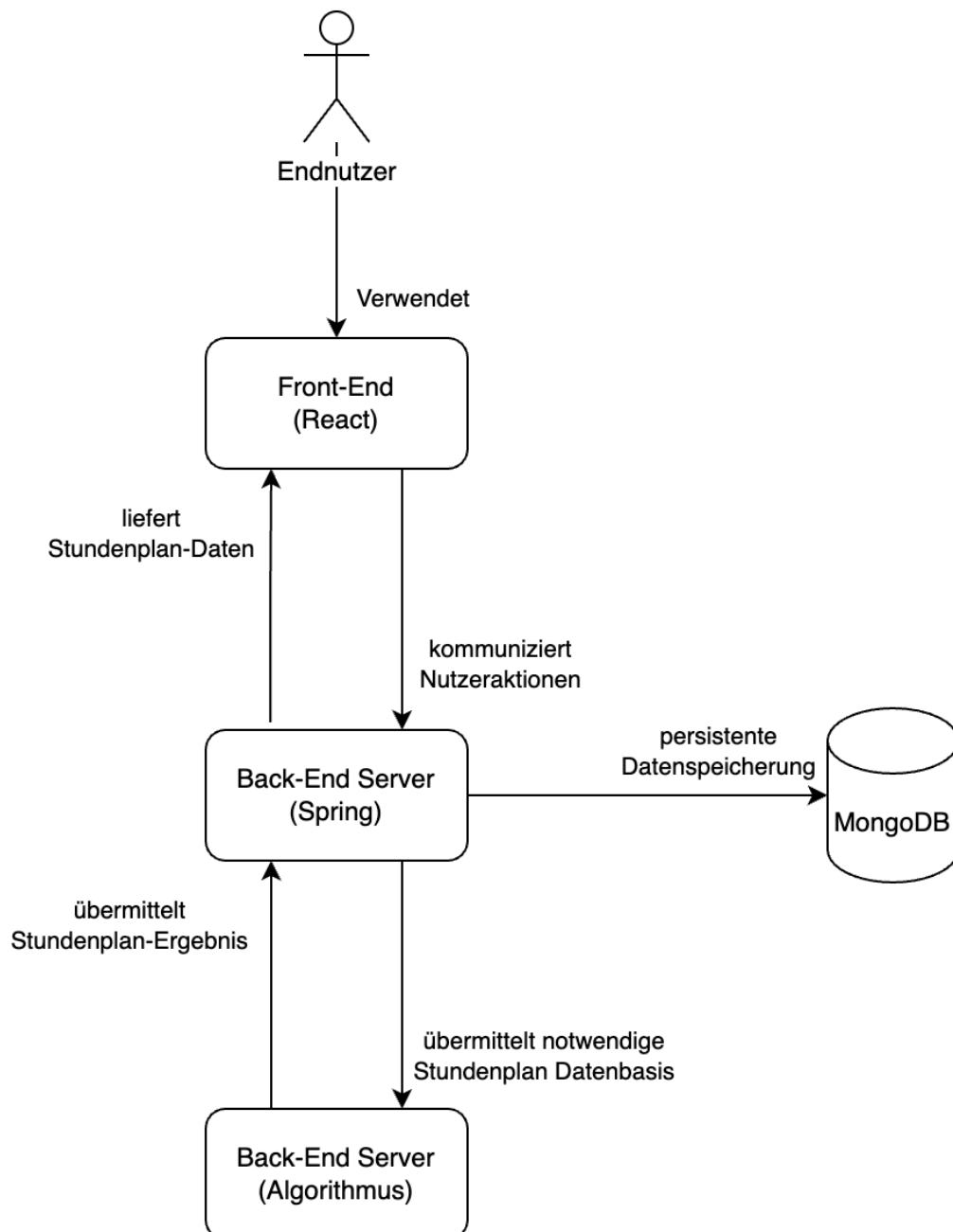


Stundenplan Softwareprojekt.....	2
Algorithmus.....	3
Prozessablauf.....	4
Setup.....	4
Endbenutzer Setup.....	5
Entwickler Setup (Mit Docker).....	6
Entwickler Setup (Ohne Docker).....	7
Testen.....	7
Nutzung.....	8
Weboberfläche.....	8
API Endpunkte.....	9
Datenmodelle.....	10
Atomare Modelle.....	10
Datenbasis.....	11
Ergebnis.....	12
Implementation.....	12
Constraints.....	13
React Frontend.....	15
Beschreibung.....	15
Stundenplan-View.....	15
Constraints-View.....	15
Admin-Dashboard-View.....	17
Setup.....	18
Endbenutzer Setup.....	18
Entwickler Setup (Mit Docker).....	20
Entwickler Setup (Ohne Docker).....	21
Projektstruktur.....	21
Spring.....	21
Spring-Projektaufbau.....	22
Spring API Übersicht.....	24
Datenbank.....	24
MongoDB.....	25
Unsere Datenbank.....	25
Endprodukt.....	27

Stundenplan Softwareprojekt

Das Softwareprojekt "Stundenplan25" bietet eine bedienbare Software zum Erstellen von Stundenplänen an.

Eine vereinfachte Darstellung der gesamten Pipeline der 4 Container sieht so aus:



Algorithmus

Der Stundenplan Algorithmus ist als Server implementiert, damit dieser nicht zwangsweise auf dem gleichen System wie die restliche Applikation laufen muss. Der Algorithmus empfängt Daten, welche eine Basis des Stundenplans darstellen, und generiert daraus einen vollständigen Stundenplan unter Bedingung von Constraints.

Der Algorithmus wurde als ein Server implementiert, damit dieser nicht zwangsweise auf dem gleichen Host-System wie die restliche Applikation laufen muss, dadurch wird die Performance des Front-Ends nicht beeinträchtigt wenn ein Stundenplan generiert wird.

Befehle und Daten werden an den Server über eine REST API kommuniziert, um eine standardisierte Schnittstelle zu bieten. Hierüber kann eine neue Datenbasis zur Grundlage eines Stundenplans gespeichert werden, der Algorithmus gestartet werden und das generierte Resultat abgerufen werden.

Der Algorithmus läuft entweder bis ein optimaler Stundenplan erstellt wurde, oder bis eine vordefinierte maximale Anzahl an Generationen erreicht wurde. Letzteres ist erforderlich, da es möglich ist, dass es keinen optimalen Stundenplan zu einer Datenbasis gibt, da ansonsten der Algorithmus nicht stoppen wird.

Ein Stundenplan ist optimal, wenn keine Constraints verletzt sind. Dabei ist zwischen drei verschiedenen Arten von Constraints zu unterscheiden:

- Core Constraints
- Hard Constraints
- Soft Constraints

Core Constraints sind Regeln, die immer erfüllt sein müssen ohne explizite Angabe.

- Ein Employee darf nicht an zwei Orten gleichzeitig eingeplant sein.
- Ein Participant darf nicht an zwei Orten gleichzeitig eingeplant sein.
- Ein Event darf nicht in einem Raum sein, der vom Raumtyp nicht passt.
- Ein Event darf nicht in einem Raum sein, der keine ausreichende Kapazität hat.

Hard- und Soft-Constraints sind von der Implementierung gleich, die Typen sind im [Constraint-Kapitel](#) erläutert. Hard Constraints sind hierbei Constraints, welche erfüllt werden müssen. Soft Constraints sind Wünsche, die nicht erfüllt werden müssen.

Prozessablauf

Ziel des Algorithmuses ist es, einen optimalen Stundenplan zu generieren. Dies soll vollständig automatisch passieren, um eine Fachkraft abzulösen. Daher ist die Priorität auf dem Erstellen des optimalen Stundenplans, wobei eine effiziente Laufzeit zweitrangig ist.

Es musste entschieden werden, nach welchem Prinzip der Algorithmus zum stoppen kommen soll. Mindestens müssen hier alle Core- und Hard-Constraints erfüllt werden, gewünscht wird auch die Erfüllung der Soft-Constraints.

Aufgrund der NP-Schweren Natur des Stundenplan-Problems lässt sich nicht im Voraus ermitteln, ob ein Soft Constraint erfüllt werden kann oder nicht, versucht der Algorithmus sowohl Hard- als auch Soft-Constraints zu erfüllen.

Beispielsweise ist nicht bekannt, ob in den nächsten wenigen Generationen ein Soft-Constraint erfüllt werden kann, wenn aktuell alle Hard-Constraints schon erfüllt sind. Daher könnte es unsinnig sein, den Algorithmus vorzeitig zu stoppen, mit der Begründung, dass die Hard-Constraints erfüllt sind, da dieser noch weiter optimiert werden kann. Gleichzeitig ist es möglich, dass es astronomisch unwahrscheinlich oder gar unmöglich ist, ein Soft-Constraint zu erfüllen, das in einem aktuellen Stundenplan nicht erfüllt ist. Da der Algorithmus im Dunkeln tappt und der zukünftige Erfolg nicht bestimmbar ist, wurde ein "greedy-approach" angenommen, welcher optimistisch annimmt, dass der Algorithmus optimiert werden kann, bis dieser schließlich optimal ist.

Eine alternative Implementation wäre, den Algorithmus zu stoppen, nachdem alle Hard-Constraints erfüllt sind, ohne als Bedingung die Erfüllung aller Soft-Constraints zu haben. Da dies aber zu weniger optimalen Stundenplänen führt, wird aktuell versucht, alle Constraints zu erfüllen.

Setup

Das Setup für den Endbenutzer liefert ein vollständiges, funktionierendes und getestetes Endprodukt. #

Das Setup für den Entwickler basiert auf der Git-Repository zu dem Algorithmus und kann Features enthalten, die noch nicht vollständig getestet wurden und in dem neuesten Docker Image veröffentlicht wurden. In der Git-Repository ist der vollständige Quellcode zum Algorithmus vorhanden, welcher auch nach Belieben modifiziert werden darf. Zusätzlich sind auch grundlegende Unit-Tests für das Testen des Servers und Erfüllung der Constraints vorhanden.

Endbenutzer Setup

Schritte zum Aufsetzen des Stundenplan Algorithmus als Endbenutzer

1. Docker Engine bereitstellen

Die Docker Engine kann am einfachsten durch das Installieren von Docker Desktop bereitgestellt werden. Dafür wird eine Registrierung bei Docker benötigt.

2. FH-Wedel Git Authentifizierung

Das Docker Image des Stundenplan-Algorithmus wird auf dem Git-Server der Fachhochschule Wedel gehostet, daher kann darauf nur zugegriffen werden, wenn man über einen Account mit entsprechenden Berechtigungen verfügt. Dazu muss folgender Befehl über die Kommandozeile ausgeführt werden:

```
docker login git.fh-wedel.de
```

3. Docker Container Starten

Die Docker Erweiterung „Docker Compose“ wird verwendet um den Container zum laufen zu bringen, hierbei wird statt einem Kommandozeilen-Einzeiler eine Konfigurationsdatei „docker-compose.yml“ angelegt.

```
version: '2'

services:
  algorithm:
    container_name: "stundenplan_algorithm"
    image: git.fh-wedel.de/swp_stundenplan25/genetic_algorithm:1.0.0
    volumes:
      - ./resources:/app/src/python/resources
    ports:
      - "1111:1111"
```

Anschließend kann aus der Kommandozeile heraus der Algorithmus-Server gestartet werden. Voraussetzung ist jedoch, dass sich das Arbeitsverzeichnis in der Kommandozeile im gleichen Ordner befindet, wo auch die „docker-compose.yml“ liegt.

```
docker-compose up -d
```

Entwickler Setup (Mit Docker)

Schritte zum Aufsetzen des Stundenplan Algorithmus als Entwickler mit Docker

1. Docker Engine bereitstellen

Die Docker Engine kann am einfachsten durch das Installieren von Docker Desktop bereitgestellt werden. Dafür wird eine Registrierung bei Docker benötigt.

2. Git Repository Klonen

Zugriff zur Git Repository ist nicht öffentlich zugänglich und erfordert entsprechende Berechtigung auf dem Git-Server der Fachhochschule Wedel. Die Git Repository kann mit folgendem Befehl in der Kommandozeile geklont werden:

```
git clone https://git.fh-wedel.de/SWP_stundenplan25/genetic_algorithm.git
```

3. Docker Container starten

Gestartet wird der Server über mit docker-compose

```
docker-compose up -d
```

Bei Änderungen im Quellcode ist es erforderlich, das Docker Image neu zu bauen, erst dann werden die Änderungen übernommen.

```
docker-compose up -d --build
```

Entwickler Setup (Ohne Docker)

1. Git Repository Klonen

Zugriff zur Git Repository ist nicht öffentlich zugänglich und erfordert entsprechende Berechtigung auf dem Git-Server der Fachhochschule Wedel. Die Git Repository kann mit folgendem Befehl in der Kommandozeile geklont werden:

```
git clone https://git.fh-wedel.de/SWP_stundenplan25/genetic_algorithm.git
```

2. Voraussetzungen prüfen

Es muss mindestens Python mit der Version 3.12 vorhanden sein, testen kann man das in der Kommandozeile:

```
python -V
```

Wenn Python 3.12 vorhanden ist, können die Requirements für Python installiert werden

```
python -m pip install requirements.txt -r
```

3. Starten

Auf **Windows** kann der Server mit folgendem CMD einzelner gestartet werden:

```
cmd /c "set PYTHONPATH=%CD%; %PYTHONPATH% && python -u src\python\server.py"
```

Auf **Linux** kann der Server mit folgendem Shell einzelner gestartet werden:

```
PYTHONPATH=$(pwd) && export PYTHONPATH && python -u src/python/server.py
```

Testen

Der Algorithmus-Server muss laufen, damit die Tests ausgeführt werden können. Die Tests können durch das Script "test.py" in dem "test"-Ordner ausgeführt werden:

```
python test/test.py
```

Eigene Tests können entsprechend dem vorhandenen Muster in "test/units.py" ergänzt werden.

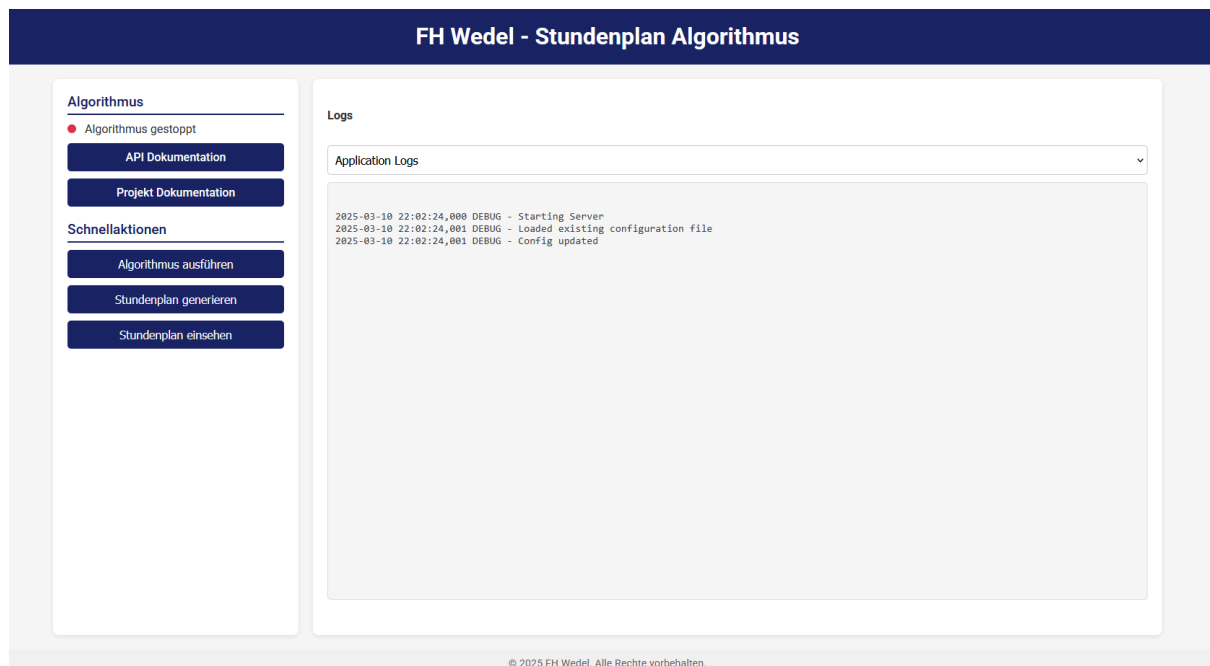
Nutzung

Der Server kann sowohl direkt als REST API angesprochen werden, bietet jedoch noch als Zusatz eine einfache Weboberfläche.

Weboberfläche

Sobald der Algorithmus Server gestartet ist, kann eine simple Benutzeroberfläche im Browser besucht werden

```
http://localhost:1111
```



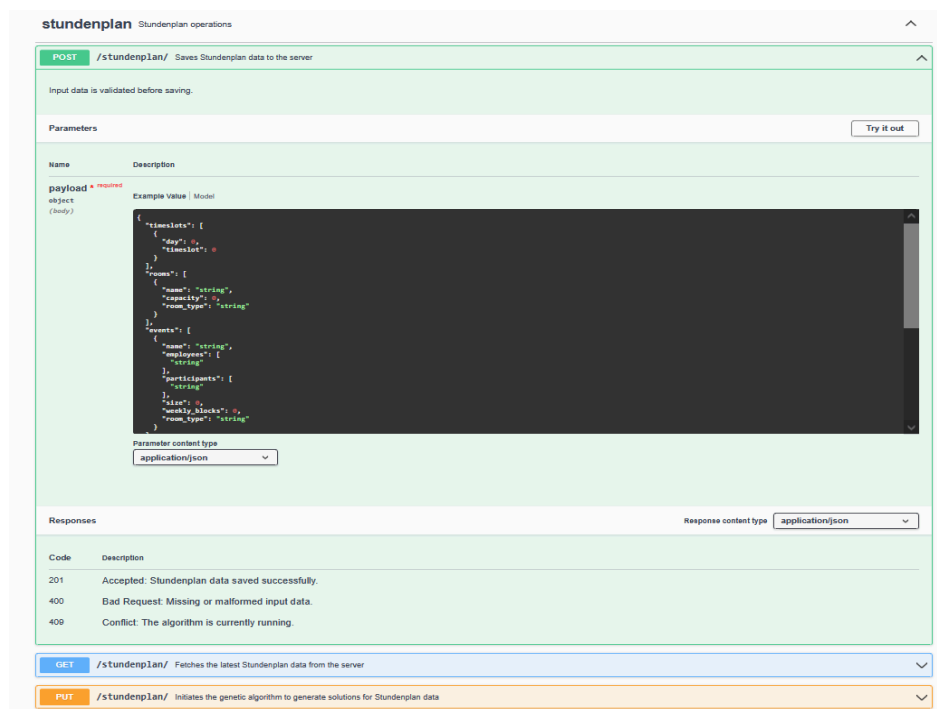
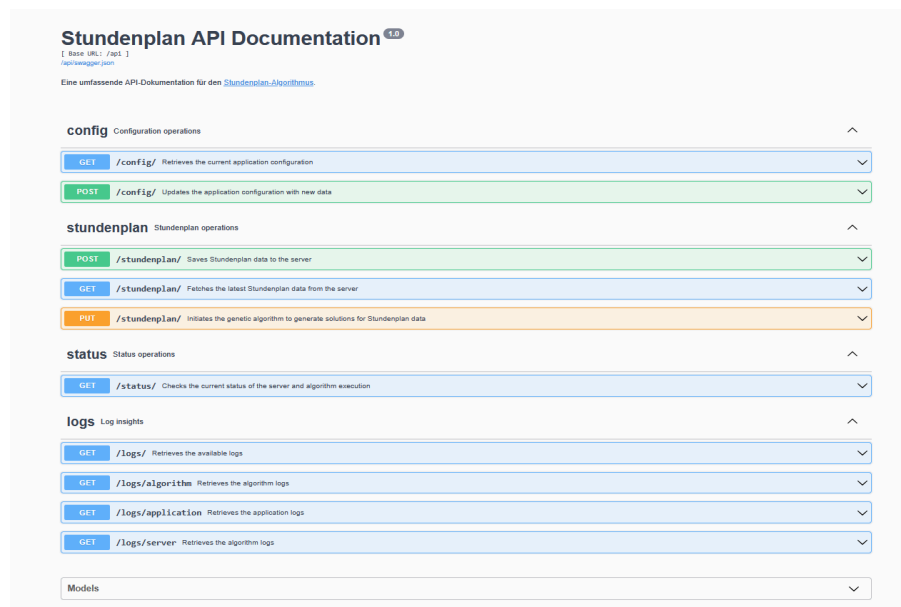
Die Weboberfläche bietet einsicht in die verschiedenen Logs des servers, hierbei werden folgende Logs geboten:

- **Application**
Logs über die Anwendung, wie zum Beispiel neuer gespeicherte Input
- **Algorithm**
Detaillierter Verlauf des Algorithmus zum generieren eines Stundenplans
- **Server**
HTTP Logs über eingehende Anfragen und dem ausgehenden HTTP Code.

API Endpunkte

Die Backend API Endpunkte sind vollständig mit Swagger dokumentiert, hier sind ebenfalls alle unterstützten Ein- und Ausgabedaten modelliert, sowie jede mögliche Antwort des Servers mit den jeweiligen HTTP Codes. Dort sind alle Endpunkte immer aktuell dokumentiert, daher wird es hier ausgelassen, diese redundant zu erläutern.

<http://localhost:1111/api/docs>



Datenmodelle

Atomare Modelle

Atomare Modelle sind die Datenmodelle, welche in anderen Datenmodellen wiederverwendet werden. Als Beispiel wird ein Event sowohl in der Datenbasis als auch im Ergebnis verwendet.

Timeslot

```
{
  "day": integer,
  "timeslot": integer
}
```

Room

```
{
  "name": string,
  "room_type": string,
  "capacity": integer
}
```

Event

```
{
  "name": string,
  "room_type": string,
  "employees": [string],
  "participants": [string],
  "size": integer,
  "weekly_blocks": integer,
}
```

Constraint

```
{
  "id": string,
  "type": string,
  "owner": string,
  "inverted": boolean,
  "fields": {...},
}
```

Core Constraint

```
{
  "employee_conflicts": 0,
  "student_conflicts": 0,
  "room_capacity": 0,
  "room_type": 0
}
```

Datenbasis

Die Datenbasis ist das Datenmodell, welches vom Server empfangen wird, welcher als Grundlage für den Stundenplan dient. Der zugehörige Endpunkt des Servers ist:

```
PUT http://localhost:1111/api/stundenplan
```

```
{
  "timeslots": [timeslot],
  "rooms": [room],
  "events": [event],
  "constraints": {
    "hard": [constraint],
    "soft": [constraint]
  }
}
```

Ergebnis

Das Ergebnis ist ein fertiger Stundenplan mit Informationen zu diesem.

```
GET http://localhost:1111/api/stundenplan
```

```
{
  "data": {
    "timetable": [ (event & timeslot) ],
    "metadata": {
      "fitness": integer,
      "runtime": string,
      "constraints": {
        "core": {
          "fitness": integer,
          "satisfied": core_constraints,
          "unsatisfied": core_constraints,
        }
        "hard": {
          "fitness": integer,
          "satisfied": [constraint],
          "unsatisfied": [constraint]
        }
        "soft": {
          "fitness": integer,
          "satisfied": [constraint],
          "unsatisfied": [constraint]
        }
      }
    },
    "timestamp": string,
    "status": string
  }
}
```

Implementation

Constraints

EmployeeFreeTimeslots	
Nicht Invertiert	Invertiert
Der Employee (owner) möchte nicht zu diesen Timeslots ein Event geplant haben	Der Employee (owner) möchte nur zu den übergebenen Timeslots Events eingeplant haben
<pre>{ "id": "string", "type": "EmployeeFreeTimeslots", "owner": "string", "inverted": boolean "fields": { "timeslots": [{"day": integer, "timeslot": integer}] } }</pre>	

EmployeeSubsequentTimeslots	
Nicht Invertiert	Invertiert
Der Employee (owner) möchte nicht mehr als das übergebene limit an Events in einer Reihe direkt hintereinander eingeplant haben	<i>(undefiniert & nicht implementiert)</i>
<pre>{ "id": "string", "type": "EmployeeSubsequentTimeslots", "owner": "string", "inverted": boolean // ignored "fields": { "limit": integer } }</pre>	

EventDistributeWeeklyBlocks	
Nicht Invertiert	Invertiert
Alle Events mit dem übergebenen Namen müssen an jeweils verschiedenen Tagen eingeplant sein	Alle Events mit dem übergebenen Namen müssen an dem gleichen Tag eingeplant sein
<pre> { "id": "string", "type": "EventDistributeWeeklyBlocks", "owner": "string", "inverted": boolean "fields": { "event": "string" } } </pre>	

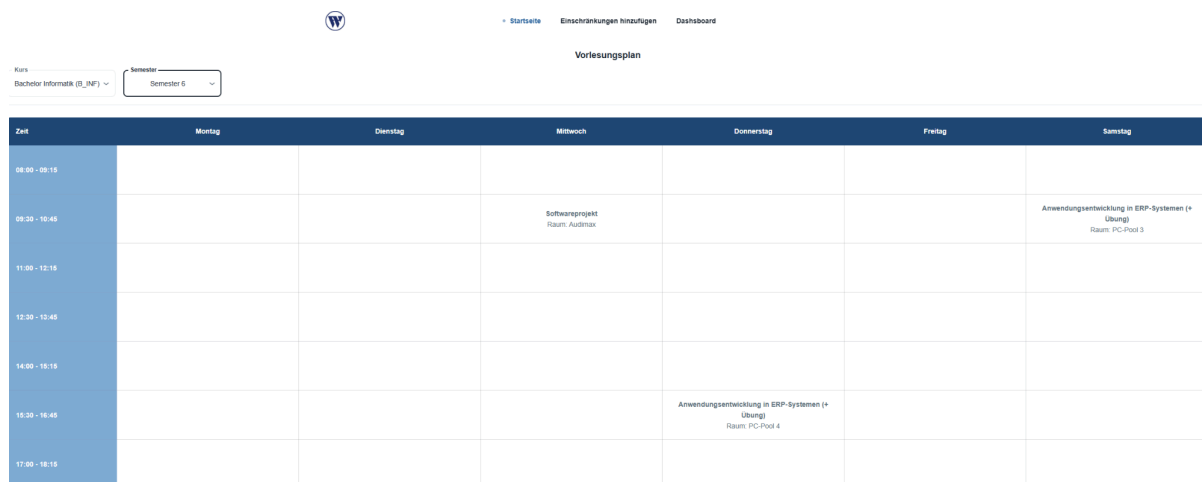
React Frontend

Beschreibung

Das Frontend besteht im Prinzip aus drei unterschiedlichen Bereichen, wo jeweils eine unterschiedliche Gruppe von Endanwendern zugreifen wird.

Stundenplan-View

Diese View ist die Startansicht, in der der aktuelle Stundenplan abgerufen werden kann. Diese Ansicht ist erstmal so aufgebaut, dass sie vorrangig von Studenten genutzt wird. Im Grunde kann nämlich hier nur nach dem Studiengang und dann nach dem entsprechenden Semester gefiltert werden, um den Stundenplan aufzurufen.



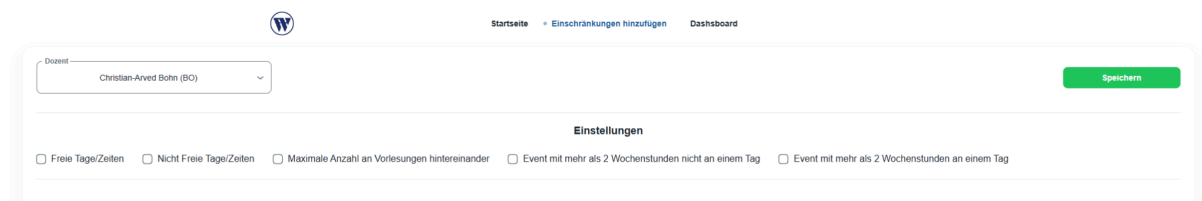
The screenshot shows the 'Stundenplan-View' interface. At the top, there is a navigation bar with a logo and links for 'Startseite', 'Einschränkungen hinzufügen', and 'Dashboard'. Below this, the title 'Vorlesungsplan' is displayed. The main content area features a filter section with 'Kurs' (Bachelor Informatik (B_INF)) and 'Semester' (Semester 6). The central part is a table with columns for the days of the week (Montag to Samstag) and rows for time slots (08:00-09:15 to 17:00-18:15). The table contains the following data:

Zeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag
08:00 - 09:15						
09:30 - 10:45			Softwareprojekt Raum: Audimax			Anwendungsentwicklung in ERP-Systemen (+ Übung) Raum: PC-Pool 3
11:00 - 12:15						
12:30 - 13:45						
14:00 - 15:15						
15:30 - 16:45				Anwendungsentwicklung in ERP-Systemen (+ Übung) Raum: PC-Pool 4		
17:00 - 18:15						

Constraints-View

Dieser View ist der Bereich, wo die Dozenten ihre Einschränkungen angeben können, die dann entsprechend im Algorithmus mit beachtet werden.

Hier kann der Dozent über ein Select sich selbst auswählen und dann für die unterschiedlichen Typen von Constraints seine Angaben machen:



The screenshot shows the 'Constraints-View' interface. At the top, there is a navigation bar with a logo and links for 'Startseite', 'Einschränkungen hinzufügen', and 'Dashboard'. Below this, the title 'Einschränkungen' is displayed. The main content area features a 'Dozent' dropdown menu with 'Christian-Arved Bohn (BO)' selected. To the right of the dropdown is a green 'Speichern' button. Below the dropdown, there are five checkboxes for different constraint types:

- ☐ Freie Tage/Zeiten
- ☐ Nicht Freie Tage/Zeiten
- ☐ Maximale Anzahl an Vorlesungen hintereinander
- ☐ Event mit mehr als 2 Wochenstunden nicht an einem Tag
- ☐ Event mit mehr als 2 Wochenstunden an einem Tag

1. Freie Tage/Zeiten

- Hier können die Zeitslots ausgewählt werden, wo der Dozent keine Vorlesungen halten möchte

2. Nicht freie Tage/Zeiten

- Dieser Typ ist invertiert zu vorherigen Typen. Hier können die Zeitslots ausgewählt werden, wo der Dozent gerne Vorlesungen halten möchte

The screenshot shows a web form for selecting lecture slots. At the top, there is a dropdown menu labeled 'Dozent' with the value 'Christian-Arved Bohn (BO)'. Below this, there are three radio buttons: 'Freie Tage/Zeiten' (checked), 'Nicht Freie Tage/Zeiten' (checked), and 'Maximale Anzahl an Vorlesungen hintereinander' (unchecked). Below the radio buttons, there are two identical rows. Each row has a dropdown menu labeled 'Timeslot', a checkbox labeled 'Zwingend erforderlich' (unchecked), and two buttons labeled '-' and '+'. The form is styled with a light gray background and rounded corners.

3. Maximale Anzahl von Veranstaltungen hintereinander

- Hier kann der Dozent angeben, wie viele Veranstaltungen er maximal hintereinander halten möchte. Das würde dann für die gesamte Woche gelten und ist nicht auf einen bestimmten Tag einschränkbar

Dozent

Christian-Arved Bohn (BO) ▼

☐ Freie Tage/Zeiten
☐ Nicht Freie Tage/Zeiten
☒ Maximale Anzahl an Vorlesungen hintereinander

Maximale Anzahl hinterei...

☐ Zwingend erforderlich

4. Event mit mehr als 2 Wochenstunden nicht an einem Tag

- Wenn der Dozent ein Event hat, welches mehr als 1 Vorlesungsblock in der Woche hat, gibt es hier die Möglichkeit zu sagen, dass beide Blöcke nicht an einem Tag stattfinden sollen

5. Event mit mehr als 2 Wochenstunden an einem Tag

- Dieser Typ ist ebenfalls invertiert zum vorherigen und sagt aus, dass beide Vorlesungsblöcke an einem Tag stattfinden sollen

Dozent

Christian-Arved Bohn (BO) ▼

Einstellungen

☐ Freie Tage/Zeiten
☐ Nicht Freie Tage/Zeiten
☐ Maximale Anzahl an Vorlesungen hintereinander
☒ Event mit mehr als 2 Wochenstunden nicht an einem Tag
☒ Event mit mehr als 2 Wochenstunden an einem Tag

Event mit mehr als 2 Wochenstunden nicht an einem Tag

Events ▼

Events ▼

Event mit mehr als 2 Wochenstunden an einem Tag

Admin-Dashboard-View

Dieser Bereich ist für die Pflege aller wichtigen Daten und ist auch das Herzstück der Anwendung, denn hier hat man auch die Möglichkeit, den Algorithmus direkt zu starten und sich einen Stundenplan zu generieren und einen Report zu sehen.

W

Mitarbeiter

Studiengänge

Veranstaltungen

Räume

Zeitslots

STUNDENPLANUNG

Stundenpläne

Startseite

Einschränkungen hinzufügen

Dashboard

Übersicht: Mitarbeiter

Neuer Mitarbeiter

Alle

Professorin

Lehrbeauftragte(r)

Wissenschaftliches Personal

Verwaltungspersonal

Suchen...

Kürzel	Name	Rolle
AMK	Anna Magdalena Köber	Lehrbeauftragte(r)
AHR	Dirk Ahrens	Dozent(in)
AAH	Alison Alpou-Hoef	Lehrbeauftragte(r)
ANN	Hendrik Aernuth	Dozent(in)
BO	Christian-Arved Bohn	Dozent(in)

Generierte Pläne

Plan generieren

Status abfragen

Zeitpunkt

10.03.2025 15:49:56

Aktueller Status zur Berechnung

Erfolgreich

Konflikte bei den Core Constraints

Core Constraints sind grundlegende Einschränkungen, die in jedem Szenario erfüllt sein müssen. Konflikte hier können darauf hinweisen, dass fundamentale Regeln verletzt wurden.

Konflikt: Dozenten

Anzahl an Dozenten, die an zwei Orten gleichzeitig gescheduled sind

0

Konflikt: Studiengänge

Anzahl an Studiengängen, die an zwei Orten gleichzeitig gescheduled sind

0

Konflikt: Events

Anzahl an Events in einem Raum mit ausreichender Kapazität

0

Konflikt: Raumtypen

Anzahl an Events, die nicht in einem Raum des richtigen Typs stattfinden

0

Konflikte bei den Hard Constraints

Hard Constraints sind zwingende Bedingungen von den Dozenten, die keine Verletzung zulassen. Wenn es hier Konflikte gibt, ist die aktuelle Lösung ungültig und muss überarbeitet werden.

Art des Konflikts	Eigner der Einschränkung	Einschränkung invertiert	Informationen

Setup

Das Setup für den Endbenutzer liefert eine vollständige, funktionierende und getestete Web-Anwendung, die im Browser aufrufbar ist.

Das Setup für den Entwickler basiert auf der Git-Repository zu dem React Frontend. In der Git-Repository ist der vollständige Quellcode zur React Anwendung vorhanden, welcher auch nach Belieben modifiziert werden darf.

Endbenutzer Setup

Schritte zum Aufsetzen des Stundenplan-Frontend als Endbenutzer

4. Docker Engine bereitstellen

Die Docker Engine kann am einfachsten durch das Installieren von Docker Desktop bereitgestellt werden. Dafür wird eine Registrierung bei Docker benötigt.

5. FH-Wedel Git Authentifizierung

Das Docker Image des Stundenplan-Frontends wird auf dem Git-Server der Fachhochschule Wedel gehostet, daher kann darauf nur zugegriffen werden, wenn

man über einen Account mit entsprechenden Berechtigungen verfügt. Dazu muss folgender Befehl über die Kommandozeile ausgeführt werden:

```
docker login git.fh-wedel.de
```

6. Docker Container Starten

Die Docker Erweiterung „Docker Compose“ wird verwendet, um den Container zum Laufen zu bringen, hierbei wird statt einem Kommandozeilen-Einzeiler eine Konfigurationsdatei „docker-compose.yml“ angelegt.

```
version: '2'

services:
  react-frontend:
    container_name: "stundenplan_frontend"
    image: git.fh-wedel.de/swp_stundenplan25/react_frontend:1.0.0
    volumes:
      - REACT_APP_SPRING_URL=http://localhost:8080
    ports:
      - "3000:80"
```

Anschließend kann aus der Kommandozeile heraus das Frontend gestartet werden. Voraussetzung ist jedoch, dass sich das Arbeitsverzeichnis in der Kommandozeile im gleichen Ordner befindet, wo auch die „docker-compose.yml“ liegt.

```
docker-compose up -d
```

Entwickler Setup (Mit Docker)

Schritte zum Aufsetzen des Stundenplan-Frontend als Entwickler mit Docker

4. Docker Engine bereitstellen

Die Docker Engine kann am einfachsten durch das Installieren von Docker Desktop bereitgestellt werden. Dafür wird eine Registrierung bei Docker benötigt.

5. Git Repository Klonen

Zugriff zur Git Repository ist nicht öffentlich zugänglich und erfordert entsprechende Berechtigung auf dem Git-Server der Fachhochschule Wedel. Die Git Repository kann mit folgendem Befehl in der Kommandozeile geklont werden:

```
git clone https://git.fh-wedel.de/SWP_stundenplan25/react_frontend.git
```

6. Docker Container starten

Gestartet kann das Ganze über mit docker-compose

```
docker-compose up -d
```

Bei Änderungen im Quellcode ist es erforderlich, das Docker Image neu zu bauen, erst dann werden die Änderungen übernommen.

```
docker-compose up -d --build
```

Entwickler Setup (Ohne Docker)

4. Git Repository Klonen

Zugriff zur Git Repository ist nicht öffentlich zugänglich und erfordert entsprechende Berechtigung auf dem Git-Server der Fachhochschule Wedel. Die Git Repository kann mit folgendem Befehl in der Kommandozeile geklont werden:

```
git clone https://git.fh-wedel.de/SWP_stundenplan25/react_frontend.git
```

5. Voraussetzungen prüfen

Es muss mindestens Node.js mit der Version 20.11.1 vorhanden sein, testen kann man das in der Kommandozeile:

```
node -v
```

Es befindet sich im Repo eine package.json mit alle Dependencies, welche benötigt werden, diese können über die Kommandozeile installiert werden:

```
npm i oder npm install
```

6. Starten

Starten lässt sich das Projekt lokal über das `start` Script, welches automatisch beim Anlegen eines React Projekts in der `package.json` definiert wird und ausführen lässt sich das mit:

Starten lässt sich das Projekt lokal über das "start" Script, welches automatisch beim Anlegen eines React Projekts in der package.json definiert wird und sich in der Kommandozeile ausführen lässt:

```
npm start
```

Projektstruktur

Die Anwendung folgt einer **modularen Architektur**, um eine klare Struktur, **Skalierbarkeit** und **Wartbarkeit** zu gewährleisten. Jede Funktionalität ist in einem separaten Modul organisiert, was die Code-Wiederverwendbarkeit und Erweiterbarkeit erleichtert.

- **Public Ordner**

Der Public-Ordner enthält alle statischen Dateien, die direkt von der Anwendung verwendet werden, ohne von Webpack verarbeitet zu werden.

- index.html – Haupt-HTML-Datei der Anwendung, in die die React-App eingebunden wird.
- Favicon – Das kleine Symbol, das im Browser-Tab angezeigt wird.
- manifest.json – Konfigurationsdatei für Progressive Web Apps (PWA).
- robots.txt – Steuerung für Suchmaschinen-Crawler.

- **Src Ordner**

Der src-Ordner enthält den gesamten Haupt Quellcode der Anwendung.

1. **api**

Dieses Verzeichnis enthält die API-Service-Funktionen, die für HTTP-Anfragen an das Spring-Backend genutzt werden. Die Kommunikation erfolgt typischerweise über Axios, wobei:

- GET, POST, PUT, DELETE-Requests abstrahiert werden.
- Fehlerhandling zentral geregelt werden kann.
- Authentifizierungs-Token bei jedem Request mitgesendet werden kann.

Dadurch bleibt die API-Integration sauber und modular.

2. **assets**

Hier werden statische Ressourcen verwaltet, darunter:

- Illustrationen – z. B. für Fehlermeldungen oder leere Zustände.

Durch eine zentrale Verwaltung der Assets können Änderungen unkompliziert durchgeführt werden.

3. **auth**

Hier wird die Authentifizierungslogik für einen zukünftigen Login & Benutzerverwaltung vorbereitet.

Typische Inhalte könnten sein:

- Token-Handling (z. B. JWT-Token im localStorage oder sessionStorage speichern).
- Login, Logout, Registrierungs-Logik.

- Authentifizierte Routen, um geschützte Bereiche der Anwendung nur für eingeloggte Nutzer zugänglich zu machen.

Obwohl das Feature derzeit nur angelegt ist, lässt sich das Auth-Modul in Zukunft problemlos erweitern.

4. components

Hier befinden sich alle wiederverwendbaren UI-Komponenten, die an mehreren Stellen innerhalb der Anwendung genutzt werden. Dazu gehören:

- Buttons – Standardisierte Schaltflächen mit einheitlichem Design.
- Modals – Pop-up-Fenster für Bestätigungen oder Warnungen.
- Formulare – Standardisierte Formularelemente wie Inputs oder Dropdowns.
- Tabellen, Karten & Listen – Strukturelle UI-Komponenten zur Darstellung von Daten.

Diese Modularität sorgt für eine konsistente Benutzeroberfläche und erleichtert spätere Änderungen.

5. hooks

Eigene benutzerdefinierte React Hooks, um Logik aus den Komponenten auszulagern und wiederverwendbare Funktionen bereitzustellen. Dazu gehören z. B.:

- useResponsive – Ein Hook zur Überwachung der Bildschirmgröße.
- useBoolean – Ein Hook, um Boolean-Zustände effizient zu verwalten (true/false Umschaltung für UI-Elemente wie Modals).

Hooks ermöglichen eine saubere Trennung zwischen Geschäftslogik und UI.

6. layouts

Hier sind die verschiedenen Seitenlayouts organisiert. Diese steuern die übergeordnete Struktur einer Seite, z. B.:

- Compact – Ein kompaktes Layout für Fehlerseiten oder Hinweise.
- Dashboard – Layout für das Haupt-Dashboard mit Navigation und Sidebar.
- Main – Standard-Layout für allgemeine Seiten.

Layouts sorgen dafür, dass die UI strukturiert bleibt und zentrale UI-Elemente (wie Navigation oder Footer) nicht mehrfach in einzelnen Seiten definiert werden müssen.

7. pages

Alle Hauptseiten der Anwendung sind hier organisiert. Sie sind weiter unterteilt in:

- Fehlerseiten (404, 500)
- Dashboard-Bereich – Die zentrale Verwaltungsseite der Anwendung.
- Hauptseiten – Seiten für Kernfunktionen der App.

Jede Seite enthält ihre eigene Logik und nutzt die Komponenten aus dem components/-Ordner.

8. routes

Hier wird das Routing der Anwendung definiert, mit Hilfe von react-router.

- Jede Route ist einer page-Komponente zugeordnet.
- Geschützte Routen (z. B. Login erforderlich) können hier verwaltet werden.
- Routen mit dynamischen Parametern (z. B. /user/:id) können hier konfiguriert werden.

Durch eine zentrale Routing-Datei bleibt die Navigation der App übersichtlich und erweiterbar.

9. theme

Dieses Verzeichnis enthält die Design- und Stildefinitionen der Anwendung, darunter:

- Farbpaletten – Einheitliche Farben für Buttons, Hintergründe und Texte.
- Fonts & Typografie – Vorgaben für Schriftarten und -größen.
- Overrides für MUI-Komponenten – Falls Material UI (MUI) genutzt wird, können hier Standardstile überschrieben werden.

Dadurch bleibt das Design konsistent und anpassbar.

10. utils

Hier befinden sich Hilfsfunktionen, die immer wieder in der Anwendung benötigt werden, darunter:

- Formatierungsfunktionen (z. B. Datumsformatierung, Währungsdarstellung).
- Konvertierungsfunktionen (z. B. Umrechnung von Einheiten).

Durch diese Hilfsfunktionen wird die Codebasis sauber gehalten, indem sich wiederholende Logik ausgelagert wird.

11. App.js

Dies ist die Hauptkomponente der React-App, in der:

- Der Router für die Navigation definiert wird.
- Layouts und globale Provider (z. B. ThemeProvider, AuthContext) eingebunden werden.
- Die Hauptlogik der Anwendung zusammenläuft.

Die App.js ist das zentrale Bindeglied zwischen UI, Logik und Routing.

12. index.js

Der Einstiegspunkt der Anwendung. Hier wird:

- Die App mit ReactDOM.createRoot in das div in der index.html gerendert.
- Globale Provider eingebunden.
- Die React-App initialisiert.

Die index.js verbindet React mit der HTML-Struktur und startet die Anwendung

Spring

Spring-Projektaufbau

Der Aufbau des Spring-Projekts unterteilt sich in Controller, Configurations, Components, Repositories, Request-Models und Rest-Data-Models.

Spring Controller:

- In den Controller-Klassen des Packages "controller" werden Endpunkte bereitgestellt.
- Die Klassen verarbeiten HTTP-Anfragen und steuern den Datenfluss zwischen React-Client und Python-Backend.
- Annotiert mit `@RestController` und dem Path.

Beispiel:

- Die Klasse "AlgorithmController" deckt alle Pfade ab, die mit der Basis-URL + "/algorithm" beginnen.
- Die Methode "checkLecturePlanStatus()" ist mit `@GetMapping("/status")` annotiert.
- Wird also durch den Aufruf GET Basis-URL + "/algorithm" + "/status" getriggert und liefert den Status bzw. den fertigen Plan.

Spring Configuration:

- In diesem Projekt gibt es eine "RestTemplateConfig", welche RestTemplates konfiguriert,
- die zum Abschieken von REST-Anfragen an das Python-Backend genutzt werden.
- Die Klasse verwaltet zentrale Einstellungen und Beans (von Spring gemanagte Objekte).
- Annotiert mit `@Configuration` und kann Beans mit `@Bean` bereitstellen.

Beispiel:

- "RestTemplateConfig" erstellt eine RestTemplate-Bean, sodass RestTemplates von Spring gemanagt werden können.

Spring Component:

- In diesem Projekt gibt es eine "AlgorithmComponent".
- Die Klasse ist eine generische Bean, die von Spring verwaltet wird.
- Annotiert mit `@Component`, wodurch sie automatisch als Bean registriert wird.
- Components werden für allgemeine Dienste oder Hilfsklassen verwendet, die nicht in `@Service`, `@Repository` oder `@Controller` fallen.

Beispiel:

- "AlgorithmComponent" gibt durch den Aufruf der Methode "getBackendUrl()" den URL wieder, der in der Datei "application.properties" steht.
- Dieser entspricht dem Basis-URL des Python-Backends.

Spring Repository:

- In dem Package "repository" finden sich mehrere Repositories wieder.
- Ein Repository entspricht einer Datenbanktabelle (In MongoDB: Collection).
- Die Interfaces bilden eine Schnittstelle zur Datenbank und ermöglichen CRUD-Operationen.
- Annotiert mit `@Repository` müssen sie nicht implementiert werden.
- Über JPA können Methoden mit SQL-artigen Namen definiert werden, die von Spring automatisch implementiert werden.

Beispiel:

- "ConstraintRepository.findByIsHard(Boolean isHard)" wird von Spring implementiert und findet alle Constraints in der Tabelle "Constraint", die das Attribut "isHard" auf den Wert "True" gesetzt haben.

Request Model:

- In dem package "request_model" finden sich alle Klassen wieder, die Request- oder Response-Bodys entsprechen.
- Diese Klassen bieten also Datenstrukturen für HTTP-Requests.
- Wird mit `@RequestBody` als Methodenargument in Controllern-Klassen verwendet.

Beispiel:

- In der Methode "patchLecturePlan()" im "AlgorithmController" wird als Request-Body ein Objekt des Typs "SemesterDto" vom Frontend erwartet.

Rest Data Model:

- Das package "rest_data_model" enthält alle Klassen, die in Repositories verwaltet werden.
- Jedes Objekt einer solchen Klasse entspricht also einer Zeile (In MongoDB: Document) in der Datenbanktabelle.
- Wird von den Controller-Klassen zurückgegeben und enthält Daten, die der Client benötigt.

Beispiel:

- "EventRepository" enthält Objekte vom Typ "Event". Wobei jede Zeile einem "Event" entspricht.

Spring API Übersicht

Für jede Rest Data Model Klasse bzw. für jedes entsprechende Repository werden in den jeweiligen Controller-Klassen Endpunkte bereitgestellt, um die CRUD-Operationen auszuführen (GET all, GET by ID, PUT, POST, DELETE). Die einzige Ausnahme bildet der generierte Plan, hier kann nur der letzte, erste oder nach ID abgefragt werden (PUT, POST, DELETE sind nicht erlaubt).

Über den "AlgorithmController" kann eine Generierung angestoßen werden (Daten an Backend schicken und Generierung starten) indem dem PATCH "{Basis-URL}\algorithm\start" aufgerufen wird.

Über GET "{Basis-URL}\algorithm\status" wird der Status der Generierung abgefragt und bei erfolgreicher Beendigung der erstellte Plan zurückgeliefert.

Datenbank

MongoDB

MongoDB ist eine NoSQL-Datenbank, die Daten in JSON-ähnlichen Dokumenten speichert, anstatt in Zeilen wie bei relationalen Datenbanken. Diese Dokumente bestehen aus Schlüsseln und Werten, wobei Werte auch Arrays oder verschachtelte Dokumente sein können.

Dadurch ist MongoDB schemalos, das heißt, Dokumente in derselben Collection (entspricht einer Tabelle) können unterschiedliche Strukturen haben. Daten werden in Collections organisiert, die wiederum in Datenbanken liegen.

MongoDB nutzt Indexes, um Abfragen zu beschleunigen, und ermöglicht leistungsstarke Aggregation für komplexe Datenanalysen. Die Kommunikation erfolgt meist über CRUD-Operationen (Create, Read, Update, Delete) über eine JSON-basierte Abfragesprache.

Unsere Datenbank

Im Folgenden eine Liste unserer Collections:

Courses:

Enthält Studiengänge mit einer Abkürzung, einem vollständigen Namen und einer Studiendauer in Semestern.

Events:

Beschreibt Veranstaltungen mit Name, Dozenten, teilnehmenden Studiengängen, Raum- und Zeitbedarfen sowie weiteren Eigenschaften wie Größe und Typ.

Lecturers:

Enthält Dozierende mit Abkürzung, Vor- und Nachname, Typ und ihren spezifischen Constraints.

Plans:

Speichert Planungsdaten (das Data-Objekt), eine ID, den Zeitstempel der Erstellung, sowie den Status.

Rooms:

Enthält Räume mit ID, Namen, Abkürzung, Raumtyp und Kapazität.

Timeslots:

Speichert verfügbare Zeitfenster mit Tag, Start- und Endzeit sowie einer **active**-Flag, die angibt, ob der Zeitslot genutzt werden darf.

Endprodukt

Mit der folgenden "docker-compose.yml" kann das vollständige Endprodukt gestartet werden, hierbei werden die 4 Docker Container gestartet.

```
version: "3.8"

services:
  mongodb:
    image: git.fh-wedel.de/swp_stundenplan25/mongo_db:1.0.0
    ports:
      - "27017:27017" # MongoDB-Port wird exponiert
    environment:
      MONGO_INITDB_DATABASE: swp_stundenplan # Initiale Datenbank

  algorithm-backend:
    image: git.fh-wedel.de/swp_stundenplan25/genetic_algorithm:1.0.0
    ports:
      - "1111:1111"

  react-frontend:
    image: git.fh-wedel.de/swp_stundenplan25/react_frontend:1.0.0
    environment:
      - REACT_APP_SPRING_URL=http://localhost:8080
    ports:
      - "3000:80"

  spring-backend:
    image: git.fh-wedel.de/swp_stundenplan25/spring_backend:1.0.0
    environment:
      - ALGORITHM_URL=http://algorithm-backend:1111 # Algorithmus Schnittstelle
      - SPRING_DATA_MONGODB_HOST=mongodb # MongoDB-Host (Docker-Service-Name)
      - SPRING_DATA_MONGODB_PORT=27017 # MongoDB-Port
      - SPRING_DATA_MONGODB_DATABASE=swp_stundenplan # MongoDB Datenbank Name
    ports:
      - "8080:8080" # Exponiert den Spring Boot-Port
    depends_on:
      - mongodb # Spring-Backend startet erst, wenn MongoDB läuft
      - algorithm-backend

volumes:
  mongodb-data:
```