Strings entfällt.

Ebendalls erfüllt der EventConverter die Aufgabe, Background Events zu erstellen, welche lediglich im Editor Modus benötigt werden. Background Events unterscheiden sich in ihrer Ansicht, da sie eine visuelle Transparenz aufweisen. Ist ein Event transparent grau eingefärbt, wird der Block als *BLOCKED* definiert. Obwohl in diesem Bereich keine Kurse platziert werden sollten, ist es dennoch möglich. Ausschließlich im Wizard ist es möglich Background Events zu erstellen, allerdings fehlt die Bearbeitungsansicht. Um die Transparenz des Des Events darzustellen, wird das *display?: string* Attribut auf 'Background' gesetzt.

7.4.2 Data Transfer Object (DTO) Pattern

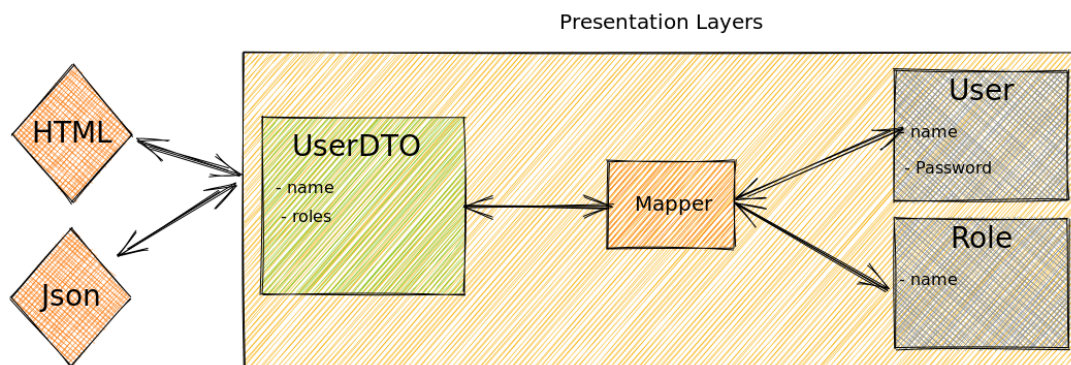


Abbildung 7.6: Data Transfer Object Pattern, Quelle: Dto Pattern Website

Das Data Transfer Object (DTO) wurde eingeführt, um die Anzahl an Methodenaufrufen zu minimieren. Erstmals wurde das Pattern von Martin Fowler in seinem Buch EAA [Fowler (2012)] vorgestellt. Laut Fowler besteht das Ziel darin, die Anzahl der Anfragen an den Server zu reduzieren, indem mehrere Parameter in einem einzigen Aufruf gebündelt werden. Nehmen wir als Basis die Darstellung 7.6. Ohne den jeweiligen Mapper, welcher die Blöcke User und Role miteinander verbindet, müssen zwei anstelle von einer Liste übertragen werden. Effektiv beträgt die Anzahl an Methoden Aufrufen dabei zwei mal die Anzahl der BenutzerInnen. Wird der Mapper schon im Backend anstelle des Frontend angewendet, reduzieren wir die Anzahl der Methoden Aufrufe um die Hälfte.

Ein weiterer Vorteil ist die Kapselung der Serialisierungslogik. Als Serialisierung bezeichnet man den Mechanismus, der die Objekt und Datenstruktur in ein bestimmtes Format übersetzt, welches gespeichert und übertragen werden kann. Dadurch entsteht ein zentraler Punkt für Änderungen an den Feinheiten der Serialisierung. Zudem entkoppelt es die Daten-Modelle von der Präsentationsschicht, sodass beide unabhängig voneinander geändert werden können. [Dto Pattern Website]

7.4.3 Services

Abbildung 7.7 zeigt das Klassendiagramm der Kern Komponenten und die Verwendung der jeweiligen Services, welche in bestimmte Kriterien gruppiert werden können.

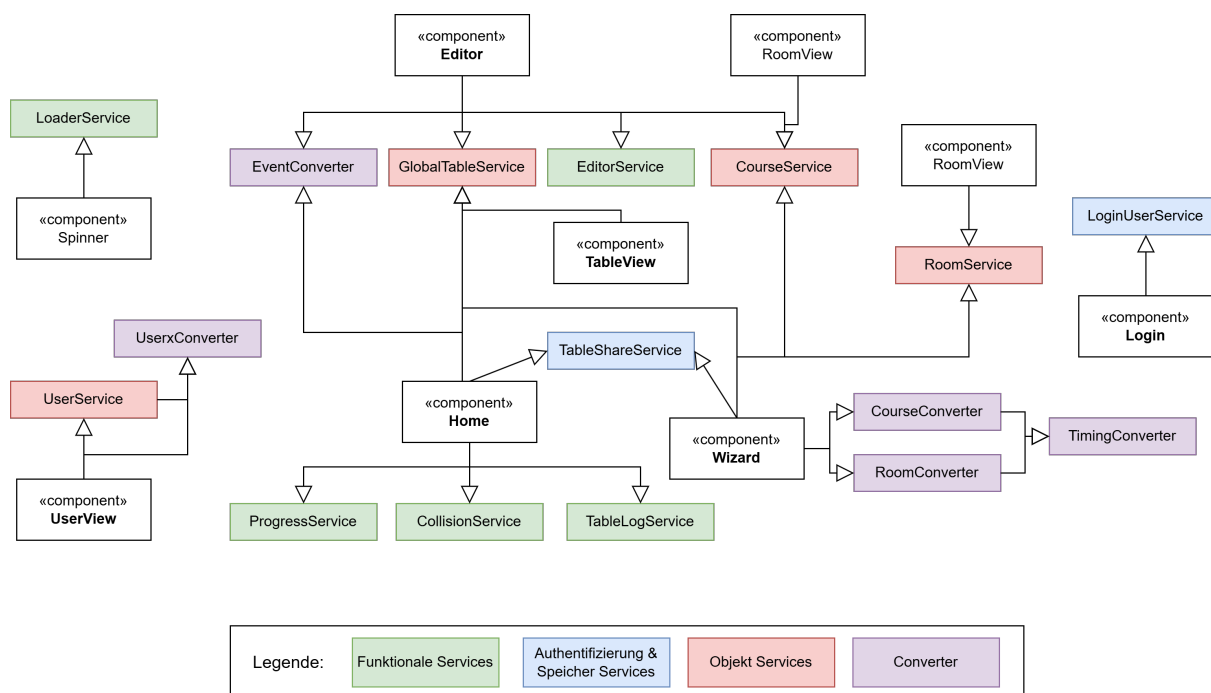


Abbildung 7.7: Klassendiagramm Services und Converter

Objekt Services

Eine Gruppe stellen die Objekt Services, Rot in der Abbildung 7.7, wie *CourseService*, *RoomService*, *TableViewService*, *UserService* und *GlobalTableService* dar. Diese bieten Methoden um den Kontakt zwischen Frontend und Backend aufzubauen. In jeder Service Klasse sind die jeweiligen API Schnittstellen, hinterlegt.

Um mittels HTTP zu kommunizieren, benötigt jeder Services die Instanz des Router Moduls. Dieses stellt generische Methoden für alltägliche HTTP Befehle zur Verfügung. Dazu gehören, *GET*, *POST*, *PUT* und *DELETE*. Anhand der ausgewählten Methode wird die zu ausführende Operation definiert. Wird ein Aufruf als HTTP GET gestellt, geht ein Objekt als Retour Wert ein. Beispielsweise nutzt RoomService die API Schnittstelle `'environment.baseUrl/api/rooms'` um die Liste an allen existierenden Räumen vom Backend zu laden.

POST hingegen bietet die Möglichkeit ein Objekt mit zu schicken. Wird ein im Wizard erstellter **TmpTimeTable** an das Backend gesendet, wird ein HTTP POST Aufruf mit dem TmpTimeTable als Objekt gesendet. Als Antwort erhalten wir den konvertierten TimeTable.

Ähnliche wie bei POST, kann mittels einem HTTP PUT ein Objekt mitgeschickt werden. Während bei einem POST Aufruf der erwartete Datentyp des Antwort Objektes sich vom Sender Objekt unterscheidet, wird bei einem PUT Aufruf auf den selben aktualisieren Datentyp gewartet. Anwendung findet die Funktion beispielsweise beim Aktualisieren von Kurs Daten. Farblich markiert in der Abbildung 7.7 wird diese Gruppe in Blau dargestellt. Angular bietet mehrere Möglichkeiten um Komponenten übergreifend Daten auszutauschen. Verweisen die Komponenten auf einander, gibt es vorab implementierte (dynamische) Interfaces.

Verschachtelte Komponenten werden selten in der Logikdatei (.ts) referenziert, weshalb die Referenzen aus dem DOM (Data Object Model) gezogen werden müssen. Die Verschachtelung ist meistens in der Struktur Datei (.html) angegeben, da jede Komponente einem eigenen Webiste Block entspricht. Benötigt man einen Wert der Parent-Komponente, wird in der Child-Komponente eine @Input() Variable definiert. Dadurch kann im HTML Tag die jeweilige Vari-

ble übergeben werden. Anstelle der Referenz, wird eine Kopie des Objektes übergeben. Näheres dazu ist im Kapitel 7.8 beschrieben. Services kommen dann zum Einsatz, wenn kein direkter Zusammenhang zweier Komponenten besteht.

Speicher und Authentifizierung Services

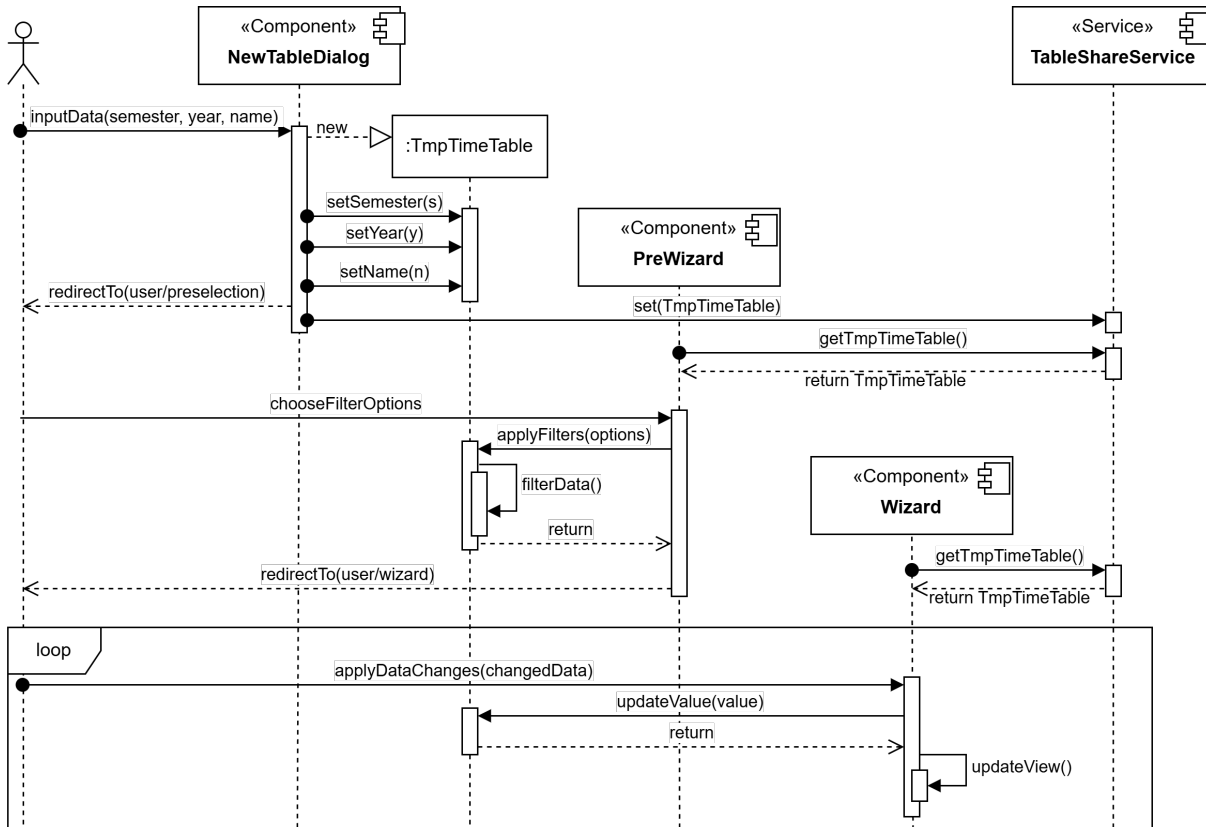


Abbildung 7.8: TableShareService Sequenz Diagramm

Die Anwendung eines Share-Service wird in der Abbildung 7.7 anhand des *TableShareService* dargestellt. Beim öffnen des *NewTableDialog* erschienen Auswahl Felder für die Eingabe des Semesters, des Jahres und ein Textfeld für Namen. Wählen BenutzerInnen die 'Create' Option, wird anhand der eingegebenen Daten ein neues *TmpTimeTable* Objekt erstellt und dem *TableShareService* übergeben. Nach der Umleitung auf die Pre-Wizard Ansicht, erhalten BenutzerInnen die Option, diverse Filter und Automatisierungen auszuwählen. Beendet die NutzerInnen die Auswahl, wird der im *TableShareService* gespeicherte Table auf die Filter Optionen angepasst. Abschließend können die Daten manuell angepasst werden. Mit jeder Änderung, wird ebenfalls das Objekt im *TableShareService* angepasst. Wird ein Kurs, in der Kurs Auswahl Ansicht, aus der ausgewählten Liste entfernt, muss sowohl der Table als auch die Ansicht angepasst werden. Während der *TableShareService* dem Speichern von Daten dient, wird sämtliche Authentifizierung von dem *LoginUserInfo* Service übernommen. Sendet ein/eine BenutzerIn seine/ihre Authentifizierungsdaten an das Backend, wird der *LocalUserInfo* Service aufgerufen. Sind die Eingabedaten korrekt, erhält der/die UserIn einen Login Token, welcher im sessionStorage gespeichert wird. Ebenfalls bietet der Service die Möglichkeit, die Rollen von BenutzerInnen zu überprüfen, welche in vielen Komponenten Gebrauch findet. Um die Anschaulichkeit der Abbildung 7.7 zu erhalten wurden dieses Verbindungen nicht eingezeichnet.

Funktionale Services

Dargestellt wird die Funktionale Gruppe in Grün in der Abbildung 7.7. Für Services gibt es keine Webiste (.html) Datei. Daher werden Services oft in Kombination mit Komponenten erstellt. Während der Service sämtliche Logik beinhaltet, stellt die Komponente den visuellen Teil dar. Genauso kann ein Service als ausgelagerte Logik agieren, was dem Single Responsibility-Pattern laut den SOLID Prinzipien entspricht. Gerade in der Home Komponente gibt dank der Features sehr viel unterschiedliche Logik ohne näheren Zusammenhang. Beispielsweise hat die Überprüfung auf Kollisionen keinen Zusammenhang mit dem exportieren von Kalender Daten. Das Collision-Modul besteht aus dem CollisionService und der CollisionDialog Komponente.

Wird der 'Collision Check' Button gedrückt, sendet der Collision Service eine Anfrage an Backend mit der Id des jeweiligen Tables. Sofern Kollisionen vorhanden sind, erhalten BenutzerInnen eine Liste an Kollisionen. Daten als JSON verpackt sind meist unübersichtlich weshalb anhand dem Collision Dialog eine vereinfachte Ansicht geboten wird.

Erscheint, aufgrund fehlender Kollisionen, eine leere Ansicht erhalten UserInnen weder Feedback über etwaige Kollisionen, noch ob die Daten überhaupt geladen wurden. Daher informiert der CollisionService BenutzerInnen über den aus PrimeNG importierten MessageService, in Form einer kleinen Popup Nachricht.

Um NutzerInnen mehr Feedback bezüglich der Datenübertragung zu geben, wurde der Loader-Service entwickelt. Wird eine Anfrage mittels HTTP an eine beliebige Adresse gestellt, wird diese im Interceptor mitverfolgt. Solange auf Pakete gewartet wird, setzt der Interceptor das Loading Attribut des Loader Service auf true. Sind alle Pakete eingetroffen, ändert sich der Wert auf false. Je nach loading Status wird im Frontend das Spinner Element angezeigt.

Ähnlich zu dem Loading Service gibt es den Progress Service. Anstelle eines sich drehenden Kreises, gibt die Progress Bar mehr Informationen bezüglich dem Fortschritt zurück. Die Auftrennung basiert auf den jeweiligen Wartezeiten. Während die Spinner Komponente um die 3 Sekunden angezeigt wird, kann die Progressbar bis zu 20 Sekunden anhalten. Gezeigt wird die Komponente mit der 'Print Current (Rooms)' Methode. Die Screenshots und Berechnungen zur Generierung des PDF Dokuments können bei Verwendung aller Räume bis zu 30 Sekunden andauern.

Intern ist der Progress Service in Kombination mit der Progress Komponente ähnlich aufgebaut wie der Loader Service. Anstelle des Interceptors, setzt die 'Print Current (Rooms)' Methode den loading Wert auf true bei Beginn beziehungsweise false beim Beenden.

Eine der Anforderungen war es, die Änderungen eines Tables nachvollziehbar anzuzeigen. Mit jedem speichern des Tables, wird im Backend ein Vergleich für jede CourseSession gemacht. Wird beispielsweise ein Kurs von 12:00 Uhr Dienstags auf 15:00 Uhr Donnerstags verschoben, erkennt das Backend die Änderung und erstellt ein neues GlobalTableChange Objekt. Dabei wird immer nur die letzte Kurs Änderung gespeichert. Zieht eine/eine BenutzerIn den Kurs von Donnerstags 12:00 Uhr auf Freitag 08:00 Uhr ohne zwischen zu Speichern, wird die Änderung auf 12:00 Uhr nicht gespeichert.

Die Darstellung der Änderungen wird in der TableLog Komponente angezeigt. Diese verwendet den TableLogService um die Liste an Änderungen zu erhalten. Gleich wie beim CollisionDialog, wurden nur dann Daten angezeigt, wenn es bereits Änderungen gegeben hat. Aufgrund fehlender Funktionalitäten dient die Ansicht als Information. Daher implementiert der TableLog Service lediglich eine HTTP GET Methode.

Um Änderungen im Editor zu speichern, wird der Editor Service aufgerufen. Dort gibt es die `pushSessionChanges(...)` Methode welche den aktualisierten Table dem Backend übermittelt.

7.5 Models

7.5.1 CourseSession

Der Großteil unserer Arbeit dreht sich um das CourseSession Model gezeigt in der Abbildung 7.9. Basierend auf dem Course Model [7.10] dient es als Erweiterung. Während von jedem Kurs nur eine Instanz persistent abgespeichert wird, können beliebig viele CourseSessions davon abgeleitet werden. Dabei speichert jede Session die ID des jeweiligen Course (id:number) in der Variable CourseID ab. Jede CourseSession ist durch eine eindeutige ID (id:number) gekennzeichnet.

Bei dem Namen (name:string) gibt es zwei Fälle die zu unterscheiden sind, Gruppen und Splits. Besteht ein Kurs aus mehreren Gruppen, setzt sich der Name aus dem Course Name konkateniert mit der Bezeichnung 'Group' und der jeweiligen Gruppen Nummer zusammen. Beispiele dafür sind Proseminare so wie Seminare. Ebenfalls besteht die Möglichkeit gewisse Kurse wie Vorlesungen oder aufzuteilen. Als Split bezeichnet setzt sich der Name aus dem Course Name konkateniert mit dem Kürzel 'Split' und der jeweiligen Split Zahl zusammen.

Wie im Use-Case-Diagramm [4.1] beschrieben, gibt es die Möglichkeit Kurse zu fixieren (fixed:boolean). Dadurch ist es weder dem/der BenutzerIn noch dem Algorithmus möglich den Kurs neu anzuordnen, zumindest bis er erneut freigegeben wurde. Anhand der Teilnehmer (numberOfParticipants: number) lässt es sich überprüfen ob der jeweilige Raum, ausreichend Kapazitäten bereitstellt.

Zwischen der zugewiesenen Zeit (timing: TimingDTO | null) und dem ausgewählten Raum (roomTable: RoomTableDTO | null) besteht ein Zusammenhang. Wird ein Kurs einem Raum zugewiesen, erhält die CourseSession ebenfalls einen Time Slot. Der Time Slot setzt sich aus einer Start Zeit am Kalender und der Dauer (duration: number) des Kurses zusammen.

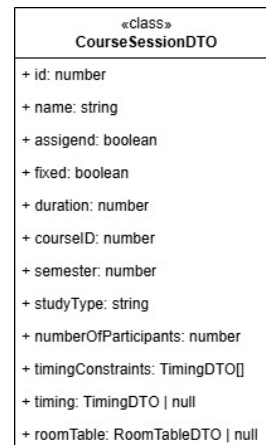


Abbildung 7.9: CourseSessionDTO Model

7.5.2 Course

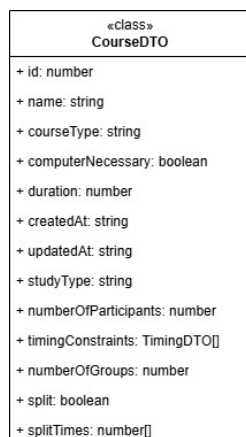


Abbildung 7.10: CourseDTO Model

Course 7.10 ist eine Abbildung der universitären Kurse. Wiedererkennbare Eigenschaften wie die ID (id: number), den Namen (name: string), Kurs Typ (courseType: string) und dem Studien Grad des Kurses (studyType: string), werden direkt übernommen. Während die größten Komponenten (Home 7.7 und Editor 7.9) mit CourseSessions arbeiten, behandelt die Wizard Komponente (7.8) ausschließlich Kurse. Wird ein Kurs als aufgeteilt (split: boolean) definiert. Gibt es entweder mehrere Gruppen oder Aufteilungen. Die Anzahl der Gruppen/Aufteilungen wird separat abgespeichert (numberOfGroups: number). Bei Splits kann es zu einer abweichenden Länge des jeweiligen Splits kommen. Daher wird in einem eigenen Array (splitTimes: number[]) die Dauer des jeweiligen Splits gespeichert. Ein Beispiel für Splits ist die Vorlesungs Lehrveranstaltung Betriebssysteme. Insgesamt sind für die Vorlesung drei Stunden vorgesehen. Um der mit der Zeit sinkenden Konzentration entgegen zu wirken, wurde das Modul aufgeteilt. Der erste Slot (VO Betriebssysteme - Split 1) dauert 2 Stunden während der zweite Slot (VO Betriebssysteme - Split 2) lediglich eine Stunde andauert.

7.5.3 RoomTable

Ebenfalls wird für jeden Raum eine weiterer RoomTable erstellt. Das RoomTable Model kombiniert die letzten zwei Schritte des Wizards Dem Auswählen der verfügbaren Räume und der Bestimmung von zeitlichen Einschränkungen. Gemeint sind damit, Zeiten zu denen der Raum nicht gebucht werden darf.

Die Informationen welcher Raum zu welcher Zeit belegbar ist, ändern sich mit jeder Planung des neuen Semester. Unter anderem wird aufgrund dieser Eigenschaft, mit jedem neuen Plan eine neuer RoomTable erstellt. Ebenfalls gibt es nach beenden des Wizards keine Möglichkeit die blockierten Zeit Slots eines Raumes zu ändern.

7.5.4 TmpTimeTable

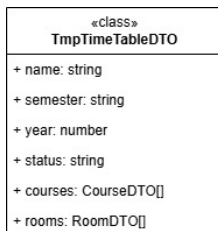


Abbildung 7.11: TmpTimeTable Model

Das Modell *TmpTimeTableDTO* dient als Darstellung eines temporären Stundenplans, der für die Planung, Organisation und Verwaltung von Lehrveranstaltungen genutzt wird. Es handelt sich hierbei um ein flexibles Datenmodell, das die wesentlichen Eigenschaften eines Stundenplans abbildet und dabei eine enge Verbindung zwischen Kursen, Räumen und zeitlichen Rahmenbedingungen herstellt.

Im wesentlichen beinhaltet der Stundenplan grundlegende Daten, welche er vorab von dem CreateTableDialog Komponente erhalten hat. Wie den Namen (name: string) sowie das zugehörige Semester (semester:string) und das Jahr (year:number), welche den zeitlichen Bereich des Plans definieren. Der Status gibt darüber hinaus an, in welcher Phase sich der Stundenplan befindet.

Der Lebenszyklus eines Stundenplans(status: string), wie in Abbildung 7.12 dargestellt, beschreibt die verschiedenen Phasen eines TmpTimeTable, von der Erstellung bis zur Fertigstellung. Beginnend mit dem Status NEW, der einen neu erstellten, noch unbearbeiteten Table abbildet. Anschließend wechselt der Status zu EDITED, sobald Änderungen vorgenommen wurden. Sowohl das Hinzufügen und Entfernen von Kursen beziehungsweise abändern der Gruppen Anzahl oder dem Hinzufügen neuer Zeit Slots im finalen Schritt wird als Änderung angesehen.

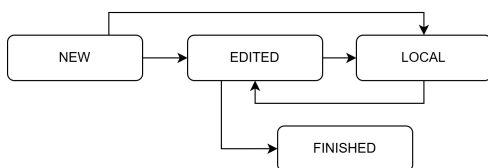


Abbildung 7.12: Lebenszyklus eines TmpTimeTable

Darauf folgt der Status LOCAL, welcher besagt, dass der Table temporär im Localstorage gespeichert wird. Einerseits erscheint der Status, wenn ein Table manuell gespeichert wurde, andererseits wird mit jedem Schritt im Wizard, der Table zwischengespeichert. Der Zyklus endet mit dem Status FINISHED, in dem der Stundenplan vollständig abgeschlossen und abgeschickt wird. Die Angabe sollte die BenutzerInnen unterstützen um Datenverlust zu vermeiden.

Weniger als Unterstützung und mehr dem eigentlichen Zweck dienend, gibt es Arrays für die Kursen (courses: CourseDTO[]) und die Räume (rooms: RoomDTO[]). Die CourseDTO Liste enthält sämtliche Kurse, die in den Stundenplan aufgenommen wurden. Die RoomDTO Liste ergänzt dies, indem sie die ausgewählten Räume speichert.

7.5.5 TimeTable

Das TimeTableDTO Model dient als Erweiterung des TmpTimeTable. Wie in Kapitel 7.5.4 beschrieben, bildet der TmpTimeTable die Rohdaten ab, während im TimeTable für jeden im Wizard hinzugefügten Kurs, eine eigene CourseSession erstellt wird. Im Wesentlichen beinhaltet ein RoomTable die gleichen Elemente wie das Room Model, mit dem Unterschied, dass nicht belegbare Zeiten Slots zusätzlich gespeichert werden.

Erstmals lädt der/die BenutzerIn den TimeTable in der Home Komponente. Dort gibt es ein Dropdown Menu für die Auswahl des Table. Um die zu übertragenden Daten klein zu halten und somit die Wartezeit des/der UserIn zu reudieren, wurde eine abgespeckte Variante des TimeTableDTO names TimeTableNames erstellt. Diese bildet wiedererkennbare Merkmale wie das Jahr, die Semesterwahl und den Namen ab. Ebenfalls enthalten ist die ID des eigentlichen Tables. Wählt ein/eine BenutzerIn den jeweiligen Table im Dropdown Menü aus, werden mittels einer HTTP Abfrage die eigentlichen Kurs Daten, roomTables und courseSessions, abgefragt.

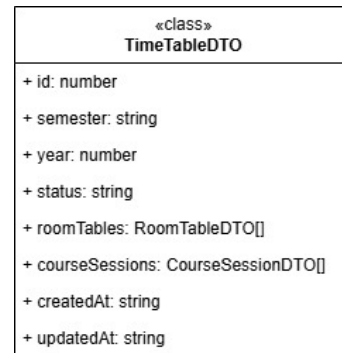


Abbildung 7.13: TimeTable Model

7.6 Login

Die Login Komponente kombiniert die Darstellung und Verarbeitung der Login-Funktionalität. Sie verwendet das Angular-Decorator-Pattern, wobei die *@Component*-Annotation den selektierten HTML-Tag (`<app-login>`), das zugehörige Template, der eigentlichen HTML Datei, und die Styles definiert. In Logik des Codes gibt es eine Unterteilung. Die Variable loginObj hält die Login-Daten des/der BenutzerIn, welche er/sie zuvor auf der Website eingegeben hat. Unter Verwendung der dynamischen Interfaces wird der eingegebene Wert mit jeder Veränderung an die Variable übertragen. Während *rememberMe* als Boolean-Wert dient, um festzulegen, ob die Anmeldedaten gespeichert werden sollen. Im Unterschied zu Enterprise Software, wird der Localstorage anstelle eines Cookies verwendet. Grund dafür war fehlendes Wissen zu Beginn der Entwicklung. Der Login-Prozess wird über die Methode login() gesteuert, welche den LoginUserService aufruft. Auch in der Login Komponente wird Logik ausgelagert. Die vorhin erwähnte Speichermethode, wird im Service anstelle der Komponente gehandhabt.

Innerhalb der Methode login() wird eine HTTP-Anfrage abgesetzt, um die Login-Daten zu überprüfen. Die Validierung der Daten übernimmt die im HTML Code verwendete Password Komponente. Bei erfolgreicher Authentifizierung wird der/die BenutzerIn mithilfe des Angular-Routers (router:Router) auf die Home Komponente (/user/home) umgeleitet. Fehler werden mit dem MessageService behandelt, der dem/der BenutzerIn durch eine einfache, farblich herausstechende Nachricht. Behandelt werden Informationen wie Beispielsweise Probleme bezüglich ungültige Anmeldedaten oder Serverfehler.

Wird eine Komponente nicht mehr benötigt, zerstört Angular diese. Die Verwendung der ngOnDestroy() Methode gewährleistet, dass alle Subscriptions sauber abgebrochen werden. Damit bleibt der Speicher frei von Memory Leaks was zu einer erhöhten Leistung und damit ein kommende Effizienz führt. Der/die NutzerIn wird daraufhin automatisch auf die Startseite der Anwendung weitergeleitet. Der JWT dient ab diesem Moment als Nachweis der Authentifizierung und wird bei jeder weiteren Kommunikation zwischen dem Client und dem Server verwendet.

Wenn eine/eine UserIn auf eine geschützte Ressource zugreifen möchte, wird der JWT in den HTTP Header der Anfrage eingebettet. Der Server prüft bei jeder Anfrage die Gültigkeit des Tokens, um sicherzustellen, dass er weder manipuliert wurde noch abgelaufen ist. Nur wenn

Verwendung jener Daten wird in Der Abbildung 7.4 erklärt. Zuletzt wird, in Blau dargestellt, die Signatur des Token mitgeschickt. Diese ist ein zentraler Bestandteil des Sicherheitsmechanismus und spielt eine entscheidende Rolle in der Integrität und Vertrauenswürdigkeit des Tokens. Ihre Hauptaufgabe besteht darin, sicherzustellen, dass der Inhalt des Tokens nicht manipuliert wurde.

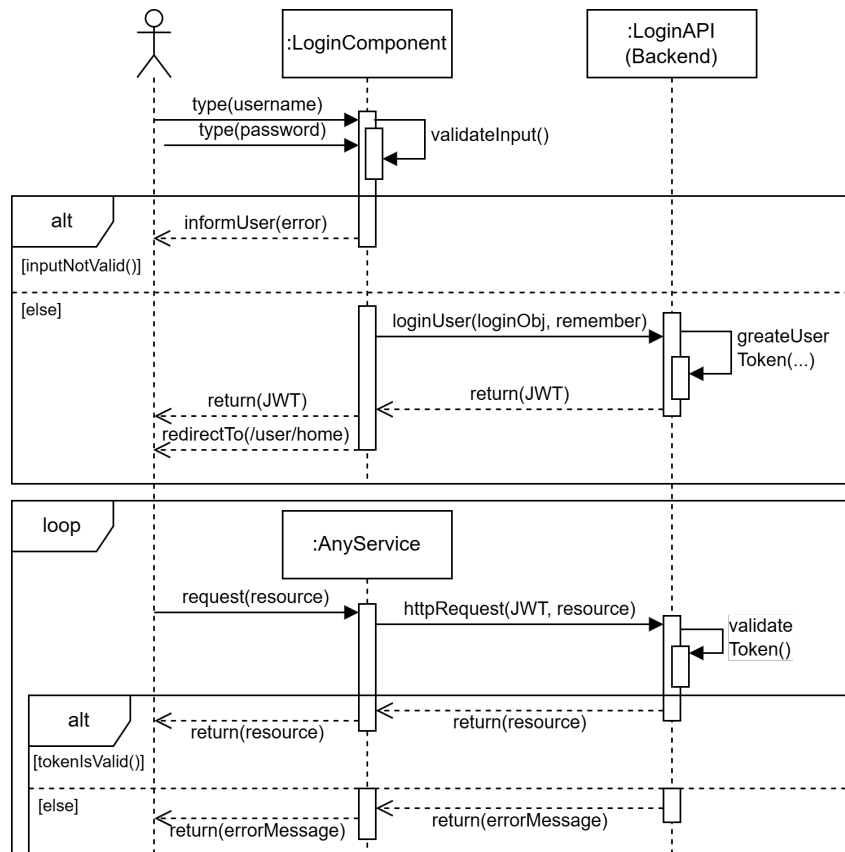


Abbildung 7.16: Ablauf Token Austausch

7.7 Home

Nach dem Login werden UserInnen auf die Home Komponente umgeleitet. Vor sich sehen sie einen mit Kursen gefüllten Kalender. Um die Daten zu erhalten, benötigt es mehrere Schritte beginnend mit dem Laden aller TimeTableName Objekte im Constructor.

7.7.1 Constructor

In der Abbildung 7.17 wird der GlobalTableService und die Home Komponente, mit den für die Initialisierung benötigten Variablen, angezeigt. Die Liste der existierenden Table, des Datentyp `Observable<TimeTableNames[]>`, entspringt dem Observable Muster der Bibliothek RxJs. Um die Daten zu erhalten, muss das Objekt abonniert werden. Innerhalb der `.subscribe()` Methode werden die Daten in ein separates Array (`availableTables: TimeTableNames[]`) gespeichert.

Die Subscription welche beim Abonnieren entsteht, wird in der Variable `availableTableSubs` gespeichert. Überlässt man ausschließlich dem *Garbage Collector* das Bereinigen des Speichers, kann es zu Speicher Löchern kommen. Gerade kleine Daten wie Subscriptions, die weiterhin

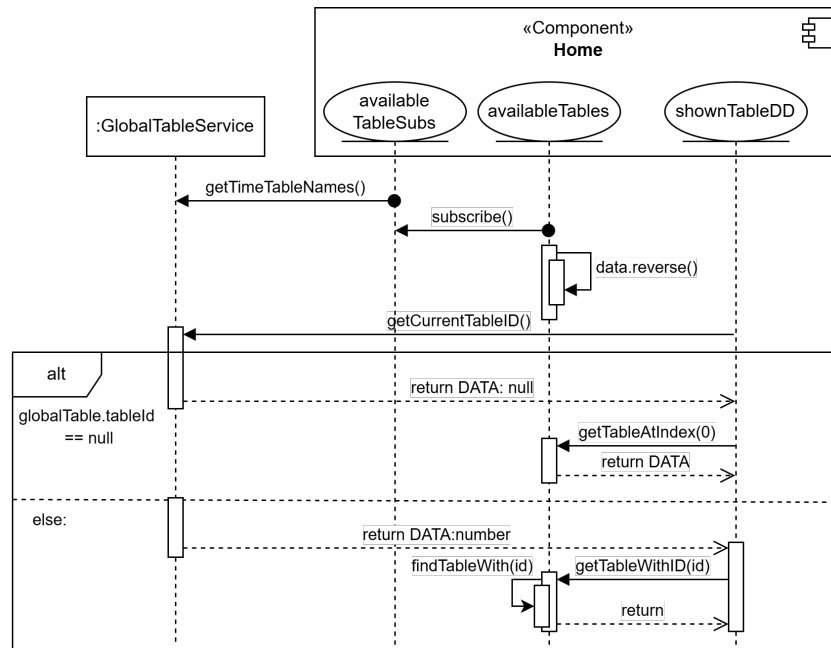


Abbildung 7.17: Construtor Ablauf

referenziert allerdings nicht benötigt werden, erzeugen solche Löcher. Aufgrund dessen wird die Variable gespeichert um beim Zerstören einer Komponente die Subscription manuell frei zu geben und somit aus dem Speicher zu löschen.

Die Liste der existierenden Tables wird invertiert, um die Auswahl des Tables mit dem Neuesten beginnen zu lassen. Der GlobalTableService speichert eine eigene Variable der momentanen TableId. Dies hat den Hintergrund, dass beim Beenden des Editors, die TableId im Service auf die Id im Editor gesetzt wird. Beendet nun der/die BenutzerIn die Arbeit im Editor und kehrt zur Home Ansicht zurück, wird der eben editierte Table mit den bereits neuen Änderungen angezeigt.

Der Alt-Block des Diagramms 7.17 zeigt zwei Fälle die auftreten können. Wird die Komponente direkt nach der Anmeldung aufgerufen, beträgt der Wert der Variable tableId *null*. Damit wird der Eintrag bei Index 0 der *avaibleTables* Variable geladen. Zweitens gibt es den Fall, dass eine Id im Service hinterlegt ist.

Dies ist Beispielsweise nach dem Beenden des Wizard oder dem Editor der Fall. Dieser Zustand wird unterhalb im Else-Block dargestellt. Anhand der Id wird die Liste der *avaibleTables* durchsucht. Dieser Schritt diente lediglich als Initialisierung der Seite. Im nächsten Schritt, dem Laden eines spezifischen Tables, werden die Einträge des Tables geladen.

7.7.2 Laden eines konkreten Tables

Unter anderem wird die Methode *.loadSpecificTable()* im Constructor aufgerufen. Im DropDown Menü, wird die Liste der TimeTableNames Optionen angezeigt. Wird eine Option manuell ausgewählt passieren zwei Aktionen. Erstens wird der Variable *shownTableDD* der Option des Menüs zugewiesen. Zweitens aktiviert sich das *onChange* Interface, welches die *.loadSpecificTable()* Methode aufruft. Initial werden die bisher angezeigten Kurs Kollisionen gelöscht. Andererseits würden beim Aufrufen des *Collision Dialog* die Kurse des letzten Kalenders angezeigt werden. Dieser Schritt wird mit der *.clearCollisions()* Methode gezeigt. Der *Change Dialog* ist nach der selben Logik wie der *Collision Dialog* aufgebaut. Daher müssen auch bei diesem die bestehenden Daten bereinigt werden. Im *Change Dialog* werden Änderungen der Kurse anstelle von Kollisio-

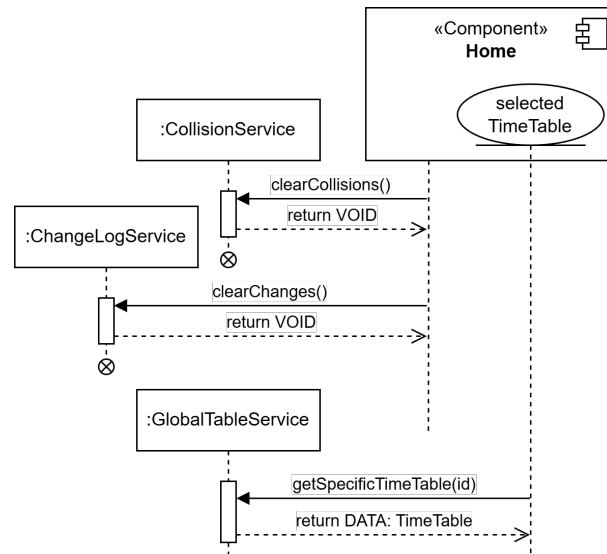


Abbildung 7.18: Laden eines spezifischen Tables

nen angezeigt. Daher wurde die Methode zur Bereinigung der Daten *.clearChanges()* benannt. Abschließen wird die *.getSpecificTimeTable(id)* des GlobalTableService aufgerufen. Diese sendet eine HTTP Anfrage an die API Schnittstelle des Backends *\$environment.baseUrl/api/global/\$id*. Dadurch wird der Table der mitgeschickten Id zurückgegeben.

7.7.3 Aktualisieren aller Kurse

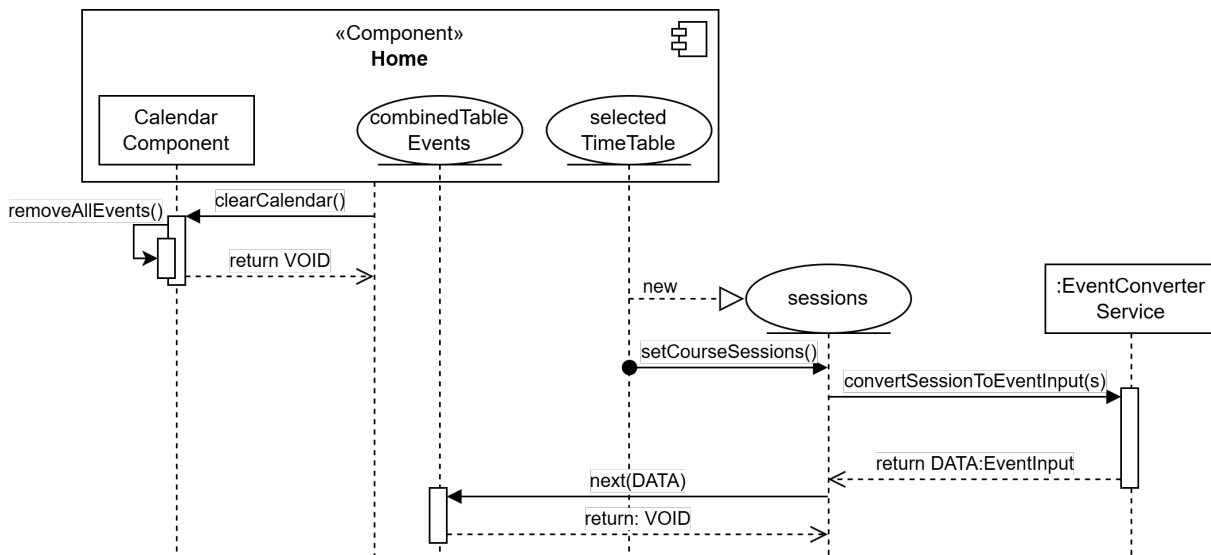


Abbildung 7.19: Aktualisieren der Kalender Daten

Das Sequenz Diagramm der Abbildung 7.19 beschreibt den finalen Schritt um einen TimeTable anzuzeigen. Nach der Auswahl eines konkreten Zeitplans (Table) im Editor müssen die zugehörigen Daten von *CourseSession* Daten in *EventInput* Daten umgewandelt werden. Da die Kalender Komponente der FullCalendar Bibliothek entspricht, musste eine Konvertierung eingebaut werden. Um eine Kollision bisher geladener und den neu hinzugefügten Kurse zu vermeiden, wird die *.clearCalendar()* Methode der Kalender Komponente aufgerufen.

Beim ersten Laden eines Tables ist die Kalender Komponente bereits leer. Dies ist allerdings nur direkt nach dem Login der Fall. Mit jeder neuen Auswahl im DropDown Menü, wird nicht nur die Logik des vorherigen Kapitels 7.7.2, sondern auch die *.updateCourses()* Methode.

Ist der Kalender bereinigt, kann die eigentliche Arbeit beginnen. Um die Leserlichkeit des Codes zu vereinfachen, wurde in der *.updateCourses()* Methode eine neue Variable *names sessions* erstellt. Diese referenziert auf die Liste an *CourseSessions* aus der *selectedTimeTable* Variable. Die Funktionsweise und etwaige Komplikationen in Bezug auf die Konvertierung werden im Kapitel 7.4.1 erklärt. Einfach gesagt, werden die wichtigsten Eigenschaften wie der Titel, der Beginn, und die Dauer in die jeweiligen Variablen gespeichert.

Nachdem die konvertierten Daten empfangen wurden, wird die nächste Session verarbeitet, bis alle Sessions aus der Zeittabelle in *EventInput*-Daten überführt wurden. Dieser Prozess stellt sicher, dass die Daten des ausgewählten Zeitplans korrekt in den Kalender integriert werden können. Das Diagramm zeigt, wie die verschiedenen Komponenten – der Editor, die Kalenderkomponente und der *EventConverter Service* – miteinander interagieren, um die Kursdaten zu aktualisieren und darzustellen.

Abschließend wird der Rückgabe Wert der *convertSessionToEventInput(s)* Methode an die Variable *combinedTableEvents* (*BehaviorSubject<EventInput[]>*) übergeben. Diese wird zwischen dem Kalender und der Home Komponente geteilt. Um zu verhindern, dass mit jedem neu geladenen Table das letzte Observable de-abonniert werden muss, wird ein *BehaviorSubject* verwendet. Mit jedem Aufruf der *.next(DATA)* wird der alte Datensatz gelöscht und durch einen neuen ersetzt. Da der Kalender kein Interface für die *BehaviorSubject* Klasse bietet, muss die *.asObservable()* Methode in Kombination mit der Async Pipe verwendet werden. Endlich werden alle Daten des *TimeTable* angezeigt und der/die BenutzerIn kann mit etlichen Features vorgehen.

7.8 Wizard

Die Abbildung 7.20 beschreibt den Aufbau der Wizard Komponente, In Kombination mit den jeweiligen PrimeNG-Komponenten *p-stepper*, *p-stepperPanel* und *Dialog* wurde eine übersichtliche Darstellung ermöglicht. Die Hauptkomponente, *Wizard*, umfasst den gesamten Ablauf des Wizards, schrittweise dargestellt in einem Panel der *p-stepper*, welches den momentanen Schritt anzeigt. Jede Komponente innerhalb der *p-stepperPanel* Seite repräsentiert eine eigene Seite. Um Wartezeiten durch die Umleitung auf weitere Seiten zu verhindern, wurde die Komponenten verschachtelt.

Innerhalb des Wizards ist der *p-stepper* das zentrale Element, das die Navigation zwischen verschiedenen Panels ermöglicht. Jedes Panel wird von einer *p-stepperPanel*-Komponente dargestellt, die eine individuelle Überschrift (*Header*) inklusive der Farben, Icons und den spezifischen Inhalt des jeweiligen Schrittes enthält.

Innerhalb der Wizard Komponente befindet sich die *InfoDialog* Komponente. Dieser gibt Information bezüglich dem momentanen Schritt (*active:number*) im Wizard. Beispielsweise befindet sich ein/e BenutzerIn in dem ersten Schritt des Wizards, *Course Selection* und öffnet den *InfoDialog*. Daraufhin erscheint eine Informationsbox, welche in einfachen Worten mit Beispielen die Kursauswahl und ihre Relevanz erklärt.

Im Wizard wird der momentane Index des stepper mitgezählt. Beendet der/die BenutzerIn den momentanen Schritt, und geht in den Nächsten über, erhöht sich der Index um 1. Ebenfalls ist es möglich in frühere Wizard Schritte zurück zu gehen, woraufhin der Index reduziert wird. Basierend auf dem Index werden gewisse Buttons blockiert. Ist der Index bei 0, sprich die erste Seite des Wizards wird angezeigt, kann der/die BenutzerIn keinen Schritt zurück tätigen, da keiner existiert. Das selbe gilt für die letzte Ansicht mit Index 3.

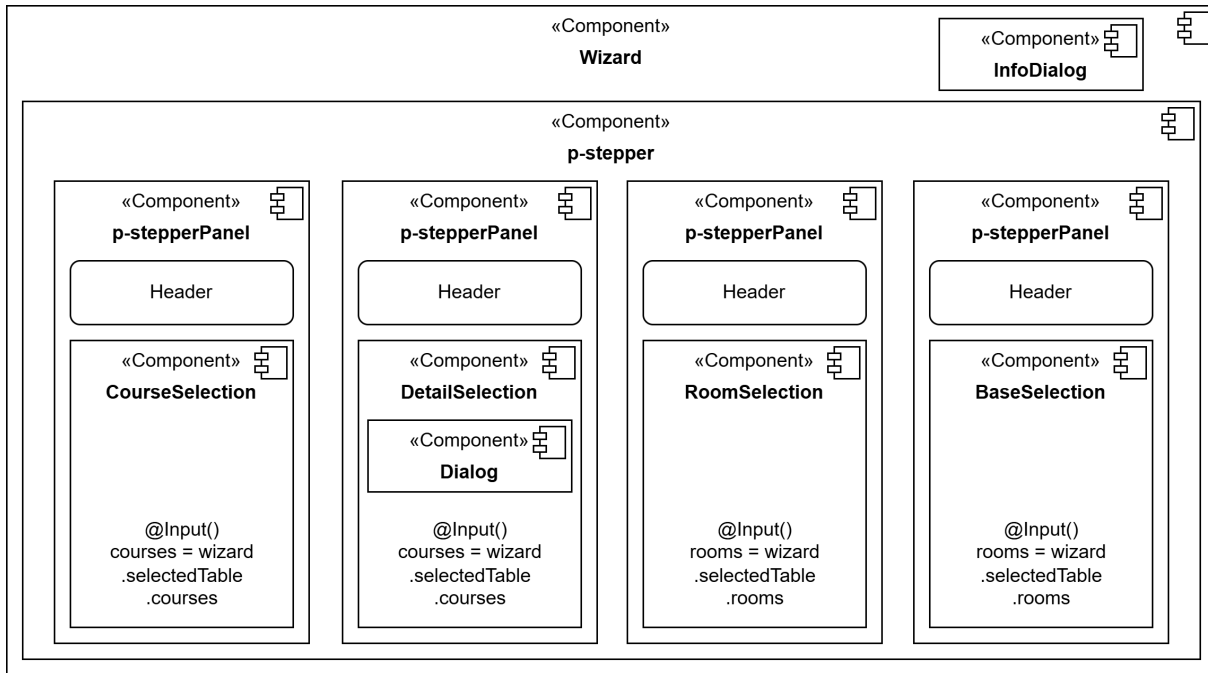


Abbildung 7.20: Wizard Komponenten Zusammenhänge

Der erste Schritt wird durch die **CourseSelection**-Komponente repräsentiert, in der die BenutzerInnen eine Auswahl an Kursen treffen können. Diese Komponente erhält ihre Daten über das `@Input()`-Binding, das auf `wizard.selectedTable.courses` verweist, um auf die in der Hauptkomponente gespeicherten Kurse zuzugreifen. Die Variable `@Input() courses: Courses[]` wird in die Kindkomponente kopiert und nicht referenziert.

Da Sämtliche Kurse in der Wizard Komponente gespeichert werden, gibt es zwei `@Output`-Methoden vom Typ `EventEmitter<Course>` names `add-` und `removeCourseInParent`. Beim Aufruf einer der Funktionen, wird ein Kurs von der Child-Komponente `CourseSelection` an die Parent-Komponente `Wizard` übergeben. Je nach Methode wird im Wizard der Kurs hinzugefügt beziehungsweise entfernt.

Der nächste Schritt des Wizards, die **DetailSelection**, bestimmt die Anzahl der Gruppen beziehungsweise Splits für die zuvor ausgewählten Kurse. Wählt der Benutzer einen Kurs aus, öffnet sich ein weiteres Dialog Fenster. Darin enthalten ist der Name des Kurses so wie eine Anzeige der Dauer des Kurses. Je nach Kurs Typ, werden unterschiedliche Oberflächen angezeigt. Mittels der `.hasPsType()` Methode wird entschieden, welche Oberfläche geladen wird. Hat der Kurs Proseminar Charakter, erscheint ein Eingabefeld für die Anzahl der Gruppen.

Standardmäßig ist der Wert auf 0 gesetzt. Ändert sich der Wert auf N , wird ein neues Array erstellt, welches N Einträge der Kursdauer speichert. Handelt es sich bei dem Kurs um kein Proseminar, ändert sich die Ansicht. Mittels dem ersten Feld werden die Anzahl der Splits angegeben. Basierend auf Dieser Eingabe, werden $N-1$ weitere Felder erstellt. Jedes Feld repräsentiert die Dauer des jeweiligen Splits. Ein zusätzliches Feld wird am Ende erstellt, welches anhand dieser Formel,

$$\text{Final}_{Duration} = \text{Course}_{Duration} - \sum_{i=1}^{N-1} \text{Split}_{Duration}$$

berechnet wird. Ist der berechnete Wert kleiner als 0, wird eine Fehlermeldung ausgegeben, schließlich können Teile eines Kurse keine negative Zeit beanspruchen.

Im dritten Schritt, der durch die `RoomSelection`-Komponente dargestellt wird, können Räume ausgewählt werden. Diese Komponente arbeitet ähnlich wie die `CourseSelection`, jedoch bindet sie sich an `wizard.selectedTable.rooms`, um auf die verfügbaren Räume zuzugreifen.

Der vierte und letzte Schritt ist die `BaseSelection`, welche einen Kalender der `FullCalendar Library` anzeigt. In diesem Kalender können Zeitfenster ausgewählt werden, welche entweder orange oder schwarz markiert sind. Orangene Zeitfenster werden bevorzugt und sollen von BenutzerInnen ausgewählt werden, während schwarze ignoriert werden müssen. Dies bietet eine intuitive Möglichkeit, Zeitpläne auf Basis von Verfügbarkeiten oder Präferenzen festzulegen.

Die modulare Architektur des Wizards ermöglicht eine klare Trennung der einzelnen Schritte und eine einfache Erweiterbarkeit. Die globale `InfoDialog`-Komponente sowie die geschickte Nutzung von Datenbindungen erleichtern die Wiederverwendbarkeit und Wartbarkeit des Codes.

7.9 Editor

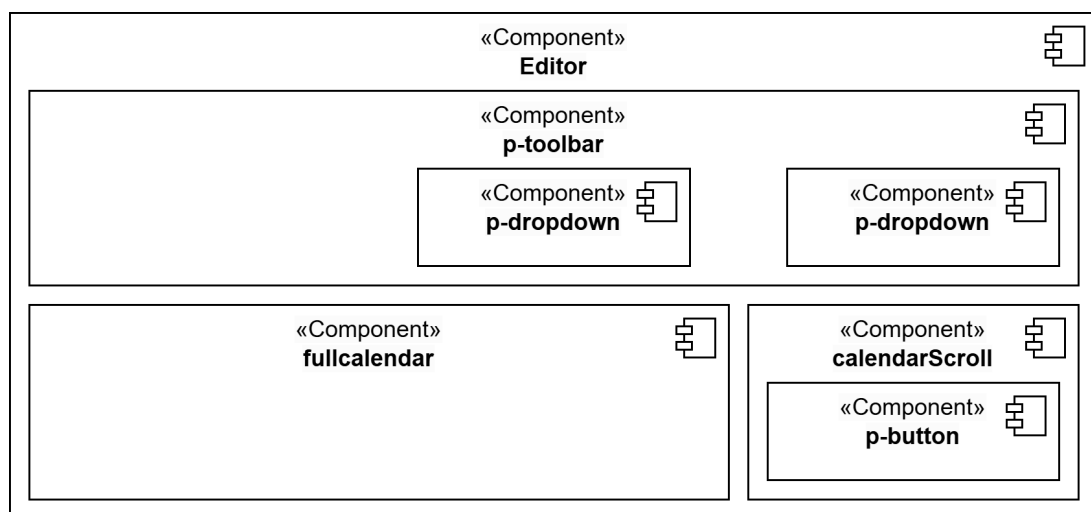


Abbildung 7.21: Editor Komponenten Diagramm

Die Editor Komponente basiert auf mehreren Modulen. Wie Üblich gibt es, visuell gesehen, oben eine Menü Leiste. Diese besteht aus je zwei unterschiedlichen DropDown Auswahlfeldern. Zentral der *p-toolbar* Komponente befindet sich die Auswahl der Räume. Beim öffnen des Editors wird der Raum mit Index 0 geladen. Entscheidet sich der/die BenutzerIn den Raum zu ändern, wird der *OnChange* Listenenser ausgelöst, welcher die *.loadNewRoom(\$event.value)* Methode aufruft. Die Änderung des Raumes verläuft ähnlich wie das Laden eines neuen Kalenders in der Home Komponente. Zuerst werden die momentan geladenen Kurse entfernt. Da mit jedem Raum nur die darin enthaltenen Kurse konvertiert werden, muss die Liste aller Kurse gefiltert und in EventInput Daten konvertiert werden. Schlussendlich wird die Liste der angezeigten Kurse getauscht und um die Hintergrund Time Slots erweitert.

Zusätzlich verbaut die Toolbar eine Suchfunktion. Befindet sich der Kurs in einem anderen Raum als dem momentanen wird die *.changeRoom(\$event)* Methode aufgerufen. die ändern den Raum im ersten DropDown Menü, woraufhin die vorher erwähnte Logik aufgerufen wird.

Zentral befindet sich die FullCalendar Komponente. Wie üblich werden dort sämtliche Kurse angezeigt. Um den Kalender per *Drag and Drop* mit anderen Komponenten zu verknüpfen, wurden die Methoden, *.eventRecieve(event: Event)*, *.eventChange(event: Event)* und *.eventDidMount(event: Event)* verwendet.

- **eventRecieve(event: Event)**

Wird ein Event von einer externen Quelle in den Kalender gezogen, wird die Methode *eventReceive* aufgerufen und. Zunächst prüft sie anhand der Methode *allowDrop*, ob das Event an der neuen Position erlaubt ist, indem die Raumkapazitäten und Kursteilnehmer verglichen werden. Übersteigt die Anzahl der Teilnehmer die Raumkapazitäten, wird die *revert()* Methode aufgerufen. Dadurch wird die Platzierung im Kalender verhindert und das Drag Element wird in die Auswahlliste hinzugefügt.

Wird das Event akzeptiert, gibt es nicht gespeicherte Daten, weshalb die *dirtyData* Flag auf *true* gesetzt wird. Diese Eigenschaft ist Teil des *canDeactivate* Interface und wird beim verlassen der Seite aufgerufen. Zusätzlich speichert sie Details des abgelegten Events für eine spätere Verarbeitung und aktualisiert die *CourseSession*-Daten, um den neuen Zustand des Kalenders zu reflektieren.

- **eventChange(event: Event)**

Die Methode *eventChange* wird aufgerufen, wenn sich ein Event im Kalender ändert, beispielsweise durch Verschieben oder Bearbeiten des/der BenutzerIn. Sobald dies geschieht, speichert die Methode die Änderung in der Eigenschaft *rightClickEvent*, für die spätere Verarbeitung.

Anschließend aktualisiert sie die Session-Daten, indem sie die Methode *updateSession* aufruft und das geänderte Event sowie einen *true*-Wert übergibt. Der *true*-Wert gibt an, dass die Session zugewiesen wurde. Spezifisch beim laden neuer Räume wird dieses Attribut aufgerufen. Ebenfalls wird erneut die *dirtyData* Flag auf *true* gestzt.

- **eventDidMount(event: Event)**

Die Methode *eventDidMount* wird ausgeführt, sobald ein Event erfolgreich in den Kalender gerendert wurde. Sie ermöglicht die Anpassung oder Interaktion mit dem gerenderten HTML-Element, das dieses Event repräsentiert.

Da die Anzeige der offenen und zugewiesenen Kurse visuell und logistisch getrennt ist, braucht es eigene *EventListener*. Daher wird in dieser Methode für das jeweilige Event ein Listener hinzugefügt, ob der Kurs gerade gezogen wird.

8 Evaluierung und Diskussion

8.1 Backend

Um während der Entwicklungsphase eine konstante Softwarequalität sicherzustellen, wurden die Backend-Komponenten, insbesondere die Service- und Controller-Klassen, mithilfe des Test-Frameworks JUnit 5 regelmäßig getestet. Durch die insgesamt über 100 Unit-Tests ergibt sich dabei eine Testabdeckung von ca. 75 %.

Für das Testen des Zuweisungsalgorithmus wurden mehrere **TimeTable**-Objekte erstellt, die auf verschiedene Szenarien und Randbedingungen abgestimmt waren:

- *TestForManyGroups*: Testszenario, welches die Zuweisung eines Kurses mit einer großen Anzahl von Gruppen in einem Raum mit starker zeitlicher Einschränkung testet.
- *TestWithSplits*: Testszenario, in dem Kurse getestet wurden, die auf mehrere Zeiteinheiten aufgeteilt sind.
- *TestWithoutConstraints*: Testszenario mit einer großen Anzahl von Kursen und Räumen ohne zeitliche Beschränkungen. Dieser **TimeTable** diente vor allem dazu, die Einhaltung aller Hard und Soft Constraints bei der Ausführung des Algorithmus zu testen.
- *TestWithManyCollisions*: Testszenario mit vielen bereits zugewiesenen Kursen, die gegen verschiedene Constraints verstoßen. Hier wurde neben dem Collision Check auch die Fähigkeit des Algorithmus getestet, anstelle aller Kurse nur kollidierende Kurse neu zuzuweisen.
- *TestWithProblematicConstraints*: Testszenario, bei dem die verfügbaren Zeiten stark eingeschränkt waren, was für die frühen Versionen des Zuweisungsalgorithmus bereits schwer zu bewältigen war.
- *RealisticTest*: Testszenario, das die Raumplanung für das Wintersemester 2024 realistisch darstellt. Für dessen Realisierung wurden die originalen Grundbelegungen sowie alle zuzuweisenden Kurse dieses Semesters miteinbezogen.

Zusätzlich zu den genannten Testszenarien wurde die Funktionalität einzelner Kernmethoden des Algorithmus durch umfassende Unit-Tests verifiziert.

8.2 Frontend

Cross-Origin Resource Sharing

CORS, kurz für Cross-Origin Resource Sharing, ist ein Mechanismus, welcher das Teilen von Ressourcen über mehrere Domänen erlaubt. Daher hat auch jedes System seine eigene Domäne. Aufgrund der *Same-Origin Policy* hat sich dieses Konzept durchgesetzt. Persönliche Informationen wie BenutzerInnendaten sollten dadurch geschützt werden.

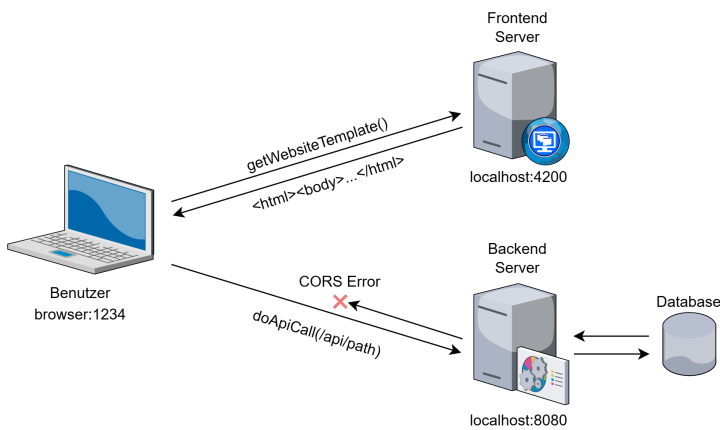


Abbildung 8.1: CORS Fehler

Die *Same-Origin Policy* erlaubt es JavaScript, nur mit Ressourcen zu interagieren, die von derselben Domain, demselben Protokoll und demselben Port stammen wie die Website, auf der das Skript ausgeführt wird. Die Systeme für Front- und Backend wurden in zwei Docker Containern über dieselbe Domain, jedoch unterschiedliche Ports aufgesetzt.

Die Problematik beginnt bei der Kommunikation zweier Systeme. Regelmäßig sendet das Frontend Anfragen an das Backend über die API

Schnittstellen. Im Entwicklermodus wird das Frontend über die Domäne *localhost:4200/* betrieben, während das Backend über die Domäne *localhost:8080/* läuft. In der Abbildung 8.1 wird dieser Fall dargestellt. Standardgemäß wird die Anfrage vom Browser blockiert, da die Anfrage an *localhost:4200/* gestellt, allerdings an *localhost:8080/* gesendet wurde. Mittels CORS sollte eine Möglichkeit geschaffen werden, um frei von Risiken bestimmte grenzüberschreitende Anfragen zu erlauben. Dazu sendet der Browser bei einer CORS-Anfrage automatisch zusätzliche HTTP-Header, wie Origin, an den Zielsystem. Dieser muss explizit durch den Header Access-Control-Allow-Origin angeben, welche Ursprünge Zugriff auf die Ressource haben. Fehlt dieser Header oder ist er falsch konfiguriert, wird die Anfrage blockiert.

Für die Lösung des Problems wurde ein Proxy eingerichtet. Der Proxy agiert als Vermittler zwischen der BenutzerInnenanfrage und der API Schnittstelle. Statt die API direkt anzusprechen, sendet das Frontend des/der BenutzerIn die Anfrage an den Proxy-Server. Dieser befindet sich in der selben Domäne wie der Backend Server. Aufgrund dessen können diese Systeme kommunizieren, ohne eine CORS Meldung zu erhalten.

9 Zusammenfassung und Ausblick

Zusammenfassung

In dieser Arbeit wurde das Lehrplanungstool **Lecture Connect** vorgestellt, welches den Prozess der Kurs- und Raumplanung am Institut für Informatik der Universität Innsbruck effizienter und benutzerfreundlicher gestaltet. Basierend auf den in Kapitel 3 definierten Anforderungen wurde eine Webanwendung implementiert, die sowohl ein robustes Backend als auch ein interaktives Frontend umfasst.

Das Backend, entwickelt mit Java Spring Boot, bietet zentrale Funktionalitäten wie die persistente Speicherung von Stundenplänen in einer MySQL-Datenbank, die Sicherstellung der Datenintegrität und die Implementierung eines Zuweisungsalgorithmus. Der Algorithmus berücksichtigt sowohl Hard als auch Soft Constraints und ermöglicht eine automatische Zuweisung von Kursen zu Räumen.

Das Frontend, basierend auf Angular, stellt eine intuitive Benutzeroberfläche bereit, die eine übersichtliche Visualisierung der Stundenpläne sowie deren interaktive Anpassung ermöglicht. Zu den zentralen Features zählen eine Kalenderansicht mit Drag-and-Drop-Funktionalität, Exportmöglichkeiten in PDF-Format und ein Wizard, der BenutzerInnen schrittweise durch die Erstellung neuer Stundenpläne führt.

Zusätzlich wurden weitere Funktionen wie der „Semi-Automatic Assignment“-Modus und Filteroptionen zur Anpassung der Kalenderansicht realisiert, die über die definierten Anforderungen hinausgehen. Die Bereitstellung der Software geschieht mittels Docker auf einem Server der Universität Innsbruck, wodurch eine einfache Wartung und Skalierbarkeit sichergestellt werden.

Ausblick

Obwohl **Lecture Connect** bereits eine umfassende Lösung für die Lehrplanung bietet, gibt es mehrere Bereiche, in denen die Anwendung zukünftig erweitert und verbessert werden kann, um zusätzliche Anforderungen zu erfüllen und die Benutzerfreundlichkeit weiter zu erhöhen.

Synchronisierung für parallelen Zugriff

Derzeit ist die Anwendung für die Nutzung durch einen einzelnen Benutzer ausgelegt. Es gibt keine Synchronisationsmechanismen wie Sperren oder ähnliche Funktionen, die verhindern, dass mehrere BenutzerInnen gleichzeitig ein **TimeTable**-Objekt bearbeiten. In zukünftigen Versionen sollte eine Synchronisierung eingeführt werden, um parallelen Zugriff zu ermöglichen. Beispielsweise könnten Sperren für **TimeTable**-Objekte implementiert werden, die verhindern, dass zwei BenutzerInnen gleichzeitig Änderungen an denselben Daten vornehmen.

Editieren von Timing Constraints nach der Erstellung

Aktuell können die Timing Constraints eines Raumes nur im Wizard festgelegt werden. Dies schränkt die Flexibilität der Anwendung ein, insbesondere wenn sich Planungsanforderungen während eines Semesters ändern. Zukünftig sollte es möglich sein, Timing Constraints wie reservierte Zeiten für Informatik oder andere Studienrichtungen auch nachträglich im Editor hinzuzufügen, zu entfernen oder zu bearbeiten.

Ausweiten der Timing Constraints

Derzeit decken die Timing Constraints vor allem die Studienrichtung Informatik ab. In einer erweiterten Version der Anwendung könnten Timing Constraints so gestaltet werden, dass sie auch andere Studienrichtungen wie Physik oder Mathematik abdecken. Dadurch wird die Anwendung für einen breiteren BenutzerInnenkreis interessant und flexibler einsetzbar.

Diese Erweiterungen und Verbesserungen bieten großes Potenzial, die Funktionalität von **Lecture Connect** weiter zu steigern und es als zukunftsfähiges Tool für die universitäre Lehrplanung über das Institut für Informatik hinaus zu etablieren.

Literaturverzeichnis

- components. Composition with components angular. <https://angular.dev/essentials/components>, 11 2023. (Zuletzt besucht: 24/11/2024 19:55).
- Dto Pattern Website. The dto pattern data transfer object baeldung. <https://www.baeldung.com/java-dto-pattern>, 09 2021. (Zuletzt besucht: 25/11/2024 21:25).
- Martin Fowler. P of eaa. <https://martinfowler.com/books/ea.html>, 07 2012. (Zuletzt besucht: 25/11/2024 21:40).
- interfaces. Reactivity with signals angular. <https://angular.dev/essentials/signals>, 2024. (Zuletzt besucht: 24/11/2024 19:55).
- Gerald Lach and Erhard Zorn. Ein neuer algorithmus zur lösung des raumplanungsproblems an universitäten. <https://page.math.tu-berlin.de/~lach/publications/DeLFI08.pdf>.
- lifecycle. Angular component lifecycle. <https://v17.angular.io/guide/lifecycle-hooks>, 06 2024. (Zuletzt besucht: 9/12/2024 12:52).
- Carol Meyers and James B. Orlin. Very large-scale neighborhood search techniques in timetabling problems. In *Proceedings of the PATAT Conference*, 2006.
- Robert Nieuwenhuis and Roberto Asin Acha. Curriculum-based course timetabling with sat and maxsat. In *Proceedings of the PATAT Conference*, 2010.
- signals. Reactivity with signals angular. <https://angular.dev/essentials/signals>, 2024. (Zuletzt besucht: 24/11/2024 19:55).