

Studienplanung als Generierung von Workflows mit Compliance-Anforderungen: Planerstellung und Visualisierung

Entwurfsdokumentation

Nada Chatti
Daniel Jungkind
Hannes Kuchelmeister
Ulrike Rheinheimer
Paul Samuel M. Teuber
Tim Niklas Uhl

11. Januar 2017

Inhaltsverzeichnis

1. Einleitung	4
2. Aufbau	4
2.1. Architektur	4
2.1.1. Server	5
2.1.2. Client	5
2.1.3. Hinweis	6
3. Datenbankbeschreibung	7
3.1. Moduldaten	7
3.1.1. Enumwerte	8
3.2. Benutzerdaten	8
3.2.1. Enumwerte	10
3.3. Login-Daten	10
4. REST-Webservice-Spezifikation	12
4.1. Notation	12
4.2. Authentifizierung	12
4.3. Atomare Werte	16
4.4. Spezifikation der JSON-Datenklassen	18
4.5. Zugriffsstruktur	23
4.6. Spezifikation der HTTP-Statuscodes	26
5. Verwendete Entwurfsmuster	27
6. Paketbeschreibungen	28
6.1. Package client.model	28
6.2. Package client.model.modules	30
6.3. Package client.model.plans	31
6.4. Package client.model.system	32
6.5. Package client.model.user	33
6.6. Package client.router	34
6.7. Package client.storage	35
6.8. Package client.view.components.filter	36
6.9. Package client.view.components.uielement	37
6.10. Package client.view.components.uipanel	38
6.11. Package client.view	39
6.12. Package client.view.subview	40
6.13. Package server.filter	41
6.14. Package server.generation	42
6.15. Package server.generation.objectivefunction	43
6.16. Package server.generation.standard	44

6.17. Package server.model	45
6.17.1. Package server.model.moduledata	45
6.17.2. Package server.model.userdata	47
6.18. Package server.pluginmanager	49
6.19. Package server.rest.authorization.endpoint	50
6.20. Package server.rest	51
6.21. Package server.verification	52
6.22. Package server.verification.standard	53
7. Klassenbeschreibungen	53
7.1. Übersicht	53
8. Ablaufbeschreibungen	136
8.1. Generierungs-Algorithmus	136
8.1.1. Einstieg	136
8.1.2. Evolutionärer Algorithmus	136
8.1.3. Zufällige Generierung	136
8.1.4. Zufällige Modifizierung	137
8.1.5. Kontraktion der topologischen Sortierung	137
8.2. Abläufe	138
8.3. Beispiele für REST-Kommunikation	142
9. Klassenindex	144
9.1. Client-Klassen	144
9.2. Server-Klassen	145
10. Gestrichene optionale Anforderungen	146
11. Implementierungsplan	146
11.1. Server-Implementierung	147
11.2. Client-Implementierung	149
Anhang	151
A. Abbildungen	151

1. Einleitung

Das Entwurfsdokument des Projekts „Studienplanung als Generierung von Workflows mit Compliance-Anforderungen: Planerstellung und Visualisierung“ baut auf das zuvor verfasste Pflichtenheft auf. Es dient der Strukturierung und genauen Planung der Implementierung des Projekts. Hierzu wurde zunächst ein umfassendes Klassendiagramm erstellt. Auf diesem basierend werden alle Pakete, Klassen, Methoden und Attribute ausführlich erklärt. In weiteren Kapiteln wird die Systemarchitektur, die Datenbankstruktur und der REST-Webservice beschrieben. Zum besseren Verständnis sind Ablaufbeschreibungen in Form von Sequenzdiagrammen angefügt. Zusätzlich wird auf gestrichene optionale Anforderungen aus dem Pflichtenheft und den Implementierungsplan eingegangen.

2. Aufbau

2.1. Architektur

Wie bereits im Pflichtenheft angeführt, haben wir uns für eine klare Trennung von Client und Server entschieden, um somit eine höchstmögliche Modularität zu gewährleisten. So können Client- und Server-Anwendung auf getrennten Servern laufen. Die Kommunikation erfolgt über REST-Schnittstellen, was auch in Zukunft die Entwicklung einer App ermöglicht, ohne die Server-Anwendung anpassen zu müssen.

Im Folgenden werden die Architekturen von Server und Client näher beschrieben.

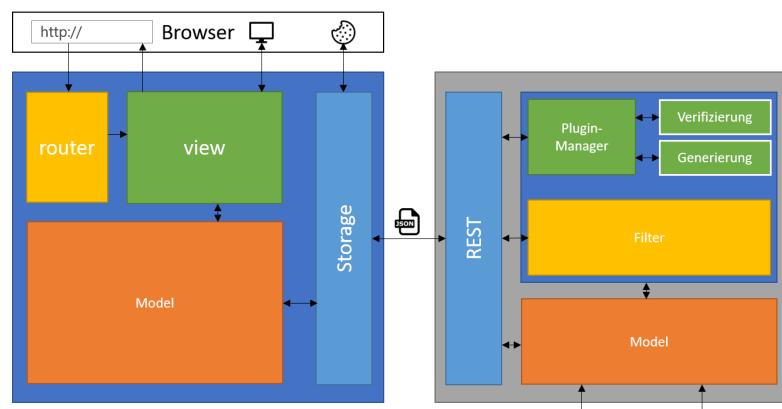


Abbildung 1: Systemarchitektur

2.1.1. Server

Der Server besteht aus zwei zentralen Komponenten: Dem Paket `rest`, sowie dem Paket `model`.

Das Paket `rest` beinhaltet die Schnittstellen zur Außenwelt, denn hier sind alle vorhandenen REST-Webservices der Server-Anwendung in Form von Ressourcen-Klassen mit entsprechenden Methoden implementiert. Des Weiteren beinhaltet das Paket Komponenten zur Erzeugung von JSON-Objekten, sowie zur Zugriffssteuerung auf die Webservices.

Das Paket `model` hingegen stellt die Schnittstelle zum Persistence-Layer bereit. Es enthält zum einen Modellklassen, welche Datenbankeinträge objektorientiert abbilden. Diese Modellklassen wurden in Form einfacher Java-Objekte entworfen, welche nur über Getter und Setter für ihre Attribute verfügen (sogenannte *POJOs*, Plain Old Java Objects). Die Modellklassen sind in Pakete zur Modellierung von Modulen, sowie zur Modellierung von Nutzerdaten und Studienplänen aufgeteilt. Zum anderen verfügt jedes dieser Modellierungspakete über ein Unterpaket zum eigentlichen Datenbankzugriff über sogenannte *Data-Access-Objects* (DAOs), welche Methoden zum Lesen und Schreiben der Datenbank bereitstellen. Diese sind in Form von Schnittstellen ausgelegt. Die konkrete Implementierung dieser verwendet *JBoss Hibernate* für den Datenbankzugriff, jedoch kann sie jederzeit durch eine andere Datenbankschnittstelle ersetzt werden. Das Paket `filter` enthält eine Kompositum-basierte Modellierung von Modulfiltern, welche im Zusammenspiel mit den DAOs den selektiven Zugriff auf Module ermöglichen.

Das Paket `pluginmanager` verwendet die *Java ServiceLoader API*, um dynamisch Generierungs- und Verifizierungstools zu laden. Hierfür müssen diese in Form von Jar-Dateien dem Klassenpfad hinzugefügt werden, wodurch ein Austausch von Tools ohne Änderungen an der Anwendung möglich ist.

Die Pakete `generation` und `verification` enthalten die zu implementierenden Schnittstellen für neue Service Provider, sowie einfache Implementierungen eines Studienplan-Generators und -Verifizierers. Das Paket `generation` enthält darüber hinaus noch eine API zur Modellierung von Zielfunktionen, welche von Generatoren verwendet werden können.

2.1.2. Client

Der Client besteht aus einer leicht abgewandelten Model-View-Controller-Architektur (MVC-Architektur), die auf Backbone.js [3] basiert. Das System besteht aus vier Haupt-Paketen: Storage, Model, View und Router.

Storage übernimmt die Kommunikation mit dem REST-Webservice sowie die Speicherung in Cookies. Somit bildet Storage die Zugriffsschicht für das Model-Paket.

Model bildet die Daten aus den Cookies und dem REST-Webservice auf Klassen ab und

Klasse	Paket
<i>MultivaluedMap<T1,T2></i>	javax.ws.rs.core
<i>ResourceConfig</i>	org.glassfish.jersey.server
<i>JSONObject</i>	javax.json
<i>ContainerRequestFilter</i>	javax.ws.rs.container
<i>GetParameters</i>	siehe Kap. 3.5 Zugriffsstruktur
<i>Condition</i>	java.util.concurrent.locks

Tabelle 1: Externe Klassen

stellt Möglichkeiten zum Abruf sowie zur Speicherung zur Verfügung, die auf Storage basieren.

View zeigt die Daten aus dem Model an und aktualisiert im Fall einer Änderung des Models die Anzeige. Auch fängt das View-Paket Interaktionen mit der Benutzeroberfläche (wie Klicks oder Drag and Drop) ab und verarbeitet diese entsprechend.

Router übernimmt schließlich die Aufgaben des Controllers in klassischen MVC-Architekturen. Der Router wird bei Änderungen des URI Fragments aufgerufen und kann so beim Wechsel auf eine andere Seite der WebApp die neuen View-Klassen initialisieren und die zugehörigen Inhalte anzeigen.

2.1.3. Hinweis

Hinweis auf blaue Klassen Im Klassendiagramm werden externe Klassen, die aus verschiedenen Paketen importiert werden sollen, in blau gefärbt, um sie von den zum System gehörigen Klassen zu unterscheiden. Dazu gehören die Klassen, die in Tabelle 1 erwähnt wurden.

3. Datenbankbeschreibung

Die Produktdaten werden in zwei getrennten Datenbanken gespeichert, eine Datenbank für Moduldaten und eine für Nutzerdaten. Dies ermöglicht es, in Zukunft eine andere Schnittstelle zum Modulhandbuch zu verwenden, wie beispielsweise eine Anbindung an das Vorlesungsverzeichnis.

Im Folgenden wird deshalb der Aufbau beider Datenbanken getrennt beschrieben. Zusätzliche Informationen, die nicht aus den Entity-Relationship-Diagrammen hervorgehen, sind in Form von Kommentaren dargestellt. Die Datentypen der Attribute gehen aus folgender Farbkodierung hervor:

Integer-Ganzzahl	
Zeichenkette	
Boolean-Wert	
Enum	

Tabelle 2: Farbkodierung der Attribute

3.1. Moduldaten

Ein Modul besitzt zwei eindeutige Schlüssel. Als Primärschlüssel besitzt es eine fortlaufend nummerierte numerische ID. Zusätzlich wird ein varchar identifier gespeichert, der nicht fortlaufend nummeriert ist und den die REST-Schnittstellen für den Modulzugriff verwenden. Hierdurch wird sichergestellt, dass ein Nutzer nicht einfach „durch Ausprobieren“ auf beliebige Module zugreifen kann.

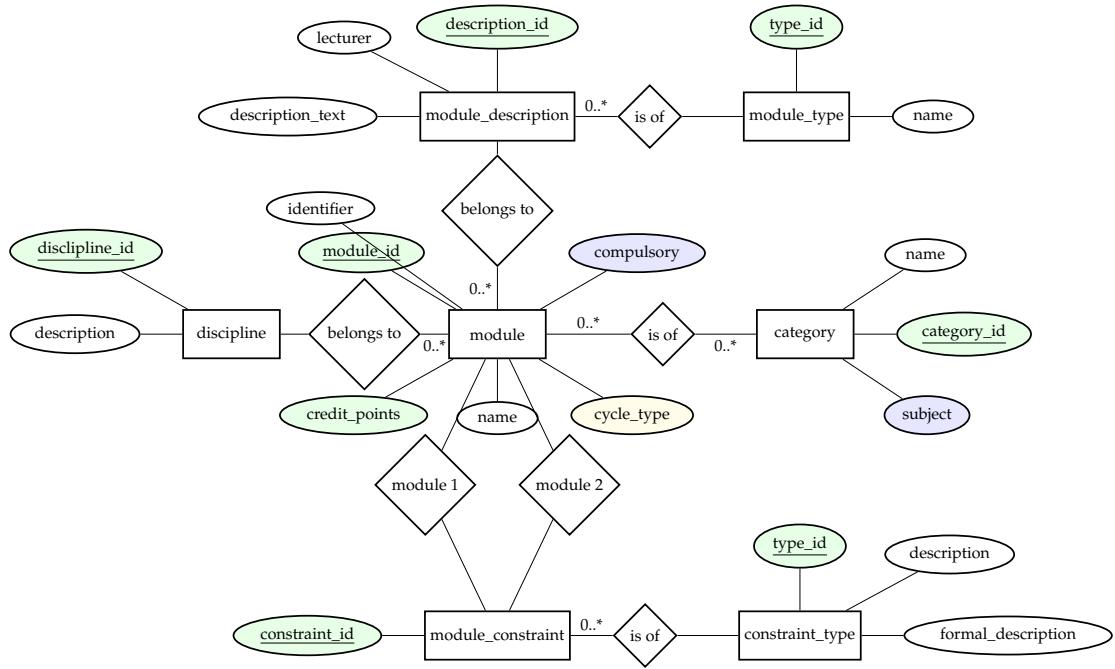


Abbildung 2: ER-Diagramm Moduldaten

3.1.1. Enumwerte

Tabellenspalte	mögliche Werte
<code>cycle_type</code>	winter_term, summer_term, both

3.2. Benutzerdaten

Die Entität `user` besitzt zwei eindeutige Schlüsselelemente. Als Primärschlüssel besitzt ein `user` eine `user_id`. Des Weiteren ist auch das Attribut `name` eindeutig. Ein Plan ist eindeutig über seine `plan_id`, sowie über seinen `identifier` ansprechbar. Da die REST-Schnittstelle zum Zugriff den `identifier` verwendet, wird sichergestellt, dass nicht „durch Ausprobieren“ auf einen beliebigen Plan zugegriffen werden kann.

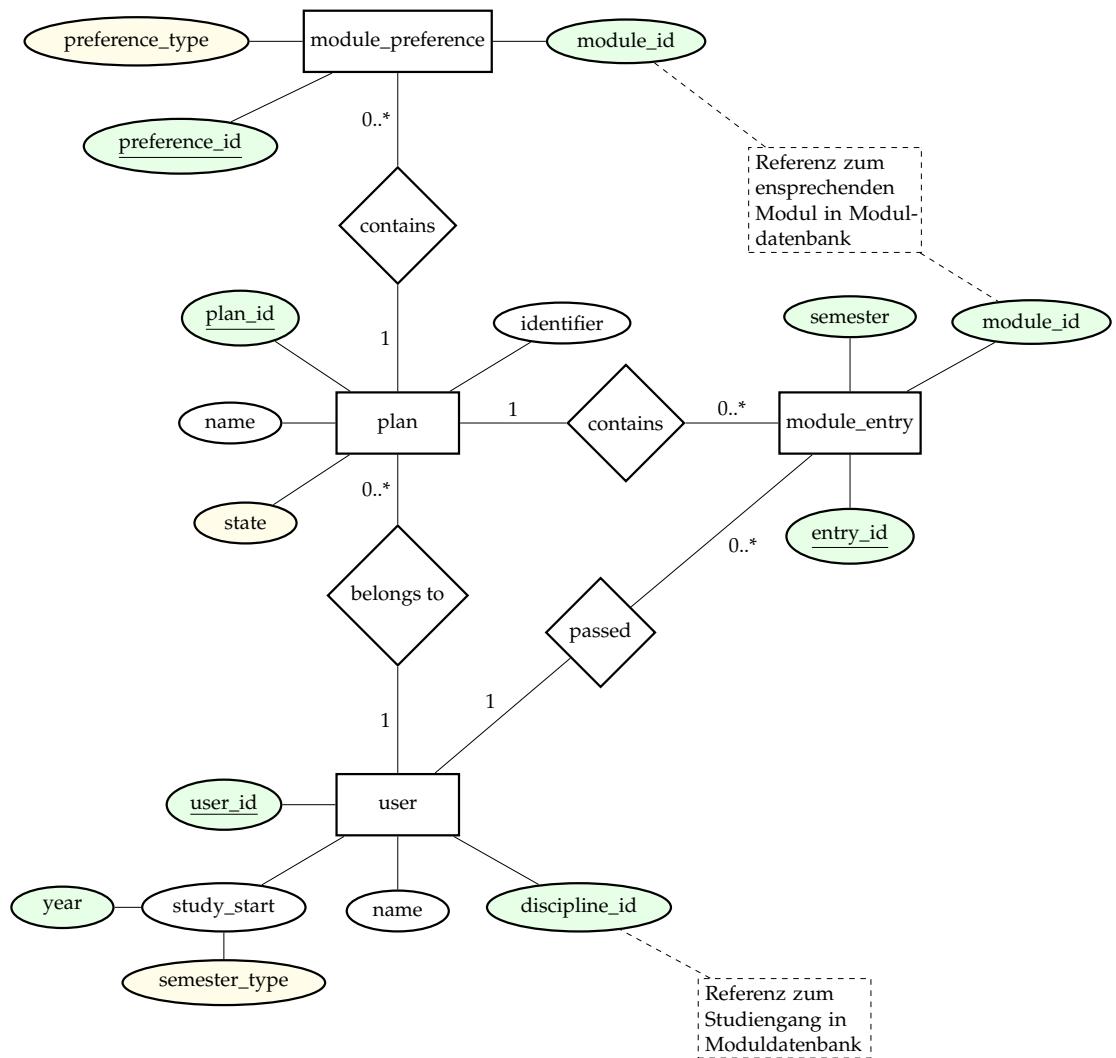


Abbildung 3: ER-Diagramm Benutzerdaten

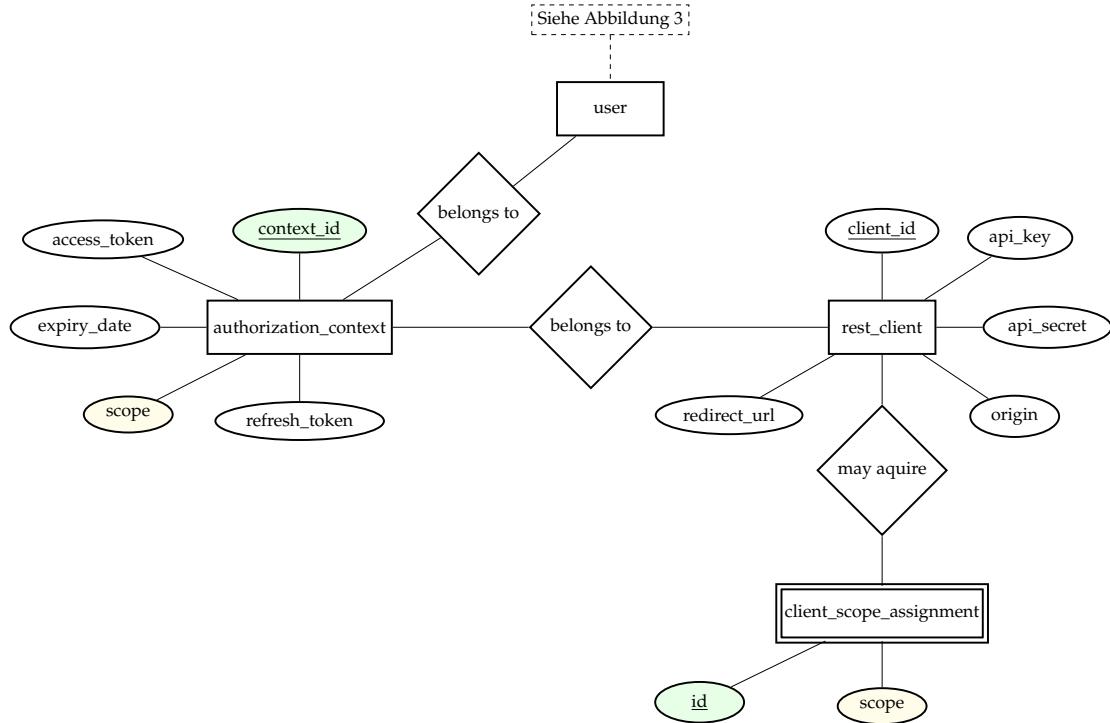


Abbildung 4: ER-Diagramm Authentifizierungsdaten

Eine Beschreibung der Attribute in Abbildung 4 findet sich in Kapitel 4.2.

3.2.1. Enumwerte

Tabellenspalte	mögliche Werte
preference_type	positive, negative
state	not_verified, valid, invalid
semester_type	winter_term, summer_term

3.3. Login-Daten

Die für den rudimentären, im Paket `server.rest.authorization.endpoint` implementierten Identity Provider benötigten Daten werden in der Benutzerdatenbank in

der Tabelle `login_information` gespeichert. Diese besitzt die Spalten `id` (Primärschlüssel), `username` und `password`.

4. REST-Webservice-Spezifikation

4.1. Notation

Atomare Werte werden mittels $\langle Name \rangle$ angegeben, wobei $Name$ ein eindeutiger Bezeichner für den Wert ist.

JSON-Datenklassen werden mittels $\langle\!\langle Name \rangle\!\rangle$ angegeben, wobei $Name$ ein eindeutiger Bezeichner für die JSON-Datenklasse ist.

JSON-Objekte, die nur eine Teilmenge aller Attribute einer JSON-Datenklasse $\langle\!\langle Name \rangle\!\rangle$ enthalten, werden mittels $\langle\!\langle Name [prop_1, \dots, prop_n] \rangle\!\rangle$ angegeben, wobei $prop_1, \dots, prop_n$ die enthaltenen Attribute benennen.

4.2. Authentifizierung

Zugrundeliegende Spezifikation Die Authentifizierung am REST-Webservice erfolgt über eine OAuth-2.0-Schnittstelle nach RFC 6749 (siehe [1]). Für das momentane System ist hierbei nur die Authentifizierung über *Implicit-Grant* [1, Kap. 4.2] notwendig, jedoch sollte das System mindestens um *Authorization-Code-Grant* [1, Kap. 4.1] und *Resource-Owner-Password-Credentials-Grant* [1, Kap. 4.3] erweitert werden können, weshalb die Basis für diesen Authentifizierungstyp ebenfalls implementiert wird. Bei dem Versuch, eine Authentifizierung mittels *Authorization-Code-Grant* oder *Resource-Owner-Password-Credentials-Grant* durchzuführen, wird dann eine Fehlermeldung zurückgegeben.

Die zugrundeliegende Spezifikation verteilt klare Rollen an die verschiedenen Akteure, welche an der Interaktion der Systeme beteiligt sind. In Tabelle 3 ist angegeben, welches Subsystem die jeweiligen Rollen bei der OAuth-Authentifizierung in unserem System übernimmt.

Rolle	Subsystem
<i>resource owner</i>	Benutzer
<i>resource server</i>	REST-Webservice
<i>client</i>	Aktuell ausschließlich die WebApp. Zukünftig vielleicht weitere Systeme, welche auf den REST-Webservice zugreifen
<i>authorization server</i>	REST-Webservice
<i>user agent</i>	Web-Browser des Benutzers

Tabelle 3: Rollen in der OAuth 2.0 Spezifikation

Name	Beschreibung
<i>api_key</i>	Eindeutige, öffentliche ID des Klienten. Beginnt mit dem Präfix „key-“
<i>api_secret</i>	Nur dem Klienten bekannte Kennung. Beginnt mit dem Präfix „secret-“
<i>scope</i>	Berechtigungen (auch mehrere), die der Klient anfordern darf. Vorerst immer nur „student“
<i>redirect_url</i>	URL, an welche beim <i>Implicit-Grant</i> bzw. <i>Authorization-Code-Grant</i> weitergeleitet wird.
<i>origin</i>	Domains, von welchen aus der Klient auf die Ressourcen zugreifen darf. Wird als Regulärer Ausdruck angegeben. Dieser Wert wird bei Anfragen mit Hilfe des <i>Referer-Header</i> überprüft.

Tabelle 4: Daten eines vertrauenswürdigen Klienten

Registrierung der Klienten Klienten sind bei der Authentifizierung alle auf den REST-Webservice zugreifende Systeme, welche sich dafür im Namen eines Nutzers beim REST-Webservice authentifizieren.

Ein Beispiel für einen solchen Klienten ist die WebApp.

Alle Klienten werden in der Datenbank gespeichert.

Wir unterscheiden zwischen zwei Typen von Klienten: *Vertrauenswürdige* sowie *öffentliche* Klienten. Die Definitionen hierzu finden sich auch in [1, Kap. 2.1]. Bei der WebApp handelt es sich auf Grund der Zugänglichkeit des Codes und der Daten um einen öffentlichen Klienten.

Vertrauenswürdiger Klient Für einen vertrauenswürdigen Klienten werden die Informationen aus Tabelle 4 gespeichert.

Öffentlicher Klient Für einen öffentlichen Klienten werden die Informationen aus Tabelle 5 gespeichert.

Schnittstellen Für die Authentifizierung werden die Schnittstellen aus Tabelle 6 verwendet. Beim Login wird hierbei der Web-Browser des Benutzers, welcher die Web-App geöffnet und auf den Button „Login“ geklickt hat, an den URI /auth/login des REST-Webservice weitergeleitet. Auf dieser Seite kann sich der Nutzer authentifizieren. Anschließend wird er auf die Hauptseite der Web-App weitergeleitet. Die Kommunikation der Authentifizierungsdaten zwischen Klient und REST-Webservice verläuft über den *Query* [2, Kap 3.4] bzw. das *Fragment* [2, Kap. 3.5] des URI.

Name	Beschreibung
api_key	Eindeutige, öffentliche ID des Klienten. Beginnt mit dem Präfix „key-“
api_secret	NULL (wird nicht benötigt)
scope	Berechtigungen (auch mehrere) die der Klient anfordern darf. Vorerst immer nur „student“
redirect_url	URL an welche beim <i>Implicit-Grant</i> weitergeleitet wird.
origin	Domains, von welchen aus der Klient auf die Ressourcen zugreifen darf. Wird als Regulärer Ausdruck angegeben. Dieser Wert wird bei Anfragen mit Hilfe des <i>Referer-Header</i> überprüft.

Tabelle 5: Daten eines öffentlichen Klienten

Methode	URL	Beschreibung
GET	/auth/login	Der in RFC 6749, Kapitel 3.1 definierte <i>Authorization Endpoint</i>
POST	/auth/token	Der in RFC 6749, Kapitel 3.2 definierte <i>Token Endpoint</i>
GET	/auth/logout	Devalidiert das genutzte Access-Token

Tabelle 6: Schnittstellen für die OAuth 2.0 Kommunikation

Parameter	Beschreibung
<i>response_type</i>	Für die Authentifizierung mittels <i>Implicit-Grant</i> der Wert „token“. Für Authentifizierung mittels <i>Authorization-Code-Grant</i> der Wert „code“. Für Authentifizierung mittels <i>Resource-Owner-Password-Credentials-Grant</i> der Wert „password“.
<i>client_id</i>	Der <i>api_key</i> des Klienten
<i>scope</i>	In den ersten Versionen des Systems immer „student“. Später möglicherweise auch andere Werte.
<i>state</i>	Ein im Web-Browser gespeicherter, nicht von dritten veränderbarer Schlüssel, der vom REST-Webservice in der Antwort mitgesendet wird.

Tabelle 7: Übergebene Parameter bei der Weiterleitung an /auth/login

Authentifizierungs-Vorgang Es wird hier lediglich die Authentifizierung mittels *Implicit-Grant* beschrieben, da andere Authentifizierungs-Typen nicht zur Implementierung vorgesehen sind.

Zunächst leitet der Klient den Web-Browser des Nutzers beim Login auf den URI /auth/login des REST-Webservices weiter. Hierbei werden im *Query* die Parameter aus Tabelle 7 übergeben.

Wenn die *client_id* einem Klienten des REST-Webservices zuordenbar ist, wird der Web-Browser des Nutzers nach der Authentifizierung an den *redirect_url* weitergeleitet. Ist die *client_id* nicht zuordenbar, so wird ein Status-Code 420 Policy Not Fulfilled mit einer Fehlermeldung zurückgegeben. Die Authentifizierung schlägt in folgenden Fällen mit den Parametern aus Tabelle 8 im *Fragment* fehl (siehe hierfür auch [1, Kap. 4.2.2.1]):

invalid_request	Fehlen von in Tabelle 7 spezifizierten Parametern
unsupported_response_type	Nicht unterstützter <i>response_type</i> (alles außer „token“)
invalid_scope	Nicht unterstützter <i>scope</i> (momentan ist der einzige unterstützte Scope „student“)
server_error	Fehler des Servers

Ist die Authentifizierung erfolgreich, so werden bei der Weiterleitung die Parameter aus Tabelle 9 im *Fragment* übergeben.

Ressourcen-Zugriff Beim Zugriff auf die Ressourcen des REST-Webservices übergibt der Klient das *access_token* über den Header-Wert „Authorization: Bearer <*access_token*>“. Der Zugriff wird gestattet, wenn das *access_token* gültig und nicht abgelaufen ist, und der

Parameter	Beschreibung
<i>error</i>	Ein den Fehler identifizierenden Schlüssel (siehe hierfür auch [1, Kap. 4.2.2.1])
<i>state</i>	Das bei der Anfrage vom Klienten übergebene <i>state</i> -Parameter

Tabelle 8: Übergebene Parameter bei einer fehlgeschlagenen Authentifizierung

Parameter	Beschreibung
<i>access_token</i>	Ein Token, mit welchem der Klient im Namen des Nutzers auf den REST-Webservice zugreifen kann
<i>token_type</i>	„Bearer“ (siehe [1, Kap. 7.1])
<i>expires_in</i>	Dauer, welche das <i>access_token</i> gültig ist.
<i>scope</i>	In den ersten Versionen des Systems immer „student“. Später möglicherweise auch andere Werte.
<i>state</i>	Das bei der Anfrage vom Klienten übergebene <i>state</i> -Parameter

Tabelle 9: Übergebene Parameter bei einer erfolgreichen Authentifizierung

Referer-Header-Wert dem regulären Ausdruck *origin* des Klienten entspricht. Ist der Klient nicht authentifiziert bzw. das *access_token* ungültig, so wird mit 401 Unauthorized geantwortet.

4.3. Atomare Werte

Es folgt die Beschreibung der für die REST-Kommunikation notwendigen atomaren Werte.

Bezeichner	Erklärung
<i>{Modul-Turnus}</i>	$\in \{ "WS", "SS", "both" \}$.
<i>{Semester-Typ}</i>	$\in \{ "WS", "SS" \}$.
<i>{Jahr}</i>	Jahreszahl (Ganzzahl).
<i>{Studienfach-ID}</i>	ID, welche ein eindeutiges Studienfach repräsentiert.
<i>{Studienfach-Name}</i>	Name eines Studienfachs.
<i>{Modul-ID}</i>	ID, welche ein eindeutiges Modul repräsentiert.
<i>{Modul-Name}</i>	Name eines Moduls.
<i>{Kategorie-ID}</i>	ID, welche eine eindeutige Modul-Kategorie repräsentiert.
<i>{Kategorie-Name}</i>	Name einer Modul-Kategorie (String).

Bezeichner	Erklärung
$\langle \text{Modul-Semester} \rangle$	Das Semester, in welchem sich ein Modul befindet (Ganzzahl).
$\langle \text{Modul-Creditpoints} \rangle$	ECTS-Zahl eines Moduls.
$\langle \text{Modul-Dozent} \rangle$	Name des Dozenten des Moduls.
$\langle \text{Modul-Präferenz} \rangle$	$\in \{\text{"positive"}, \text{"negative"}, \text{""}\}$.
$\langle \text{Modul-Beschreibung} \rangle$	Beschreibungstext eines Moduls.
$\langle \text{Constraint-Name} \rangle$	Bezeichner eines Constraints.
$\langle \text{Constraint-Typ} \rangle$	Kurzer Text, der den Constraint-Typ beschreibt (für UI).
$\langle \text{Filter-ID} \rangle$	ID, welche einen eindeutigen Filter repräsentiert.
$\langle \text{Filter-Name} \rangle$	Name des Filters (Titel für UI).
$\langle \text{Filter-Default} \rangle$	Für List-Filter: Nummer des standardmäßig ausgewählten Elements. Für Contains-Filter: Standardmäßiger Suchstring. Für Range-Filter:
	<pre>{ "min": <Filter-Minimum>, "max": <Filter-Maximum> }</pre>
$\langle \text{Filter-Tooltip} \rangle$	Kurzer Beschreibungstext des Filters für ein UI-Tooltip.
$\langle \text{Filter-Typ} \rangle$	$\in \{\text{"range"}, \text{"list"}, \text{"contains"}\}$.
$\langle \text{Filter-Minimum} \rangle$	Ganze Zahl, die eine untere Schranke eines Range-Filters beschreibt.
$\langle \text{Filter-Maximum} \rangle$	Ganze Zahl, die eine obere Schranke eines Range-Filters beschreibt.
$\langle \text{Item-ID} \rangle$	Nummer des $\langle \text{Filter-Wahlitem} \rangle$ eines UI-Filter-Auswahlfeldes (die Wahlitems sind nullbasiert fortlaufend nummeriert).
$\langle \text{Item-Text} \rangle$	Anzeigetext der Wahlmöglichkeit eines UI-Filter-Auswahlfeldes (String).
$\langle \text{Zielfunktion-ID} \rangle$	ID, welche eine eindeutige Zielfunktion repräsentiert.
$\langle \text{Zielfunktion-Name} \rangle$	Name einer Zielfunktion.
$\langle \text{Zielfunktion-Beschreibung} \rangle$	Beschreibungstext einer Zielfunktion für die UI.
$\langle \text{Studienplan-ID} \rangle, \langle \text{Plan-ID} \rangle$	ID, welche einen eindeutigen Studienplan repräsentiert.

Bezeichner	Erklärung
$\langle\text{Studienplan-Status}\rangle$	$\epsilon \{"valid", "invalid", "not-verified"\}.$
$\langle\text{Studienplan-Name}\rangle$	Name eines Studienplans.
$\langle\text{Studienplan-Gesamt-Creditpoints}\rangle$	Gesamtzahl aller ECTS eines Studienplans.
$\langle\text{Untere-ECTS-Schranke}\rangle/\langle\text{Obere-ECTS-Schranke}\rangle$	$\langle\text{Filter-Minimum}\rangle/\langle\text{Filter-Maximum}\rangle$ für ECTS-Filter.
$\langle\text{Semester-Minimum}\rangle/\langle\text{Semester-Maximum}\rangle$	Minimale/Maximale Anzahl an Semestern, die der Generierungsvorschlag haben soll.
$\langle\text{Semester-ECTS-Minimum}\rangle/\langle\text{Semester-ECTS-Maximum}\rangle$	Minimale/Maximale Anzahl von ECTS pro Semester, die der Generierungsvorschlag haben soll.
$\langle\text{Access-Token}\rangle$	Das in Kapitel 4.2, Tabelle 9 spezifizierte Access-Token.

4.4. Spezifikation der JSON-Datenklassen

Im Folgenden werden die zur REST-Kommunikation nötigen JSON-Datenklassen mithilfe der in Kapitel 4.1 beschriebenen Notation spezifiziert.

Modellklassen

```
«Student» = {
    "discipline": «Studienfach [id]»,
    "study-start": «Studienbeginn»,
    "passed-modules": [«Modul [id, semester]», ...]
}
```

```
«Studienbeginn» = {
    "semester-type": {Semester-Typ},
    "year": {Jahr}
}
```

```
«Studienfach» = {
```

```

    "id": <Studienfach-ID>,
    "name": <Studienfach-Name>
}

```
Modul`` = {
 "id": <Modul-ID>,
 "name": <Modul-Name>,
 "categories": [<Kategorie>, ...],
 "semester": <Modul-Semester>,
 "cycle-type": <Modul-Turnus>,
 "creditpoints": <Modul-Creditpoints>,
 "lecturer": <Modul-Dozent>,
 "preference": <Modul-Präferenz>,
 "description": <Modul-Beschreibung>,
 "constraints": [<Constraint>, ...]
}

```
Constraint`` = {
    "name": <Constraint-Name>,
    "first": ``Modul [id]``,
    "second": ``Modul [id]``,
    "type": <Constraint-Typ>,
}

```
Kategorie`` = {
 "id": <Kategorie-ID>,
 "name": <Kategorie-Name>
}

```
Filter`` = {
    "id": <Filter-ID>,
    "name": <Filter-Name>,
    "default-value": <Filter-Default>,
    "tooltip": <Filter-Tooltip>,
    "specification": ``Filter-Eigenschaften``
}

```
Filter-Eigenschaften`` = {
 "type": <Filter-Typ>,
 "case type == "range": {
 "min": <Filter-Minimum>,
 "max": <Filter-Maximum>
 },
}

```

```

case type == "list": {
 "items": [«Filter-Wahlitem», ...]
},
case type == "contains": {}
}

```

Anmerkung: Die case-Notation beschreibt eine Fallunterscheidung; die in den einzelnen Fällen in geschweifte Klammern gekapselten Attribute liegen hierbei in derselben Ebene wie die Fallunterscheidungen. Es versteht sich von selbst, dass stets genau einer der Fälle zutrifft.

```

«Filter-Wahlitem» = {
 "id": {Item-ID},
 "text": {Item-Text}
}

```

```

«Zielfunktion» = {
 "id": {Zielfunktion-ID},
 "name": {Zielfunktion-Name},
 "description": {Zielfunktion-Beschreibung}
}

```

```

«Studienplan» = {
 "id": {Studienplan-ID},
 "status": {Studienplan-Status},
 "creditpoints-sum": {Studienplan-Gesamt-Creditpoints},
 "name": {Studienplan-Name},
 "modules": [«Modul [id, name, semester, creditpoints, lecturer]», ...],
 "violations": [«Constraint», ...]
}

```

## Kommunikations-Datenstrukturen

```

«ModulesResult» = {
 "modules": [«Modul [id, name, creditpoints, lecturer]», ...]
}

```

```

«ModuleResult» = {
 "module": «Modul [id, name, categories, creditpoints, lecturer, description,
 constraints]»
}

```

```

```
<<StudentResult>> = {
    "student": <<Student>>
}
```
```
<<StudentPutRequest>> = {
    "student": <<Student>>
}
```
```
<<StudentDeleteRequest>> = {
    "student": <<Student [id]>>
}
```
```
<<PlansGetResult>> = {
    "plans": [<<Studienplan [id, status, creditpoints-sum, name]>>, ...]
}
```
```
<<PlansPostRequest>> = {
    "plan": <<Studienplan [name]>>
}
```
```
<<PlansPostResult>> = {
    "plan": <<Studienplan [id, name]>>
}
```
```
<<PlanResult>> = {
    "plan": <<Studienplan>>
}
```
```
<<PlanPutRequest>> = {
    "plan": <<Studienplan>>
}
```
```
<<PlanPatchPostRequest>> = {
    "plan": <<Studienplan [name]>>
}
```
```
<<PlanPatchPostResult>> = {
    "plan": <<Studienplan [id, name]>>
}
```

```

```

```
<<PlanModulesResult>> = {
    "modules": [ <<Modul [id, name, creditpoints, lecturer, preference]>>, ...]
}

<<PlanModuleResult>> = {
    "module": <<Modul>>
}

<<PlanModulePutRequest>> = {
    "module": <<Modul [id, semester]>>
}

<<PlanModulePutResult>> = {
    "module": <<Modul [id, semester]>>
}

<<ModulePreferencePutRequest>> = {
    "module": <<Modul [id, preference]>>
}

<<ModulePreferencePutResult>> = {
    "module": <<Modul [id, preference]>>
}

<<PlanVerificationResult>> = {
    "plan": <<Studienplan [id, status, violations]>>
}

<<PlanProposalResult>> = {
    "plan": <<Studienplan [status, modules, violations]>>
}

<<FiltersResult>> = {
    "filters": [ <<Filter>>, ...]
}

<<ObjectiveFunctionsResult>> = {
    "functions": [ <<Zielfunktion>>, ...]
}
```

```

```
«DisciplinesResult» = {
 "disciplines": [«Studienfach», ...]
}
```

```
«SubjectsResult» = {
 "subjects": [«Kategorie», ...]
}
```

## 4.5. Zugriffsstruktur

Im Folgenden wird die Zugriffsstruktur der REST-Schnittstelle anhand HTTP-Methoden und URIs beschrieben. GET-Parameter-Definitionen (\*) sind weiter unten separat aufgeführt.

| Methode | URI                 | Beschreibung                                                          | Anfrage-Parameter /Rückgabewerte (Statuscodes)                   |
|---------|---------------------|-----------------------------------------------------------------------|------------------------------------------------------------------|
| GET     | /modules            | Lese planunabhängige Modul-Liste (gefiltert) (Name, ECTS und Dozent)  | Anfrage: <i>Modules-Parameter</i> *<br>Rückgabe: «ModulesResult» |
| GET     | /modules/{Modul-ID} | Lese Modul (planunabhängig)                                           | Anfrage: —<br>Rückgabe: «ModuleResult»                           |
| GET     | /student            | Lese Informationen über Student                                       | Anfrage: —<br>Rückgabe: «StudentResult»                          |
| PUT     | /student            | Ersetze Informationen über Student, lösche Verifikationsinformationen | Anfrage: «StudentPutRequest»<br>Rückgabe: «StudentResult»        |
| DELETE  | /student            | Löscht Student                                                        | Anfrage: «StudentDeleteRequest»<br>Rückgabe: —                   |
| GET     | /plans              | Lese Planliste                                                        | Anfrage: —<br>Rückgabe: «PlansGetResult»                         |
| POST    | /plans              | Erstellt neuen Studienplan                                            | Anfrage: «PlansPostRequest»<br>Rückgabe: «PlansPostResult»       |

| Methode | URI                                            | Beschreibung                                                                                 | Anfrage-Parameter /Rückgabewerte (Statuscodes)                                                         |
|---------|------------------------------------------------|----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| GET     | /plans/{Plan-ID}                               | Lese Plan                                                                                    | Anfrage: —<br>Rückgabe: « <i>PlanResult</i> »                                                          |
| PUT     | /plans/{Plan-ID}                               | Ersetze Plan                                                                                 | Anfrage: « <i>PlanPutRequest</i> »<br>Rückgabe: « <i>PlanResult</i> »                                  |
| PATCH   | /plans/{Plan-ID}                               | Bearbeite Plan:<br>Name ändern                                                               | Anfrage:<br>« <i>PlanPatchPostRequest</i> »<br>Rückgabe:<br>« <i>PlanPatchPostResult</i> »             |
| DELETE  | /plans/{Plan-ID}                               | Lösche Plan                                                                                  | —                                                                                                      |
| POST    | /plans/{Plan-ID}                               | Dupliziere Plan                                                                              | Anfrage:<br>« <i>PlanPatchPostRequest</i> »<br>Rückgabe:<br>« <i>PlanPatchPostResult</i> »             |
| GET     | /plans/{Plan-ID}/modules                       | Lese Modul-Liste<br>(gefiltert) (Name,<br>ECTS, Dozent und<br>Präferenz)                     | Anfrage:<br><i>PlanModules-Parameter</i> *<br>Rückgabe:<br>« <i>PlanModulesResult</i> »                |
| GET     | /plans/{Plan-ID}/modules/{Modul-ID}            | Lese Modul mit<br>{Modul-ID}                                                                 | Anfrage: —<br>Rückgabe:<br>« <i>PlanModuleResult</i> »                                                 |
| PUT     | /plans/{Plan-ID}/modules/{Modul-ID}            | Setze Modul in<br>Plan in gegebenes<br>Semester, setze Veri-<br>fizierung zurück             | Anfrage:<br>« <i>PlanModulePutRequest</i> »<br>Rückgabe:<br>« <i>PlanModulePutResult</i> »             |
| DELETE  | /plans/{Plan-ID}/modules/{Modul-ID}            | Lösche Modul aus<br>Plan, setze Verifi-<br>zierung zurück                                    | Anfrage: —<br>Rückgabe: —                                                                              |
| PUT     | /plans/{Plan-ID}/modules/{Modul-ID}/preference | Setze Bewertung<br>für Modul                                                                 | Anfrage:<br>« <i>ModulePreferencePutRequest</i> »<br>Rückgabe:<br>« <i>ModulePreferencePutResult</i> » |
| GET     | /plans/{Plan-ID}/verification                  | Verifiziere den<br>Plan, gebe das<br>Ergebnis zurück<br>und speichere es<br>in der Datenbank | Anfrage: —<br>Rückgabe:<br>« <i>PlanVerificationResult</i> »                                           |

| Methode | URI                                         | Beschreibung                                                            | Anfrage-Parameter /Rückgabewerte (Statuscodes)                                     |
|---------|---------------------------------------------|-------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| GET     | /plans/{Plan-ID}/proposal/{Zielfunktion-ID} | Erstelle und erhalte einen auf Basis des Plans generierten (neuen) Plan | Anfrage: <i>Proposal-Parameter</i> *<br>Rückgabe:<br>« <i>PlanProposalResult</i> » |
| GET     | /plans/{Plan-ID}/pdf                        | Lese PDF-Version des Plans                                              | Anfrage: <i>PDF-Parameter</i> *<br>Rückgabe: (Weiterleitung auf PDF)               |
| GET     | /filters                                    | Lade Filtertypen und Beschränkungen                                     | Anfrage: —<br>Rückgabe: « <i>FiltersResult</i> »                                   |
| GET     | /objective-functions                        | Lese Liste mit allen vorhandenen Zielfunktionen                         | Anfrage: —<br>Rückgabe:<br>« <i>ObjectiveFunctionsResult</i> »                     |
| GET     | /disciplines                                | Lese Liste mit allen angebotenen Studienfächern                         | Anfrage: —<br>Rückgabe: « <i>DisciplinesResult</i> »                               |
| GET     | /subjects                                   | Lese Liste mit allen angebotenen Vertiefungsfächern                     | Anfrage: —<br>Rückgabe: « <i>SubjectsResult</i> »                                  |

## GET-Anfrageparameter

Die Parameter der GET-Anfragen werden im Folgenden tabellarisch beschrieben. Parameter mit *kursivem Namen* sind optional und werden nur falls nötig angegeben.

### GET /modules (Modules-Parameter)

$filter_k \in \{ects, category, type, discipline, compulsory, cycletype, name\}$ , bildet die Liste der aktivierten Filter;  
Sofern  $filters$  leer oder nicht vorhanden ist, werden die Module ohne Filterung zurückgegeben.

| Parametername | Wert                                     |
|---------------|------------------------------------------|
| $filters$     | $filter_1, \dots, filter_n$              |
| $ects-min$    | $\langle$ Untere-ECTS-Schranke $\rangle$ |

| Parametername     | Wert                                                     |
|-------------------|----------------------------------------------------------|
| <i>ects-max</i>   | $\langle\text{Obere-ECTS-Schranke}\rangle$               |
| <i>category</i>   |                                                          |
| <i>type</i>       |                                                          |
| <i>discipline</i> | $\langle\text{Item-ID}\rangle$ des ausgewählten Elements |
| <i>compulsory</i> |                                                          |
| <i>cycletype</i>  |                                                          |
| <i>name</i>       | (Suchstring)                                             |

#### GET /plans/{Plan-ID}/modules (PlanModules-Parameter)

| Parametername | Wert                       |
|---------------|----------------------------|
|               | (siehe Modules-Parameter*) |

#### GET /plans/{Plan-ID}/proposal/{Zielfunktion-ID} (Proposal-Parameter)

| Parametername            | Wert                                         |
|--------------------------|----------------------------------------------|
| <i>min-semesters</i>     | $\langle\text{Semester-Minimum}\rangle$      |
| <i>max-semesters</i>     | $\langle\text{Semester-Maximum}\rangle$      |
| <i>min-semester-ects</i> | $\langle\text{Semester-ECTS-Minimum}\rangle$ |
| <i>max-semester-ects</i> | $\langle\text{Semester-ECTS-Maximum}\rangle$ |
| <i>preferred-subject</i> | (Nummer des ausgewählten Vertiefungsfaches)  |

#### GET /plans/{Plan-ID}/pdf (PDF-Parameter)

| Parametername       | Wert                                |
|---------------------|-------------------------------------|
| <i>access-token</i> | $\langle\text{Access-Token}\rangle$ |

## 4.6. Spezifikation der HTTP-Statuscodes

Abschließend folgt die Spezifikation der für die REST-Kommunikation verwendeten HTTP-Statuscodes. Außer bei den Statuscodes 200 und 201 werden – abgesehen vom Header – keine weiteren Daten als Antwort gesendet.

| Statuscode | Beschreibung                                                                          |
|------------|---------------------------------------------------------------------------------------|
| 200 OK     | Die Anfrage konnte korrekt entgegengenommen, ausgeführt und ggfs. beantwortet werden. |

| Statuscode                       | Beschreibung                                                                                                                                                                                                                                                                                                                           |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>201</b> Created               | Die Anfrage konnte korrekt entgegengenommen und ausgeführt werden. Eine oder mehrere Ressourcen wurden erfolgreich auf dem Server neu angelegt.                                                                                                                                                                                        |
| <b>400</b> Bad Request           | Die Anfrage-Parameter sind fehlerhaft oder unvollständig. Die Anfrage wurde daher nicht ausgeführt.                                                                                                                                                                                                                                    |
| <b>401</b> Unauthorized          | Für die Anfrage ist erst eine Authentifizierung erforderlich (siehe Kap. 4.2).                                                                                                                                                                                                                                                         |
| <b>404</b> Not Found             | Eine oder mehrere angefragte Ressourcen sind mit vorliegenden Rechten nicht zugänglich, da sie nicht existieren oder der Zugriff verboten ist.                                                                                                                                                                                         |
| <b>405</b> Method Not Allowed    | Die Zugriff auf die URI mittels dieser HTTP-Methode ist nicht zulässig.                                                                                                                                                                                                                                                                |
| <b>422</b> Unprocessable Entity  | Die Anfrage-Parameter sind syntaktisch korrekt, allerdings ist die Anfrage aufgrund semantischer Konflikte nicht ausführbar. (Kurz: Die Anfrage wird vom Server verstanden, ergibt aber keinen Sinn.) Beispiele: Entfernen einer nicht vorhandenen Ressource, Einfügen einer bereits vorhandenen Ressource, Name bereits vergeben etc. |
| <b>500</b> Internal Server Error | Es wurde versucht, die Anfrage auszuführen. Dabei ist serverseitig ein interner Fehler aufgetreten.                                                                                                                                                                                                                                    |

## 5. Verwendete Entwurfsmuster

- Builder
  - server.pluginmanager.GenerationManager (baut die Zielfunktion zusammen)
- Composite
  - server.generation.objectivefunction (gesamtes Paket)
  - server.generation.standard (Node, NodeWithOutput, NodeWithoutOutput)
  - server.filter (Filter und Multifilter)
- Façade
  - server.rest
- Factory
  - server.model.moduledata.dao.ModuelDaoFactory
  - server.model.SessionFactory
  - server.model.userdata.dao.PlanDaoFactory
  - server.model.userdata.dao.UserDaoFactory
- Null-Object
  - server.filter.TrueFilter

- Observer
  - client.view (überall wo onChange() vorhanden ist)
  - client.model.system.EventBus
  - client.router
- Singleton
  - client.model.system.TemplateManager
  - client.model.system.LanguageManager
  - client.model.system.EventBus
  - client.model.user.SessionInformation
  - studyplan.server.model.HibernateUtil (Methoden zur Bereitstellung der SessionFactories)
- StateMachine
  - client.view.components.uipanel.WizardComponent
- Strategy
  - server.pluginmanager (jeweils GenerationManager und Verification Manager)
  - server.rest.authorization.endpoint
  - client.model.system.OAuthCollection
  - server.model.moduledata.constraint.ModuleConstraintType#isValid()
  - server.model: Die Schnittstellen UserDao, PlanDao und ModuleDao abstrahieren den Datenbankzugriff. Die HibernateDaos stellen die konkrete Zugriffsstrategie dar.
- Template Method
  - server.filter.Filter (getCondition())
  - server.filter.AttributeFilter (getDescriptor() und getFilterType())
- Visitor
  - server.model.moduledata.dao.ModuleDao#getModulesByFilter(): Der übergebene Filter stellt den Besucher dar, der zum Filtern verwendet wird

## 6. Paketbeschreibungen

In diesem Kapitel wird die Paketstruktur des Projekts beschrieben. Manche Pakete werden der Übersicht wegen ausgelassen.

### 6.1. Package `client.model`

Paket, dessen Klassen die Informationen des Systems kapseln.

Beinahe alle Klassen erweitern hierbei die von Backbone zur Verfügung gestellte Basisklasse Backbone.Model und die zugehörigen Collections die Basisklasse Backbone.Collection.

Die Klassen überschreiben hierbei die Methode `parse()` um die vom Server erhaltenen Daten in die Struktur der Model-Klasse zu überführen. Aus Gründen der Übersichtlichkeit wurde diese Funktion in der Dokumentation der Subpakete weggelassen.

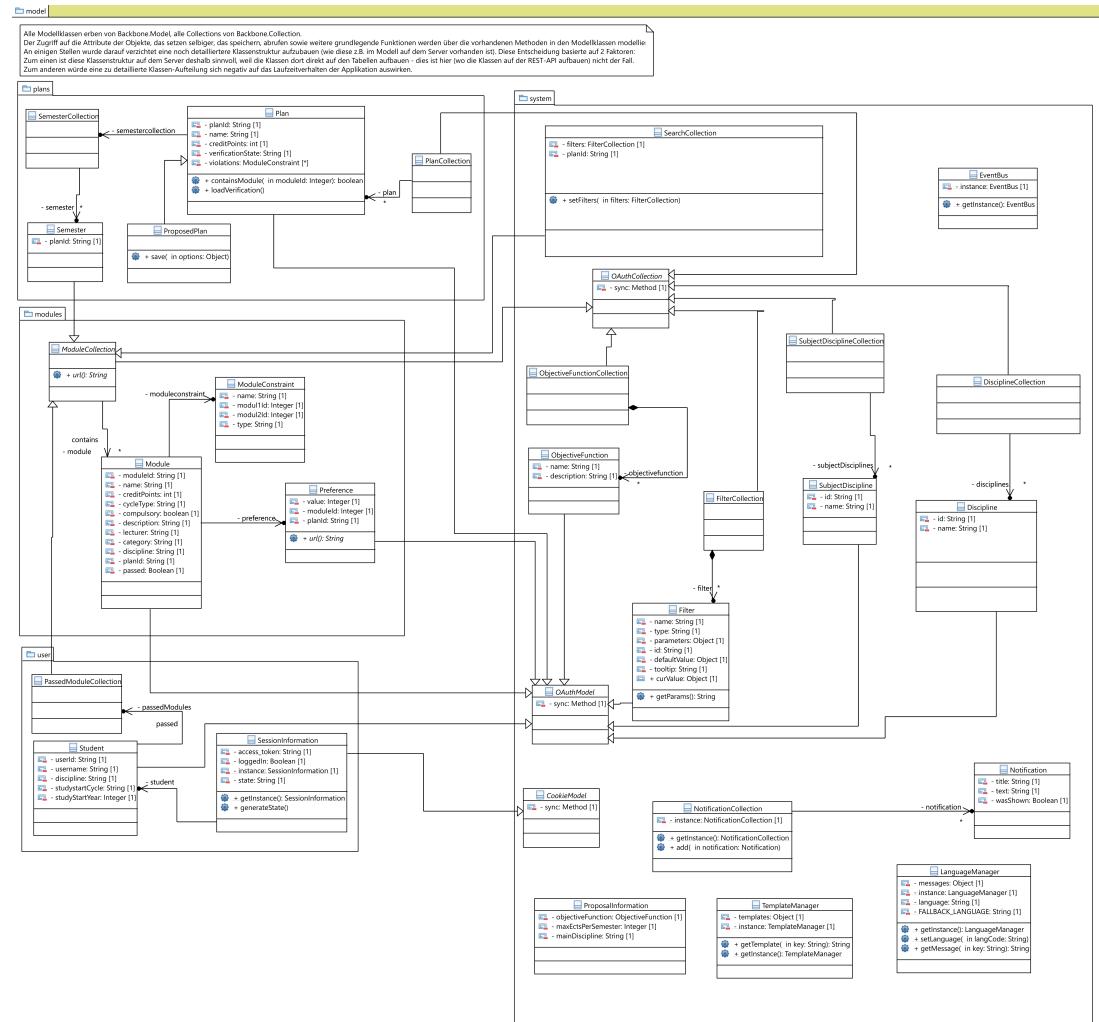


Abbildung 5: UML-Diagramm des client.model-Pakets.

## 6.2. Package `client.model.modules`

Paket, dessen Klassen die Informationen eines Moduls modellieren.  
Alle Klassen dieses Pakets erben, falls nicht anders angegeben, von `OAuthModel`, da es sich um REST-Resourcen handelt.

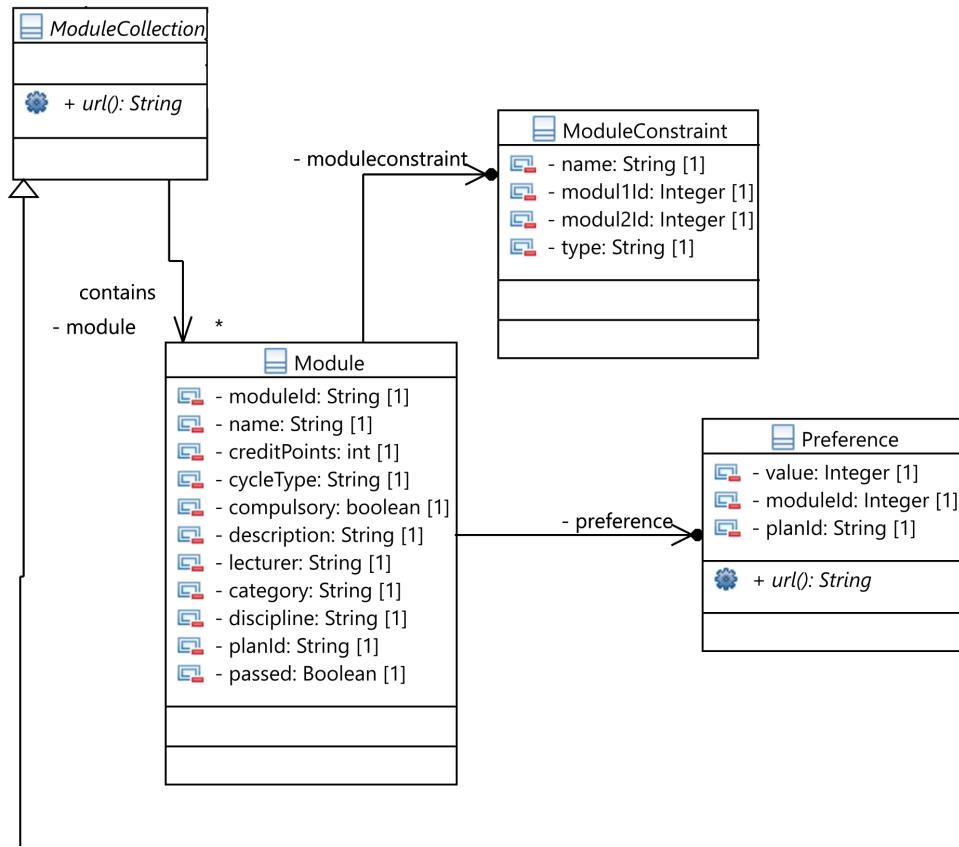


Abbildung 6: UML-Diagramm des `client.model.modules`-Pakets.

### 6.3. Package `client.model.plans`

Paket, dessen Klassen die Informationen eines Studienplans modellieren.  
Alle Klassen erben, falls nicht anders angegeben, von `OAuthModel`, da es sich um REST-Resourcen handelt.

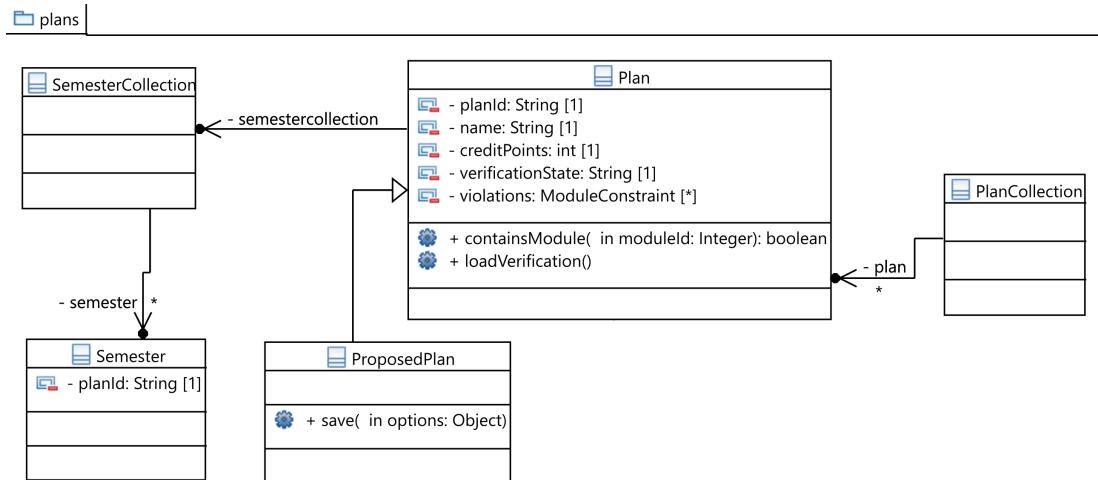


Abbildung 7: UML-Diagramm des client.model.plans-Pakets.

#### 6.4. Package `client.model.system`

Paket, dessen Klassen Systemkomponenten modellieren.

Unter anderem enthält dieses Paket auch die Klassen OAuthModel, OAuthCollection und CookieModel, welche die Speicherschnittstelle der jeweiligen Unterklassen festlegen.

system

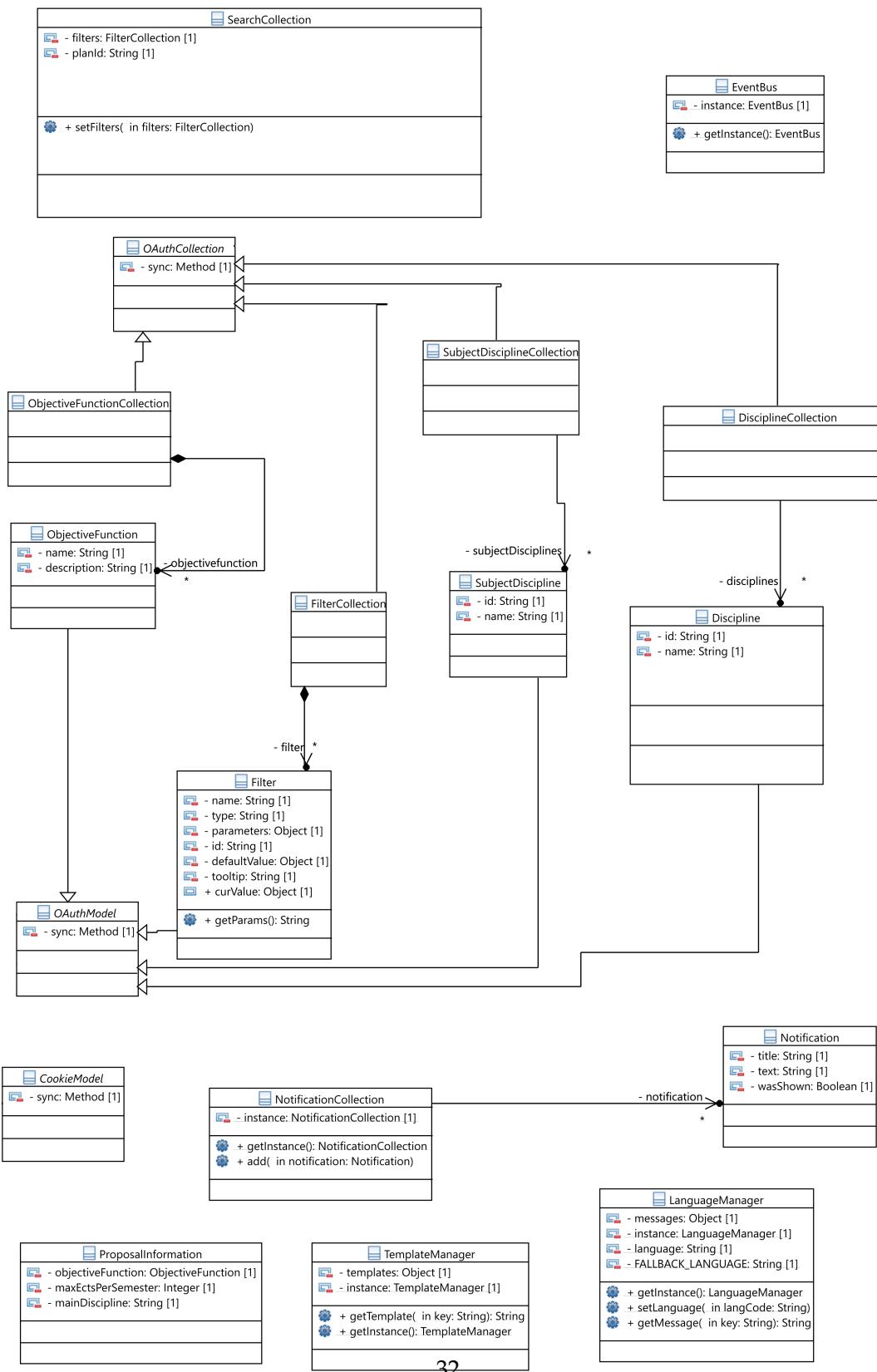


Abbildung 8: UML-Diagramm des `client.model.system`-Pakets.

## 6.5. Package `client.model.user`

Paket, dessen Klassen die Informationen eines Nutzers modellieren.  
Alle Klassen erben, falls nicht anders angegeben, von OAuthModel, da es sich um REST-Resourcen handelt.

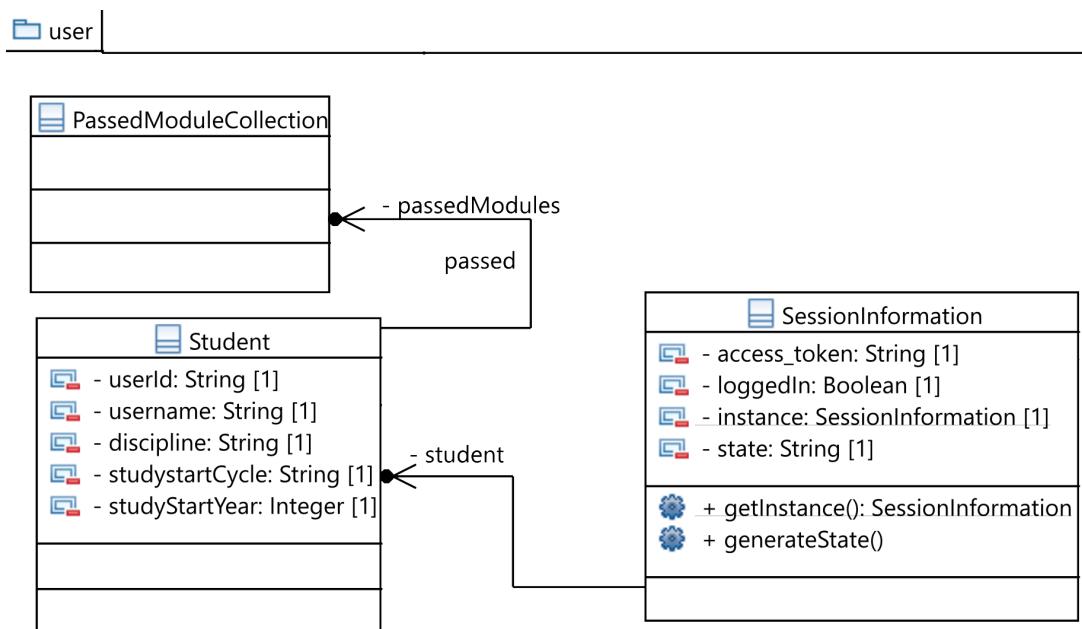


Abbildung 9: UML-Diagramm des `client.model.user`-Pakets.

## 6.6. Package `client.router`

Der Router übernimmt die Funktion des Controllers in klassischen Model-View-Controller-Architekturen. Backbone.History ruft bei einer Veränderung des URL-Fragments die jeweils passende Methode des MainRouters auf, welche dann die entsprechenden Views initialisiert.

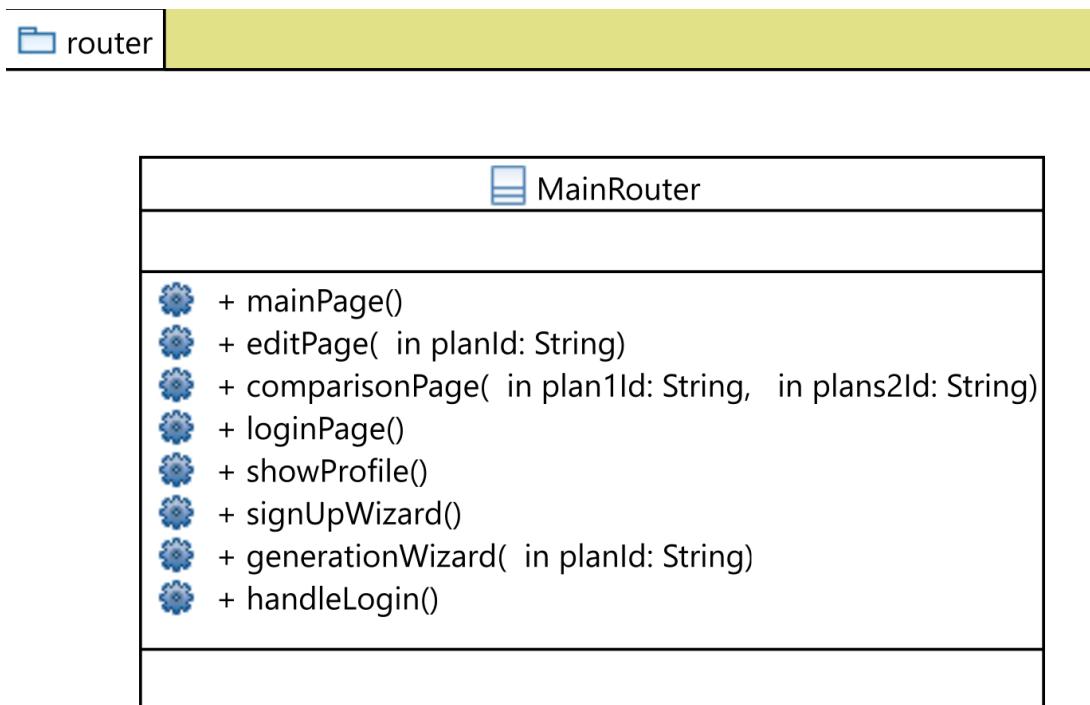


Abbildung 10: UML-Diagramm des client.router-Pakets.

## 6.7. Package `client.storage`

Dieses Paket kapselt die Synchronisierung von Model-Daten mit dem REST-Service bzw. den gespeicherten Cookies. Die Methoden sind dabei nach dem Konzept von Backbone.Sync aufgebaut und können deshalb von den Backbone.Model Klassen verwendet werden.

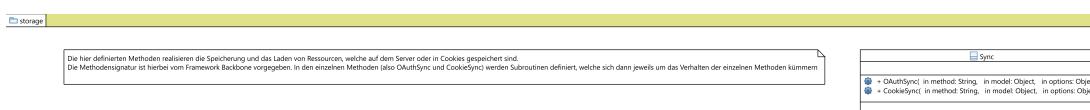


Abbildung 11: UML-Diagramm des client.storage-Pakets.

## **6.8. Package `client.view.components.filter`**

Dieses Paket enthält die Filterelemente, welche bei der Modulsuche angezeigt werden.

filter

---

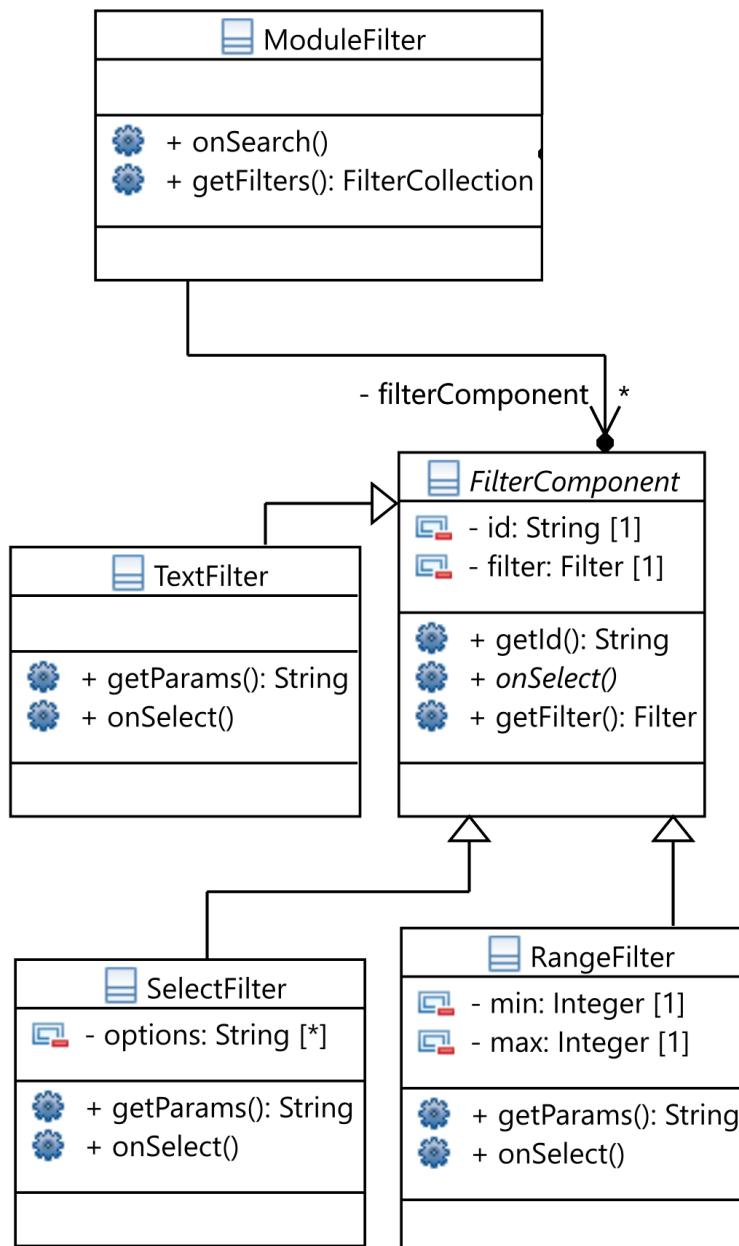


Abbildung 12: UML-Diagramm des `client.view.components.filter`-Pakets.

## 6.9. Package client.view.components.uielement

Dieses Paket enthält einzelne Elemente der grafischen Benutzeroberfläche, welche oftmals auch an mehreren Stellen zur Verwendung kommen.

Die einzelnen Elemente werden dann wiederum im Rahmen von uipanel angezeigt, welche wiederum von subviews angezeigt werden.

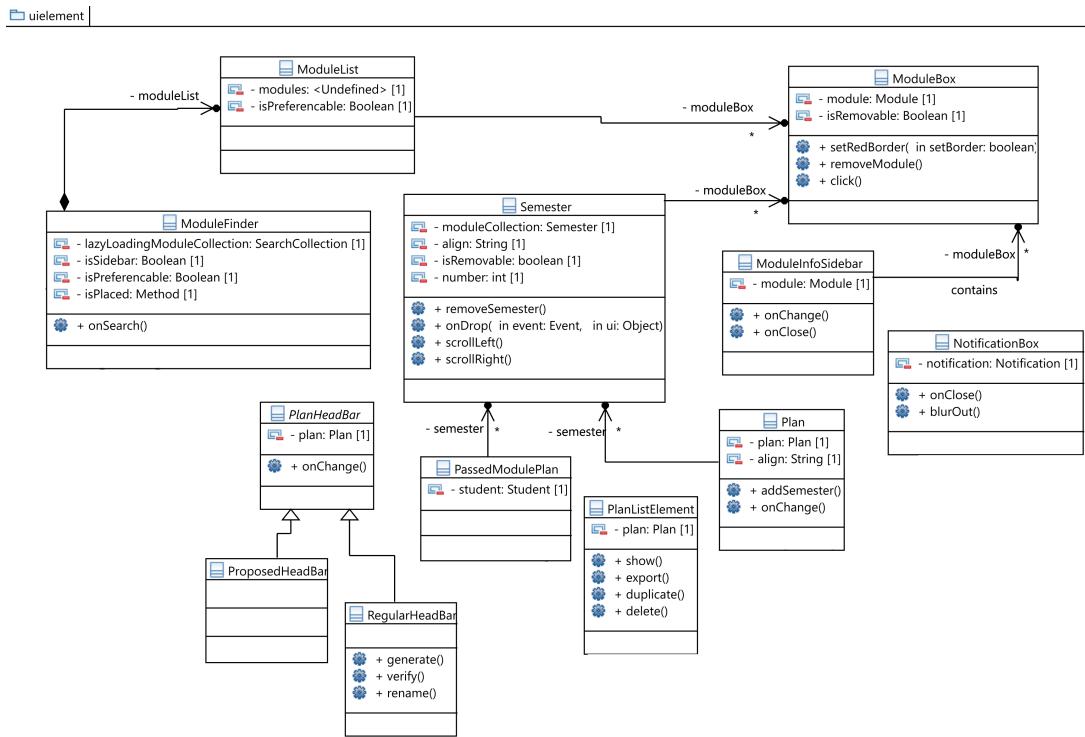


Abbildung 13: UML-Diagramm des client.view.components.uielement-Pakets.

## 6.10. Package client.view.components.uipanel

Dieses Paket enthält größere Strukturen der grafischen Benutzeroberfläche, welche sich aus einzelnen Elementen des Pakets client.view.components.uielement zusammensetzen.

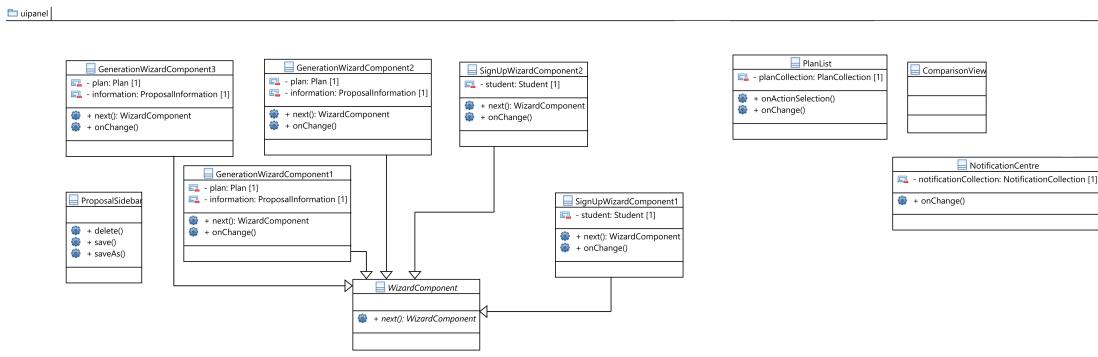


Abbildung 14: UML-Diagramm des client.view.components.uipanel-Pakets.

## 6.11. Package `client.view`

Dieses Paket enthält die in Klassen gekapselte grafische Oberfläche des Systems. Es handelt sich bei allen Klassen um Erweiterungen der Basisklasse Backbone.View. Hierbei implementiert jede der Klassen die Funktion render(), welche beim Aufruf die eigenen Inhalte als grafische Oberfläche zusammenbaut. Diese wird an dem dem Konstruktor übergebenen Ort auf der Seite anzeigt. Die render()-Funktion - welche von Backbone vorgegeben wird - ruft insbesondere auch die render()-Funktionen von im Objekt enthaltenen Views auf.

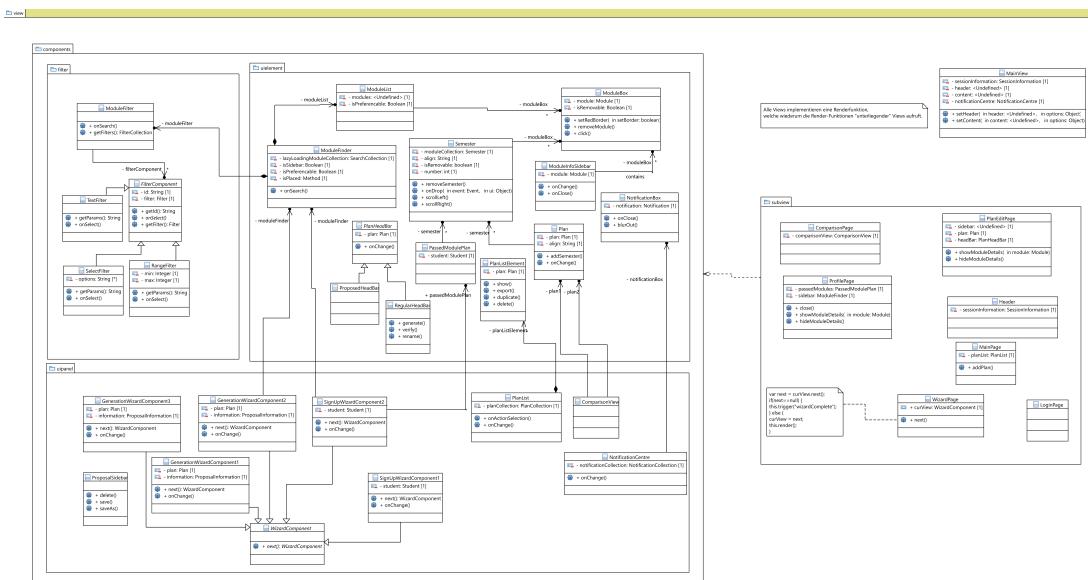


Abbildung 15: UML-Diagramm des client.view-Pakets.

## 6.12. Package client.view.subview

Dieses Paket fasst uipanels zu vollwertigen Seiten zusammen, welche dann von den Klassen dieses Pakets auf Abruf von client.router angezeigt werden.

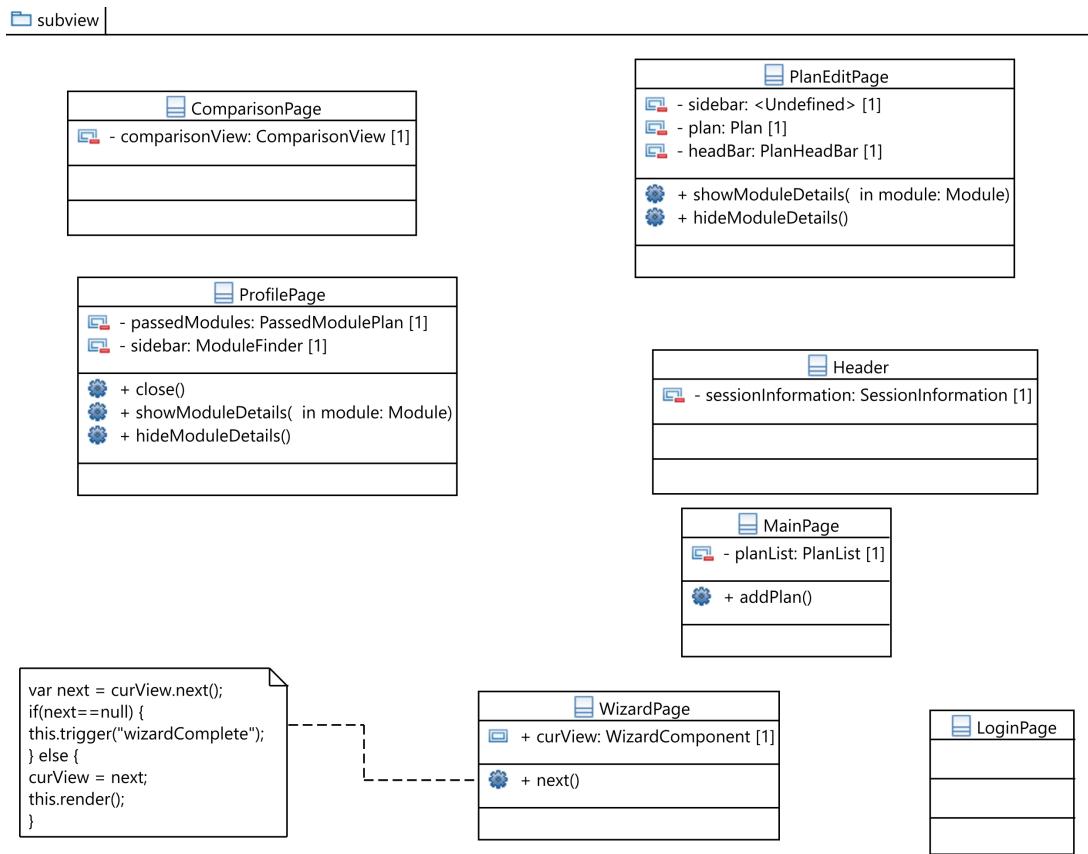


Abbildung 16: UML-Diagramm des client.view.subview-Pakets.

### 6.13. Package server.filter

Das Filter-Paket enthält das Modulfilter-System und bietet Möglichkeiten, nach außen sichtbare Filtertypen aufzuzählen, neue Filter zu erstellen sowie diese anschließend zur Filterung von Modulen zu nutzen.

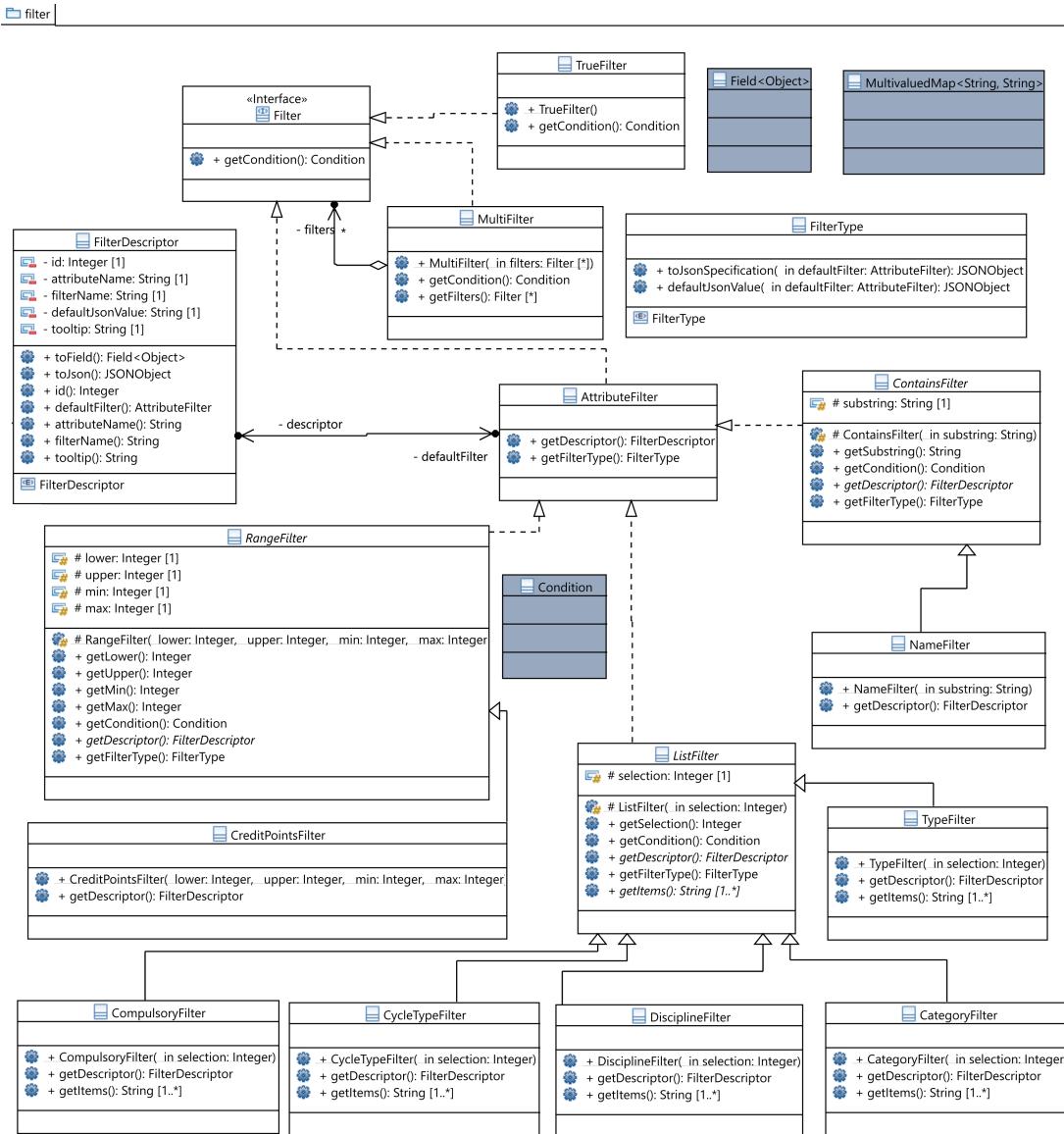


Abbildung 17: UML-Diagramm des Filter-Pakets.

## 6.14. Package server.generation

Das Paket server.generation beinhaltet die Implementierung der Generierung. Die Generierung umfasst das Entgegennehmen von angefangenen Studienplänen, das den

System-Constraints angepasste Vervollständigen des Studienplans und das den Nutzer-Constraints entsprechende Optimieren dieses Plans anhand einer Zielfunktion.

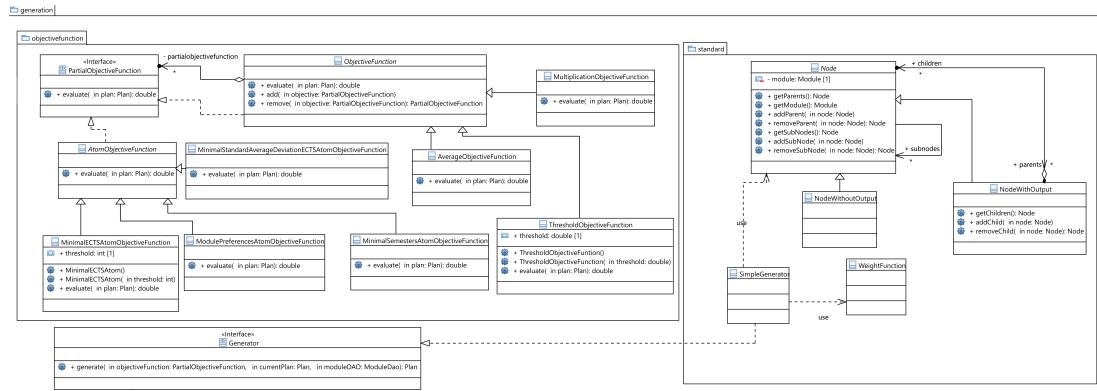


Abbildung 18: UML-Diagramm des Generation-Pakets

## 6.15. Package server.generation.objectivefunction

Das Paket server.generation.objectivefunction beinhaltet die Implementierung der Zielfunktionen für die Generierung. Eine Zielfunktion bewertet einen Studienplan anhand von Kriterien wie ECTS pro Semester, Semesteranzahl, Nutzerpräferenzen, etc.

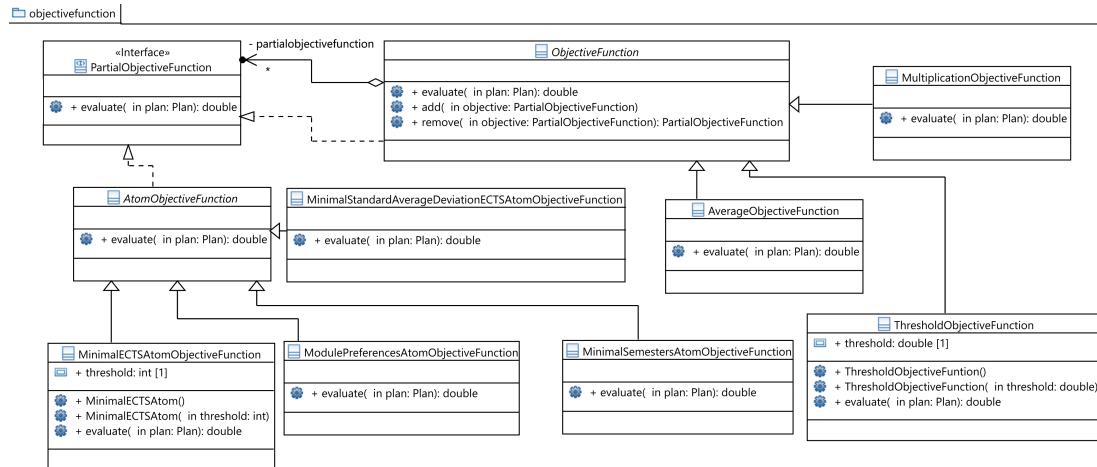


Abbildung 19: UML-Diagramm des Objectivefunction-Pakets.

## 6.16. Package server.generation.standard

Das Paket server.generation.standard beinhaltet eine konkrete Generation-Implementierung. Hierzu wird neben der generator-implementierenden Klasse SimpleGenerator eine Graphstruktur verwendet. Das Kompositum besteht aus Knoten vom Typ "Node", die entweder Ausgangskanten zu anderen Knoten "Node" haben (NodeWithOutput) oder keine Ausgangskanten haben(NodeWithoutOutput). Zudem nutzt der SimpleGenerator die Klasse WeightFunction.

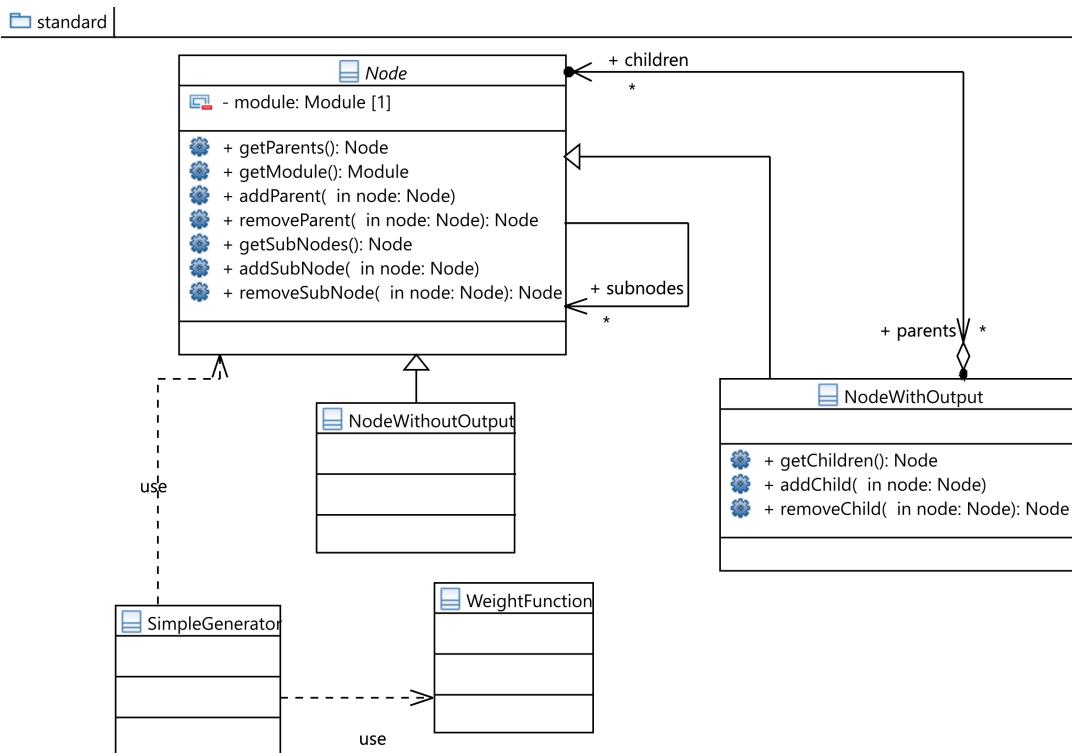


Abbildung 20: UML-Diagramm des Objectivefunction-Standard-Pakets.

## 6.17. Package server.model

Dieses Paket enthält Modellklassen für alle in der Datenbank gespeicherten Daten und überträgt die Datenbankstruktur auf ein objektorientiertes Schema. Des Weiteren stellt es Schnittstellen zum Datenbankzugriff in Form von sogenannten *Data Access Objects* (DAOs) bereit, welche einen leichten Austausch der Datenbankzugriffsstruktur ermöglichen.

### 6.17.1. Package server.model.moduledata

Dieses Paket enthält Modellklassen der Moduldaten.

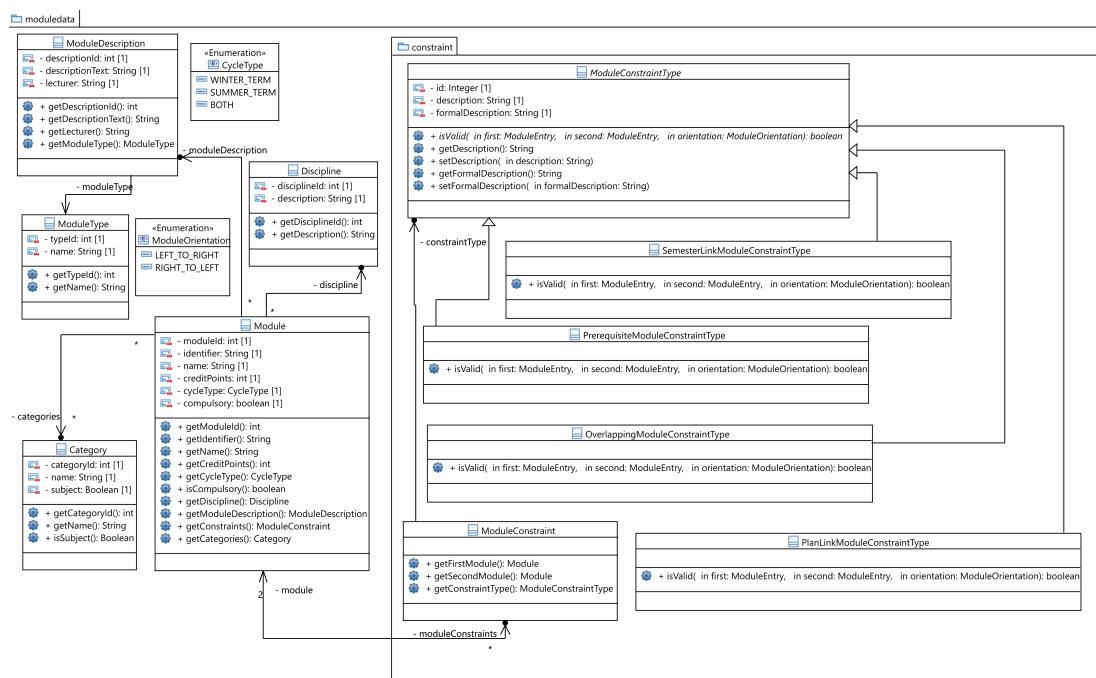


Abbildung 21: UML-Diagramm des Moduldaten-Pakets.

**Package server.model.moduledata.constraint** Das Paket enthält Klassen zur Modellierung von Modulabhängigkeiten und verschiedenen Abhängigkeitstypen.

**Package server.model.moduledata.dao** Enthält Schnittstellen-Beschreibungen zum Zugriff auf die Moduldatenbank sowie eine konkrete DAO-Implementierung mithilfe von *Hibernate*.

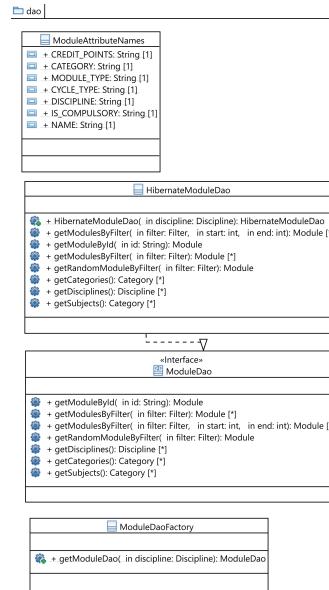


Abbildung 22: UML-Diagramm des Pakets `model.moduledata.dao`.

### 6.17.2. Package `server.model.usermodel`

Das Paket enthält Modellklassen für die Datensätze der Nutzer-Datenbank und somit für Nutzer, Studienpläne und Authentifizierungsinformationen.

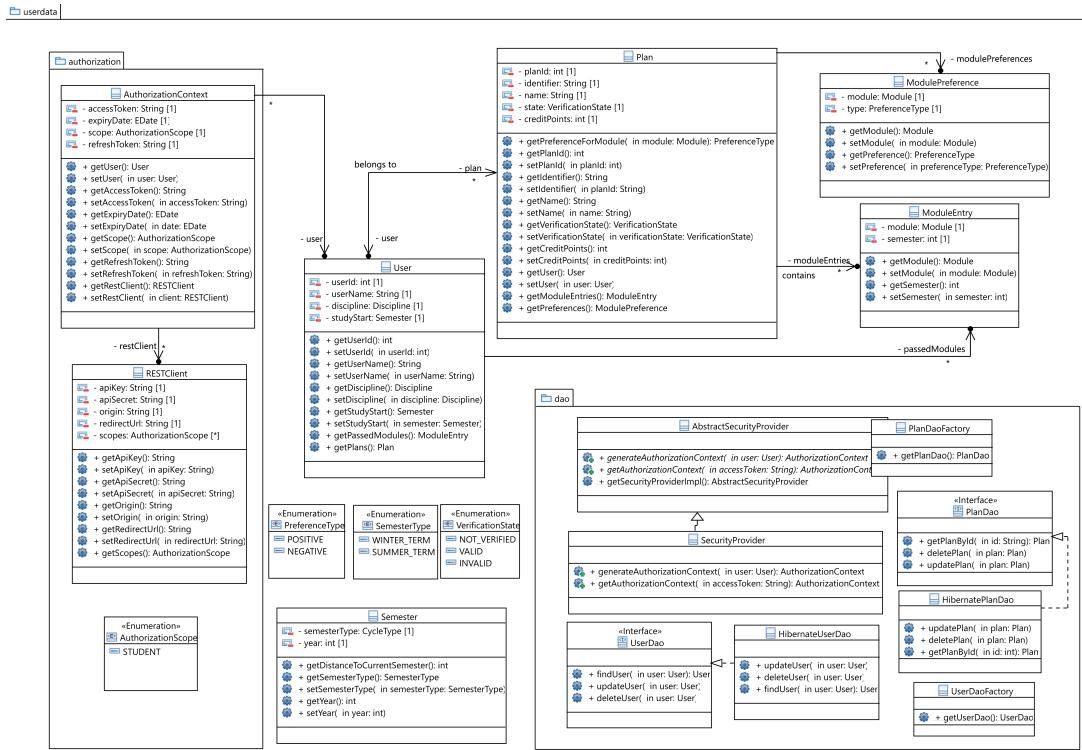


Abbildung 23: UML-Diagramm des Nutzerdaten-Pakets.

**Package `server.model.userdata.authorization`** Dieses Paket enthält die benötigten Klassen zur Modellierung von Authentifizierungsdaten wie in Kapitel 4.2 beschrieben.

**Package `server.model.userdata.dao`** Enthält Schnittstellen-Beschreibungen zum Lesen und Schreiben der Nutzer-Datenbank sowie konkrete, *Hibernate*-basierte DAO-Implementierungen zum Zugriff auf Nutzer, Studienpläne und Authentifizierungsinformationen.

## 6.18. Package `server.pluginmanager`

Dieses Paket verwaltet den Zugriff auf die Generierung- und Verifizierungs-Plug-ins. Es ist das einzige Paket, das direkt auf die Klassen der Pakete 6.14 und 6.21 zugreift.

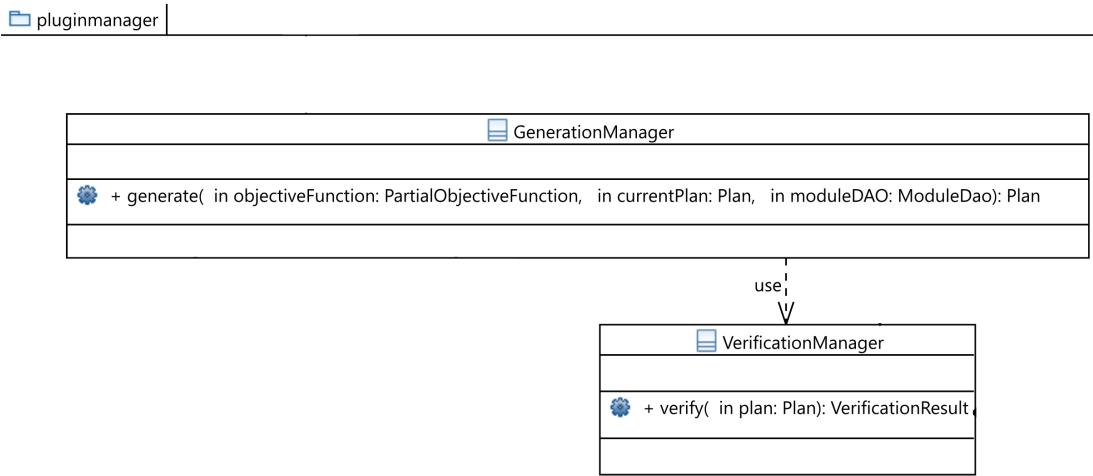


Abbildung 24: UML-Diagramm des `server.rest.pluginmanager`-Pakets.

## 6.19. Package `server.rest.authorization.endpoint`

Dieses Paket repräsentiert den Authorization Endpoint auf dem Authentifizierungs-Server, auf dem sich der Endbenutzer anmelden und dem Client die Authentifizierung erteilen kann.

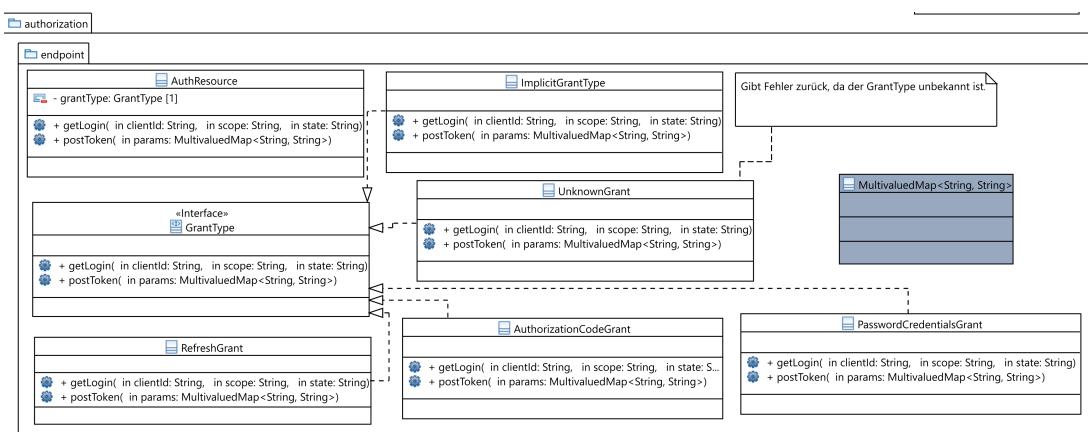


Abbildung 25: UML-Diagramm des `server.rest.authorization.endpoint`-Pakets.

## 6.20. Package server.rest

Dieses Paket enthält Klassen, die die Ressourcen des REST-Webservices repräsentieren. Es wird als Kommunikationsschnittstelle zwischen dem Client und dem Server verwendet.

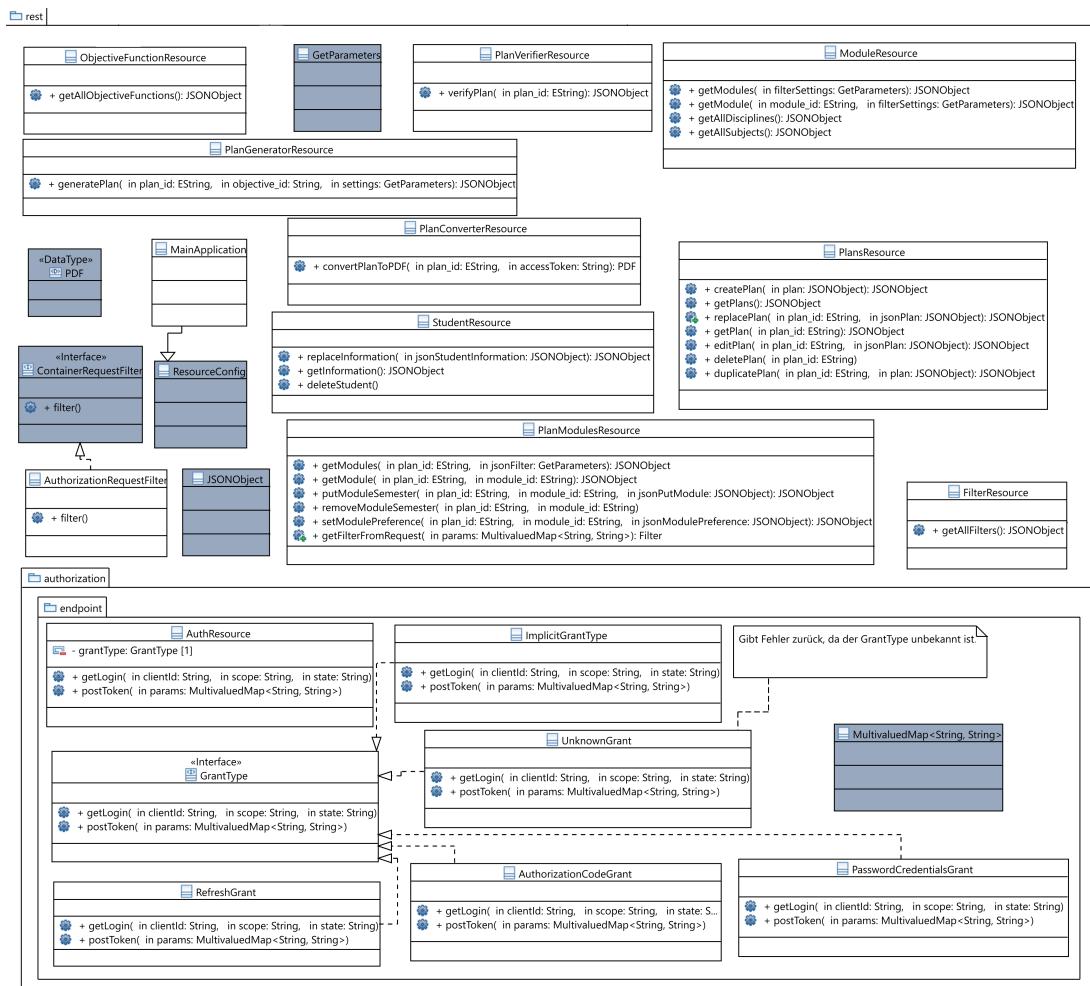


Abbildung 26: UML-Diagramm des `server.rest`-Pakets.

## 6.21. Package server.verification

Das Paket server.verification beinhaltet die Implementierung der Verifizierung. Die Verifizierung prüft, ob ein Studienplan alle System-Constraints erfüllt und somit einen erfolgreichen Studienabschluss ermöglicht.

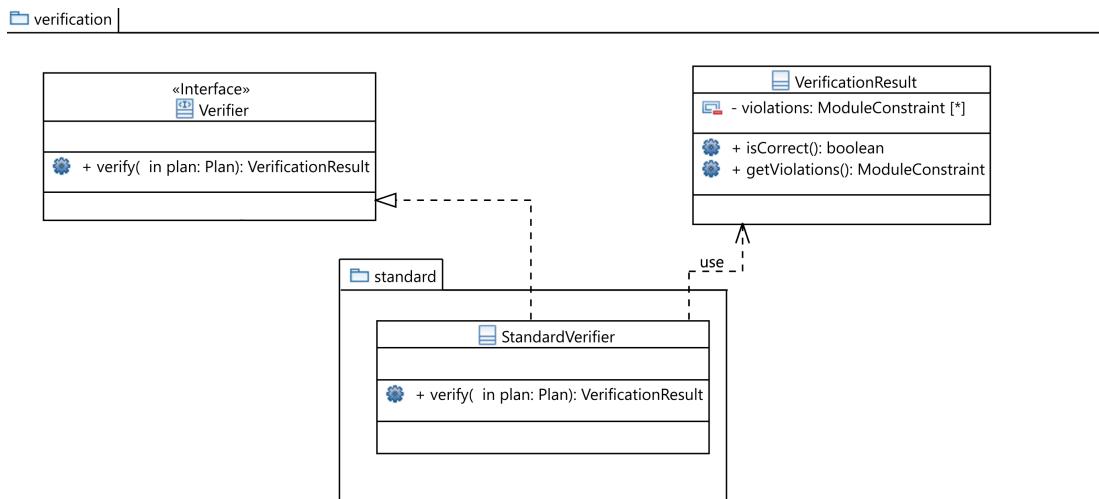


Abbildung 27: UML-Diagramm des Verification-Pakets.

## 6.22. Package server.verification.standard

Das Paket `server.verification.standard` beinhaltet eine konkrete Verification-Implementierung.

# 7. Klassenbeschreibungen

## 7.1. Übersicht

In diesem Kapitel folgt die Dokumentation aller Java-/Javascript-Klassen des Projektes. Bei Klassenmembern, deren Bedeutung sich über Vererbungs-/Implementierungs-Hierarchien hinweg nicht maßgeblich ändert, wurde auf redundante Dokumentation verzichtet. Als grobes Inhaltsverzeichnis diene folgende Package-Liste (alle Package-Namen relativ zu `edu.kit.informatik`).

|                                     |    |                                     |     |
|-------------------------------------|----|-------------------------------------|-----|
| client.model.modules                | 54 | server.generation.standard          | 100 |
| client.model.plans                  | 56 | server.model                        | 102 |
| client.model.system                 | 58 | server.model.moduledata             | 103 |
| client.model.user                   | 67 | server.model.moduledata.constraint  | 106 |
| client.router                       | 68 | server.model.moduledata.dao         | 109 |
| client.storage                      | 69 | server.model.userdata.authorization | 112 |
| client.view.components.filter       | 70 | server.model.userdata.dao           | 114 |
| client.view.components.uielement    | 72 | server.model.userdata               | 117 |
| client.view.components.uipanel      | 78 | server.pluginmanager                | 122 |
| client.view                         | 83 | server.rest.authorization.endpoint  | 124 |
| client.view.subview                 | 84 | server.rest                         | 127 |
| server.filter                       | 87 | server.verification                 | 134 |
| server.generation                   | 95 | server.verification.standard        | 134 |
| server.generation.objectivefunction | 96 |                                     |     |

## **Package client.model.modules**

### **class Module**

**Beschreibung** Diese Klasse repräsentiert ein Modul, welches vom Server geladen wird. Das Modul wird als Element zunächst nur teilweise geladen. Mittels eines Aufrufs von fetch() werden dann gegebenenfalls weitere notwendige Daten nachgeladen.

#### **Attribute**

- **private String category**  
Die Kategorie des Moduls
- **private boolean compulsory**  
Ob das Modul verpflichtend ist
- **private int creditPoints**  
Die Anzahl an Credit-Points, die das Modul erbringt
- **private String cycleType**  
Ob das Modul im Wintersemester (= "WS"), Sommersemester ("SS") oder immer ("both") stattfindet
- **private String description**  
Die Beschreibung des Moduls.
- **private String discipline**  
Das Fach des Moduls
- **private String lecturer**  
Der Dozent des Moduls
- **private ModuleConstraint moduleconstraint**  
Die zum Modul gehörenden System-Constraints
- **private String moduleId**  
Die ID des Moduls
- **private String name**  
Der Name des Moduls
- **private boolean passed**  
Ob das Modul bereits vom Nutzer bestanden wurde. Dies wird gespeichert, da die Trennung zwischen Modulen im Studienplan und bestandenen Modulen für die Anzeige der Module auf der Client-Seite aufgehoben wird. Dies ist problemlos möglich, da diese Trennung lediglich für Berechnungen interessant sind, welche nicht auf der Client-Seite stattfinden.
- **private String planId**  
Die ID des Plans zu der das Modul gehört (da die Präferenzen planspezifisch sind, sind die Module immer an einen Plan gebunden. Die einzige Ausnahme hiervon sind die Module bei der Auswahl der bereits bestandenen Leistungen. In diesem Fall ist postId = null.)
- **private Preference preference**  
Die vom Nutzer für das Modul gesetzte Präferenz

## Konstruktoren

- public **Module**(Object params)

## class ModuleCollection

**Beschreibung** Eine Sammlung von Modulen, welche vom Server abgerufen werden kann

## Attribute

- private Collection<Module> **modules**

## Konstruktoren

- public **ModuleCollection**(Object params)

## Methoden

- public abstract String **url()**

Generiert abhängig von der PlanId den URL für den Abruf der Module

**Rückgabe** Der generierte URL

## class ModuleConstraint

**Beschreibung** Diese Klasse beschreibt einen zu einem Modul gehörenden System-Constraint (erweitert Backbone.Model)

## Attribute

- private int **modul1Id**  
Die ID des ersten Moduls
- private int **modul2Id**  
Die ID des zweiten Moduls
- private String **name**  
Der Typ des ModuleConstraint. Dieser wird als String angegeben.
- private String **type**  
Textuelle Beschreibung des ModuleConstraints für den Nutzer

## Konstruktoren

- public **ModuleConstraint**(Object params)

## **class Preference**

**Beschreibung** Diese Klasse beschreibt eine zu einem Modul gehörende gesetzte Präferenz (positiv/neutral/negativ)

### **Attribute**

- **private int moduleId**  
Die ID des Moduls, zu welcher die Präferenz gehört
- **private String planId**  
Die ID des Plans für welche die Präferenz gesetzt wurde.
- **private int value**  
Beschreibt die gesetzte Präferenz (einer der drei Werte {"negativ", "positiv", ""})

### **Konstruktoren**

- **public Preference (Object params)**

### **Methoden**

- **public String url ()**  
Generiert die URL für das Speichern/Abrufen der Präferenz abhängig von planId und moduleId  
**Rückgabe** Den generierten URL

## **Package client.model**

### **Package client.model.plans**

## **class Plan**

**Beschreibung** Klasse, welche einen Studienplan beschreibt

### **Attribute**

- **private int creditPoints**  
Die Credit-Points des Plans
- **private String name**  
Der Name des Plans
- **private int planId**  
Die ID des Plans
- **private SemesterCollection semestercollection**  
Die Collection, welche die im Studienplan enthaltenen Semester beinhaltet
- **private String verificationState**  
Der Verifikationsstatus des Plans

- private Collection<ModuleConstraint> **violations**  
Die bei der Verifizierung festgestellten Fehler im Plan

### Konstruktoren

- public **Plan**(Object params)

### Methoden

- public boolean **containsModule**(int moduleId)  
Eine Methode, welche zurückgibt, ob das Modul mit der gegebenen ID im Plan vorhanden ist.

### Parameter

**moduleId** Die ID des Moduls, nach welchem gesucht werden soll

**Rückgabe** True, wenn das Modul vorhanden ist, False wenn nicht.

- public void **loadVerification()**  
Lädt die Informationen zur Verifizierung und speichert diese im Plan.

## class PlanCollection

**Beschreibung** Klasse, welche eine Collection von Studienplänen beschreibt

### Attribute

- private Collection<Plan> **plan**  
Die Collection der Studienpläne

### Konstruktoren

- public **PlanCollection**(Object params)

## class ProposedPlan

**Beschreibung** Klasse, welche einen vorgeschlagenen Plan repräsentiert

### Konstruktoren

- public **ProposedPlan**(Object params)

### Methoden

- public void **save**(Object options)  
Methode, welche den Plan ggf. speichert

#### **Parameter**

- options** Alle Optionen von Backbone.Model.save(); zusätzlich:
- Boolean newPlan: true, falls der Plan als neuer Plan geschrieben werden soll; false, falls der Plan überschrieben werden soll. Standard ist false
  - String planName: Name des Plans, falls newPlan = true

### **class Semester**

**Beschreibung** Klasse, welcher ein Semester als Collection von Modulen beschreibt.

#### **Attribute**

- **private String planId**  
Die ID des Plans, zu welchem das Semester gehört.

#### **Konuktoren**

- **public Semester(Object params)**

#### **Methoden**

- **public String url()**

### **class SemesterCollection**

**Beschreibung** Eine Collection von Semestern, welche von einem Plan genutzt wird.  
(Erbt von Backbone.Collection)

#### **Attribute**

- **private Collection<Semester> semester**  
Die in der Collection enthaltenen Semester

#### **Konuktoren**

- **public SemesterCollection(Object params)**

### **Package client.model.system**

### **class CookieModel**

**Beschreibung** Ein Model, welches in einem Cookie gespeichert wird (Erweitert von Backbone.Model)

### **Attribute**

- `private Object sync`  
Die Synchronisierungs-Methode, welche das Model-Objekt in einem Cookie speichert

### **Konstruktoren**

- `public CookieModel (Object params)`

## **class Discipline**

**Beschreibung** Klasse, welche ein Fach repräsentiert

### **Attribute**

- `private String id`  
ID des Fachs
- `private String name`  
Name des Fachs

### **Konstruktoren**

- `public Discipline (Object params)`

## **class DisciplineCollection**

**Beschreibung** Klasse, welche eine Collection von Fächern repräsentiert

### **Attribute**

- `private Collection<Discipline> disciplines`  
Die Collection der Fächer

### **Konstruktoren**

- `public DisciplineCollection (Object params)`

## **class EventBus**

**Beschreibung** Dieses Singleton wird als Event-Bus zwischen verschiedenen Komponenten der Applikation verwendet. Jedes Objekt kann mittels `getInstance()` auf den EventBus zugreifen und sich als Listener registrieren oder ein Event auslösen.

Ein Beispiel für ein solches Event wäre die Anzeige eines Moduls in der Seitenleiste beim Klick auf das Modul. (Erbt von Backbone.Event)

### **Attribute**

- `private static EventBus instance`  
Die Instanz des Event-Busses

### **Konstruktoren**

- `private EventBus()`

### **Methoden**

- `public static EventBus getInstance()`  
Methode zum Erhalten einer Instanz des Event Busses  
**Rückgabe** Die Event-Bus Instanz

## **class Filter**

**Beschreibung** Klasse, welche die Daten zu einem Filter speichert

### **Attribute**

- `private Object curValue`  
Der aktuelle Wert des Filters
- `private Object defaultValue`  
Der Standardwert des Filters (Struktur abhängig vom Filter)
- `private String id`  
Die ID des Filters
- `private String name`  
Name des Filters
- `private Object parameters`  
Die den Filter spezifizierenden Eigenschaften
- `private String tooltip`  
Ein Tooltip, der den Filter textuell beschreibt
- `private String type`  
Typ des Filters ("range", "list", "contains")

### **Konstruktoren**

- `public Filter(Object params)`

### **Methoden**

- `public String getParams()`  
Konvertiert die Filter-Daten in GET-Query-Parameter  
**Rückgabe** Die GET-Query-Parameter

## **class FilterCollection**

**Beschreibung** Eine Collection von Filtern

### **Attribute**

- `private Collection<Filter> filter`  
Die Filter der Collection

### **Konstruktoren**

- `public FilterCollection(Object params)`

## **class LanguageManager**

**Beschreibung** Singleton, welches die sprachlichen Ausgaben des Programms verwaltet. Hierbei lässt sich die Sprache, in welcher die Nachrichten ausgegeben werden, mittels `setLanguage` statisch einstellen. Die Nachrichten werden bereits zur Compile-Zeit in den Initialisierungscode des Programms geladen und sind deshalb von Beginn an verfügbar.

### **Attribute**

- `private static String FALLBACK_LANGUAGE`  
Die Ausweich-Sprache des LanguageManagers
- `private static LanguageManager instance`  
Die Instanz des LanguageManagers
- `private static String language`  
Die eingestellte Sprache des LanguageManagers (als Sprachcode)
- `private Object messages`  
Ein Objekt, welches die Strings, die das Programm ausgibt, enthält

### **Konstruktoren**

- `private LanguageManager()`

### **Methoden**

- `public static LanguageManager getInstance()`  
Gibt die aktuelle Instanz des LanguageManagers zurück
- Rückgabe** Die aktuelle Instanz des Sprachmanagers
- `public String getMessage(String key)`  
Methode zum Erhalten einer Nachricht in der aktuellen Sprache auf Basis eines Keys

#### **Parameter**

**key** Der Key, welcher den auszugebenden String identifiziert

**Rückgabe** Den String, falls dieser vorhanden ist, sonst den Key selbst

- public static void **setLanguage**(String langCode)  
Setzt die Sprache des LanguageManagers fest

#### Parameter

**langCode** Der Code der Sprache, welche gesetzt werden soll

## **class Notification**

**Beschreibung** Klasse, welche eine Benachrichtigung repräsentiert (erweitert Backbone.Model)

#### Attribute

- private String **text**  
Der Text der Benachrichtigung
- private String **title**  
Titel der Benachrichtigung
- private boolean **wasShown**  
Ob die Benachrichtigung bereits angezeigt wurde

#### Konstruktoren

- public **Notification**(Object params)

## **class NotificationCollection**

**Beschreibung** Singleton-Collection, welche alle Benachrichtigungen enthält  
(erweitert Backbone.Collection)

#### Attribute

- private static NotificationCollection **instance**  
Die aktuelle Instanz der NotificationCollection
- private Collection<Notification> **notification**  
Die Collection aller Notifications

#### Konstruktoren

- private **NotificationCollection()**

#### Methoden

- public void **add**(Notification notification)  
Methode zum Hinzufügen einer neuen Benachrichtigung

### **Parameter**

- **notification** Die Benachrichtigung, welche hinzugefügt werden soll
- public static NotificationCollection **getInstance()**  
Methode zum Erhalten der aktuellen Instanz der NotificationCollection
- **Rückgabe** Die aktuelle Instanz von NotificationCollection

## **class OAuthCollection**

**Beschreibung** Diese Klasse repräsentiert eine Collection, welche als Ressource in der REST-Schnittstelle gespeichert ist, und bei welcher sich man beim Zugriff über OAuth authentifizieren muss. (Erweitert Backbone.Collection)

### **Attribute**

- private Object **sync**  
Die Synchronisierungs-Methode, welche das Model-Objekt in einem Cookie speichert

### **Konuktoren**

- public **OAuthCollection** (Object params)

## **class OAuthModel**

**Beschreibung** Diese Klasse repräsentiert ein Model, welches als Ressource in der REST-Schnittstelle gespeichert ist, und bei welchem man sich beim Zugriff über OAuth authentifizieren muss. (Erweitert Backbone.Model)

### **Attribute**

- private Object **sync**  
Die Synchronisierungs-Methode, welche das Model-Objekt in einem Cookie speichert

### **Konuktoren**

- public **OAuthModel** (Object params)

## **class ObjectiveFunction**

**Beschreibung** Klasse, welche eine Zielfunktion repräsentiert

### **Attribute**

- private String **description**  
Beschreibung der ObjectiveFunction
- private String **name**  
Name der ObjectiveFunction

### **Konstruktoren**

- public **ObjectiveFunction**(Object params)

## **class ObjectiveFunctionCollection**

**Beschreibung** Klasse, welche eine Collection von Zielfunktionen repräsentiert

### **Attribute**

- private Collection<ObjectiveFunction> **objectivefunction**  
Die Collection aller Zielfunktionen

### **Konstruktoren**

- public **ObjectiveFunctionCollection**(Object params)

## **class ProposalInformation**

**Beschreibung** Die Methoden, welche im Zusammenhang mit der Generierung eines Studienplans abgespeichert und zum Server geschickt werden. (Erweitert Backbone.Model)

### **Attribute**

- private String **mainDiscipline**  
Das gewählte Vertiefungsfach
- private int **maxEctsPerSemester**  
Die maximale Anzahl an ECTS-Punkten pro Semester bei der Generierung
- private ObjectiveFunction **objectiveFunction**  
Die ObjectiveFunction, welche zum Generieren genutzt werden soll

### **Konstruktoren**

- public **ProposalInformation**(Object params)

## **class SearchCollection**

**Beschreibung** Die Search-Collection, welche das Lazy-Loading der Module in der Suche übernimmt. Hierbei werden in Zehnerschritten nach und nach mehr Module geladen und in einzelne SearchModuleCollections gespeichert. Beim Abruf mittels getModules() werden diese dann in eine einzige Collection zusammengeführt.

### **Attribute**

- private FilterCollection **filters**  
Die Filter, welche beim Abruf zu berücksichtigen sind
- private int **planId**  
Die ID des Plans, zu welcher die Suche gehört; oder null, falls die Suche planunabhängig ist

### **Konstruktoren**

- public **SearchCollection**(Object params)

### **Methoden**

- public void **setFilters**(FilterCollection filters)  
Methode, welche die Filter neu setzt und im Zuge dessen die alten SearchModuleCollections löscht und highestLoaded zurücksetzt. Anschließend werden erneut Module geladen.

#### **Parameter**

**filters** Die Filter, welche bei der Suche berücksichtigt werden sollen.

- public String **url**()

## **class SubjectDiscipline**

**Beschreibung** Klasse, welche ein Vertiefungsfach repräsentiert

### **Attribute**

- private String **id**  
ID des Vertiefungsfachs
- private String **name**  
Name des Vertiefungsfachs

### **Konstruktoren**

- public **SubjectDiscipline**(Object params)

## **class SubjectDisciplineCollection**

**Beschreibung** Klasse, welche eine Collection von Vertiefungsfächern repräsentiert

### **Attribute**

- private Collection<SubjectDiscipline> **subjectDisciplines**  
Die Collection der Vertiefungsfächer

### **Konstruktoren**

- public **SubjectDisciplineCollection**(Object params)

## **class TemplateManager**

**Beschreibung** Singleton, welches die Templates für die Views verwaltet. Jedes Template lässt sich über einen Key abrufen.

Die Templates werden zur Compile-Zeit in den Initialisierungscode hineingeladen und sind somit bereits beim Start der Applikation im Template-Manager vorhanden. Der Vorteil dieses Vorgehens ist, dass man beim Laden der einzelnen Views nicht auf eine Antwort vom Server (welche das Template enthält) warten muss.

### **Attribute**

- private static TemplateManager **instance**  
Die aktuelle Instanz des TemplateManagers
- private Object **templates**  
Das Objekt, welches die Templates enthält, auf welche man mittels des Namens zugreifen kann.

### **Konstruktoren**

- private **TemplateManager**()

### **Methoden**

- public static TemplateManager **getInstance()**  
Methode zum Erhalten der aktuellen Instanz des TemplateManagers  
**Rückgabe** Gibt die aktuelle Instanz des TemplateManagers zurück
- public Object **getTemplate**(String key)  
Methode zum Erhalten eines Templates, welches unter dem Key gespeichert ist

#### **Parameter**

**key** Der Key zur Identifizierung des Templates

**Rückgabe** Gibt eine Funktion (ein Underscore-Template) zurück, die mit Parametern aufgerufen werden kann und deren Rückgabewert ein befülltes Template als String ist

## Package `client.model.user`

### `class PassedModuleCollection`

**Beschreibung** Die Collection aller bestandenen Module, welche zu einem Nutzer gehört

#### Konstruktoren

- `public PassedModuleCollection(Object params)`

#### Methoden

- `public String url()`

### `class SessionInformation`

**Beschreibung** Die Informationen zur aktuellen Session der WebApp, welche in einem Cookie gespeichert werden. Diese Informationen werden in einem Singleton gespeichert, da es nicht mehr als ein SessionInformation-Objekt geben darf.

#### Attribute

- `private String accessToken`  
Das Access-Token, welches vom Server generiert wurde
- `private static SessionInformation instance`  
Die aktuelle Instanz der SessionInformation
- `private boolean loggedIn`  
Ob der Nutzer bereits eingeloggt ist
- `private String state`  
Ein String, welcher an den Server mitgesendet wird und anschließend abgeglichen wird.
- `private Student student`  
Das Objekt, das den Student/Nutzer beschreibt

#### Konstruktoren

- `private SessionInformation()`

## Methoden

- public void **generateState()**  
Methode, die ein neues StateAttribut generiert
- public static SessionInformation **getInstance()**  
Methode, welche die aktuelle Instanz der SessionInformation zurück gibt.

## class Student

**Beschreibung** Klasse, welche den Studenten beschreibt

### Attribute

- private String **discipline**  
Das Fach des Nutzers
- private PassedModuleCollection **passedModules**  
Die Collection der bestandenen Module des Studenten
- private String **studystartCycle**  
Ob der Student im Sommer- (SS) oder Wintersemester (WS) sein Studium begonnen hat.
- private int **studyStartYear**  
Das Jahr, in dem der Student das Studium begonnen hat
- private String **userId**  
Die ID des Nutzers
- private String **username**  
Der Nutzernname

### Konstruktoren

- public **Student**(Object params)

## Package client.router

### class MainRouter

**Beschreibung** Klasse, welche das Routing zu den verschiedenen Seiten hin übernimmt. Die Methoden stoßen dann jeweils die entsprechenden Änderungen im View an.

### Konstruktoren

- public **MainRouter()**

## Methoden

- public void **comparisonPage**(String plan1Id, String plan2Id)  
Methode zum Anzeigen der Pläne-Vergleichen-Seite  
*/compare/{plan1Id}/{plan2Id}*

### Parameter

**plan1Id** Die ID des ersten Plans

**plan2Id** Die ID des zweiten Plans

- public void **editPage**(String planId)  
Methode zum Anzeigen der Plan-Bearbeiten-Seite  
*/plan/{planId}*

### Parameter

**planId** Die ID des Plans

- public void **generationWizard**(int planId)  
Methode zum Anzeigen des Generierungs-Wizards  
*/plan/{planId}/generate*
- public void **handleLogin**()  
Methode zum Verarbeiten der OAuth-Antwort vom Server. Diese Methode leitet dann an die entsprechende Seite (Hauptseite oder Login) weiter.  
*/processLogin*
- public void **loginPage**()  
Methode zum Anzeigen der Login-Seite  
*/login*
- public void **mainPage**()  
Methode zur Anzeige der Hauptseite  
*/*
- public void **showProfile**()  
Methode zum Anzeigen des Profils  
*/profile*
- public void **signUpWizard**()  
Methode zum Anzeigen des Registrierungs-Wizards  
*/signup*

## Package client.storage

### class Sync

**Beschreibung** Dieses Objekt kapselt die Methoden zur Synchronisierung mit Server oder Cookies, welche von den verschiedenen Model-Typen verwendet werden.

Die Methoden sind von Backbone spezifiziert und werden jeweils Submethoden enthalten, welche die einzelnen Methoden behandeln

### Konstruktoren

- public **Sync()**

## Methoden

- public void **CookieSync**(String method, Object model, Object options)  
Methode zur Speicherung/zum Abruf von Model-Daten aus Cookies

### Parameter

- method** Die Methode "create", "read", "update" oder "delete"
- model** Das zu speicherende Model
- options** Die Optionen, welche bei Backbone.sync verfügbar sind

- public void **OAuthSync**(String method, Object model, Object options)  
Methode zur Synchronisierung mittels OAuth-REST-API

### Parameter

- method** Die Methode "create", "read", "update" oder "delete"
- model** Das zu speicherende Model
- options** Die Optionen, welche bei Backbone.sync verfügbar sind

## Package client.view.components.filter

### class FilterComponent

**Beschreibung** Klasse, welche die Anzeige eines Filters kapselt.

#### Attribute

- private Filter **filter**  
Der Filter, welcher visualisiert wird
- private String **id**  
Die ID des Filters

#### Konstruktoren

- public **FilterComponent**(Object params)

## Methoden

- public Filter **getFilter()**  
Methode zum Erhalten des Filter-Modells
- public String **getId()**  
Methode zum Erhalten der ID des Filters
- Rückgabe** Die ID des Filters
- public abstract void **onSelect()**  
Methode, welche bei der Auswahl eines Filterobjekts ausgeführt wird

## **class ModuleFilter**

**Beschreibung** Ein Filter, welcher die Menge der angezeigten Module auf Basis von Filtern einschränkt

### **Attribute**

- private Collection<FilterComponent> **filterComponents**  
Die Filterkomponenten, welche auswählbar sind

### **Konstruktoren**

- public **ModuleFilter**(Object params)

### **Methoden**

- public FilterCollection **getFilters()**  
Methode, die die ausgewählten Filter zurückgibt
- public void **onSearch()**  
Methode, die bei der Suche ausgeführt wird. Diese kann über den Konstruktor als Callback gesetzt werden

## **class RangeFilter**

**Beschreibung** Filter, welcher durch die Auswahl eines Bereichs aktiviert wird

### **Attribute**

- private int **max**  
Das Maximum des auswählbaren Bereichs
- private int **min**  
Das Minimum des auswählbaren Bereichs

### **Konstruktoren**

- public **RangeFilter**(Object params)

### **Methoden**

- public void **onSelect()**  
Methode, welche bei der Auswahl eines Filterobjekts ausgeführt wird

## **class SelectFilter**

**Beschreibung** Filter, welcher durch die Auswahl eines Elements aktiviert wird

### **Attribute**

- private Collection<String> **options**  
Die vorhandenen Optionen des Filters

### **Konstruktoren**

- public **SelectFilter**(Object params)

### **Methoden**

- public void **onSelect()**  
Methode, welche bei der Auswahl eines Filterobjekts ausgeführt wird

## **class TextFilter**

**Beschreibung** Filter, welcher durch die Eingabe eines Suchtexts aktiviert wird

### **Konstruktoren**

- public **TextFilter**(Object params)

### **Methoden**

- public void **onSelect()**  
Methode, welche bei der Auswahl eines Filterobjekts ausgeführt wird

## **Package client.view.components.uielement**

### **class ModuleBox**

**Beschreibung** Diese Klasse repräsentiert eine einzelne Box, welche ein Modul darstellt

### **Attribute**

- private boolean **isRemovable**  
Ob das Modul entfernbar ist
- private Module **module**  
Das darzustellende Modul

### **Konstruktoren**

- public **ModuleBox**(Object params)

## Methoden

- public void **click()**  
Methode, welche beim Klick auf das Modul ausgeführt wird.
- public void **removeModule()**  
Methode, welche das Modul entfernt
- public void **setRedBorder**(boolean setBorder)  
Möglichkeit, einen roten Rand um das Modul zu setzen

## Parameter

**setBorder** Ob der Rahmen gesetzt werden soll (true) oder nicht (false)

## class ModuleFinder

**Beschreibung** Klasse, welche den Modul-Finder repräsentiert. Dieser erlaubt es, nach Modulen zu suchen und diese (wenn es sich um eine Sidebar handelt) in einen Plan zu ziehen.

## Attribute

- private Method **isPlaced**  
Methode, die prüfen kann, ob Module bereits in den Plan gesetzt sind. Diese wird im Konstruktor übergeben.
- private boolean **isPreferencable**  
Ob Module präferierbar sind (dies ist nur möglich, wenn sie im Kontext eines Plans geladen sind)
- private boolean **isSidebar**  
Ob es sich bei dem Modul-Finder um eine Sidebar handelt. In diesem Fall können Module aus dem Modul-Finder in den Plan per Drag and Drop gezogen werden.
- private SearchCollection **moduleCollection**  
Die Collection der Module, welche geladen sind
- private ModuleFilter **moduleFilter**  
Die Filteransicht
- private ModuleList **moduleList**  
Die Modulansicht

## Konuktoren

- public **ModuleFinder**(Object params)

## Methoden

- public void **onSearch()**  
Methode, die bei der Suche nach Modulen aufgerufen wird.

## **class ModuleInfoSidebar**

**Beschreibung** Klasse, welche die Modul-Informationen-Sidebar kapselt.

### **Attribute**

- `private Module module`  
Modul, welches angezeigt werden soll
- `private Collection<ModuleBox> moduleBox`  
Die ModuleBoxen, welche als Abhängigkeiten angezeigt werden.

### **Konstruktoren**

- `public ModuleInfoSidebar (Object params)`

### **Methoden**

- `public void onChange ()`  
Methode, die bei Veränderungen des Moduls aufgerufen wird (notwendig wegen dynamischem Laden des Moduls).
- `public void onClose ()`  
Methode, die beim Schließen der ModuleInfoSidebar aufgerufen wird.

## **class ModuleList**

**Beschreibung** Klasse, die die Anzeige einer Liste von Modulen kapselt.

### **Attribute**

- `private boolean isPreferable`  
Ob die Module in den Boxen präferierbar sind (nur im Zusammenhang mit einem Plan)
- `private Collection<ModuleBox> moduleBox`  
Die Modulboxen, in welchen die Module angezeigt werden.
- `private SearchCollection modules`  
Die Liste der Module, welche angezeigt werden sollen

### **Konstruktoren**

- `public ModuleList (Object params)`

## **class NotificationBox**

**Beschreibung** Klasse, welche eine Benachrichtigungsbox kapselt

### **Attribute**

- private Notification **notification**  
Die anzuzeigende Benachrichtigung

### **Konstruktoren**

- public **NotificationBox**(Object params)

### **Methoden**

- public void **blurOut()**  
Die Methode, welche nach einem gegebenen Zeitintervall die Benachrichtigung ausblendet
- public void **onClose()**  
Die Methode, welche die Benachrichtigungsbox schließt

## **class PassedModulePlan**

**Beschreibung** Plan der bestandenen Module

### **Attribute**

- private Collection<Semester> **semester**  
Die Semester, in welche die bestandenen Module geordnet sind.
- private Student **student**  
Der Student, der die Module bestanden hat

### **Konstruktoren**

- public **PassedModulePlan**(Object params)

## **class Plan**

**Beschreibung** Klasse, welche die Ansicht des Studienplans kapselt

### **Attribute**

- private String **align**  
Die Ausrichtung des Studienplans: Normalerweise links, im Fall des Vergleichs aber auch rechts.
- private Plan **plan**  
Das Modell-Objekt des Plans
- private Collection<Semester> **semester**  
Die Semester des Plans

## Konstruktoren

- public **Plan**(Object params)

## Methoden

- public void **addSemester()**  
Methode zum Hinzufügen neuer Semester
- public void **onChange()**  
Methode, welche bei Änderungen des Plans aufgerufen wird, und welche auch die onChange()-Methoden der Semester aufruft.

## class PlanHeadBar

**Beschreibung** Klasse, welche die Kopfzeile des Plans kapselt

## Attribute

- private Plan **plan**  
Der Plan, der angezeigt wird.

## Konstruktoren

- public **PlanHeadBar**(Object params)

## Methoden

- public void **onChange()**  
Die Methode, welche bei einer Änderung des Plans aufgerufen wird.

## class PlanListElement

**Beschreibung** Klasse, welche ein Element einer PlanList kapselt

## Attribute

- private Plan **plan**  
Der Plan, der angezeigt werden soll

## Konstruktoren

- public **PlanListElement**(Object params)

## Methoden

- public void **delete()**  
Methode, welche beim Klick auf „Löschen“ aufgerufen wird

- public void **duplicate()**  
Methode, welche beim Klick auf „Duplizieren“ aufgerufen wird
- public void **export()**  
Methode, welche beim Klick auf „Exportieren“ aufgerufen wird
- public void **show()**  
Methode, welche beim Klick auf „Anzeigen“ aufgerufen wird

### **class ProposedHeadBar**

**Beschreibung** Kopfzeile, welche angezeigt wird, wenn ein vorgeschlagener Plan angezeigt wird

#### **Konuktoren**

- public **ProposedHeadBar**(Object params)

### **class RegularHeadBar**

**Beschreibung** Kopfzeile, welche angezeigt wird, wenn ein regulärer Plan angezeigt wird

#### **Konuktoren**

- public **RegularHeadBar**(Object params)

### **Methoden**

- public void **generate()**  
Methode, welche die Generierung eines neuen Plans anstößt
- public void **rename()**  
Methode, welche die Umbenennung eines Plans ausführt
- public void **verify()**  
Methode, welche die Verifizierung eines neuen Plans anstößt

### **class Semester**

**Beschreibung** Klasse, welche ein Semester innerhalb eines Studienplans kapselt

#### **Attribute**

- private String **align**  
Die Ausrichtung des Studienplans: Normalerweise links, im Fall des Vergleichs aber auch rechts.
- private boolean **isRemovable**  
Variable, die festsetzt, ob das Semester entfernbare ist

- `private Collection<ModuleBox> moduleBox`  
Die Modul-Boxen, welche die Module des Semesters anzeigen
- `private Semester moduleCollection`  
Die Module, welche in dem Semester liegen
- `private int number`  
Variable, die angibt, das wievielte Semester dieses ist (ab eins gezählt)

### Konstruktoren

- `public Semester(Object params)`

### Methoden

- `public void onDrop(Event event, Object ui)`  
Methode, welche beim Drop eines Moduls auf das Semester das Hinzufügen ausführt

#### Parameter

`event` Das JQuery-Event

`ui` Das JQuery-UI-Objekt

- `public void removeSemester()`  
Methode zum Löschen des aktuellen Semesters
- `public void scrollLeft()`  
Methode zum Links-Scrollen der Semesterleiste
- `public void scrollRight()`  
Methode zum Rechts-Scrollen der Semesterleiste

## Package `client.view.components.uipanel`

### `class ComparisonView`

**Beschreibung** Anzeige, welche zwei Pläne vergleicht

#### Attribute

- `private Plan plan1`  
Der linke Plan des Vergleichs
- `private Plan plan2`  
Der rechte Plan des Vergleichs

### Konstruktoren

- `public ComparisonView(Object params)`

## **class GenerationWizardComponent1**

**Beschreibung** Klasse, welche die erste Seite des Generierungs-Wizards kapselt.

### **Attribute**

- private ProposalInformation **information**  
Die notwendigen Informationen für die Generierung
- private Plan **plan**  
Plan, für welchen ein Vorschlag generiert werden soll

### **Konstruktoren**

- public **GenerationWizardComponent1**(Object params)

### **Methoden**

- public WizardComponent **next()**  
Gibt Instanz vom Typ GenerationWizardComponent2 zurück  
**Rückgabe** Instanz vom Typ GenerationWizardComponent2 (zweite Seite des Wizards)
- public void **onChange()**  
Methode, welche bei Veränderungen des Seiteninhalts aufgerufen wird.

## **class GenerationWizardComponent2**

**Beschreibung** Klasse, welche die zweite Seite des Generierungs-Wizards kapselt.

### **Attribute**

- private ProposalInformation **information**  
Die notwendigen Informationen für die Generierung
- private ModuleFinder **moduleFinder**  
Der Modul-Finder, mit welchem die Module bewertet werden können.
- private Plan **plan**  
Plan, für welchen ein Vorschlag generiert werden soll

### **Konstruktoren**

- public **GenerationWizardComponent2**(Object params)

### **Methoden**

- public WizardComponent **next()**  
Gibt Instanz vom Typ GenerationWizardComponent3 zurück

**Rückgabe** Instanz vom Typ GenerationWizardComponent3 (dritte Seite des Wizards)

- public void **onChange()**

Methode, welche bei Veränderungen des Seiteninhalts aufgerufen wird.

## **class GenerationWizardComponent3**

**Beschreibung** Klasse, welche die dritte Seite des Generierungs-Wizards kapselt.

### **Attribute**

- private ProposalInformation **information**

Die notwendigen Informationen für die Generierung

- private Plan **plan**

Plan, für welchen ein Vorschlag generiert werden soll

### **Konstruktoren**

- public **GenerationWizardComponent3**(Object params)

### **Methoden**

- public WizardComponent **next()**

Gibt null zurück, da dies die letzte Seite des Wizards ist.

**Rückgabe** null

- public void **onChange()**

Methode, welche bei Veränderungen des Seiteninhalts aufgerufen wird.

## **class NotificationCentre**

**Beschreibung** Klasse, welche die Anzeige von Benachrichtigungen kapselt

### **Attribute**

- private Collection<NotificationBox> **notificationBox**

Die NotificationBoxes, welche die Benachrichtigungen anzeigen

- private NotificationCollection **notificationCollection**

Wird mittels NotificationCollection.getInstance() geladen. NotificationCentre ist ein Observer der NotificationCollection und zeigt ggf. neue Notifications an.

### **Konstruktoren**

- public **NotificationCentre**(Object params)

## Methoden

- `public void onChange()`  
Methode, welche aufgerufen wird, wenn sich die notificationCollection verändert.

## class PlanList

**Beschreibung** Klasse, welche eine Liste von Plänen kapselt

## Attribute

- `private PlanCollection planCollection`  
Die Pläne, welche angezeigt werden sollen
- `private Collection<PlanListElement> planListElements`  
Die Elemente, welche in der Liste angezeigt werden

## Konstruktoren

- `public PlanList(Object params)`

## Methoden

- `public void onActionSelection()`  
Listener, der aufgerufen wird, wenn eine Aktion für einen/mehrere Pläne aufgerufen wird.
- `public void onChange()`  
Methode, welche bei einer Veränderung der Pläne aufgerufen wird

## class ProposalSidebar

**Beschreibung** Sidebar, welche angezeigt wird, wenn ein generierter Plan angezeigt wird.

## Konstruktoren

- `public ProposalSidebar(Object params)`

## Methoden

- `public void delete()`  
Methode, welche aufgerufen wird, wenn der Plan gelöscht werden soll
- `public void save()`  
Methode, welche aufgerufen wird, wenn der Plan gespeichert werden soll
- `public void saveAs()`  
Methode, welche aufgerufen wird, wenn der Plan unter anderem Namen gespeichert werden soll

## **class SignUpWizardComponent1**

**Beschreibung** Klasse, welche die erste Seite des Registrierungs-Wizards kapselt

### **Attribute**

- private Student **student**  
Der Student, der sich registriert

### **Konstruktoren**

- public **SignUpWizardComponent1** (Object params)

### **Methoden**

- public WizardComponent **next()**  
Gibt ein Objekt vom Typ SignUpWizardComponent2 zurück  
**Rückgabe** Objekt vom Typ SignUpWizardComponent2 (zweite Seite des Wizards)
- public void **onChange()**  
Methode, welche bei Veränderungen des Seiteninhalts aufgerufen wird.

## **class SignUpWizardComponent2**

**Beschreibung** Klasse, welche die zweite Seite des Registrierungswizards kapselt

### **Attribute**

- private ModuleFinder **moduleFinder**  
Der ModulFinder, welcher genutzt wird, um die Module als bestanden zu markieren.
- private PassedModulePlan **passedModules**  
Der Plan der vergangenen Semester mit bestandenen Modulen.
- private Student **student**  
Der Student

### **Konstruktoren**

- public **SignUpWizardComponent2** (Object params)

### **Methoden**

- public WizardComponent **next()**  
Gibt null zurück, da dies die letzte Seite des SignUpWizards ist

**Rückgabe** null

- public void **onChange()**

Methode, welche bei Veränderungen des Seiteninhalts aufgerufen wird.

## **class WizardComponent**

**Beschreibung** Klasse, welche eine Komponente eines Wizards kapselt. Die Wizards sind nach dem Konzept des State-Machine-Patterns aufgebaut

### **Konstruktoren**

- public **WizardComponent** (Object params)

### **Methoden**

- public abstract WizardComponent **next()**

Methode, welche die View-Component der nächsten Seite des Wizards zurückgibt

**Rückgabe** Die View-Component der nächsten Seite des Wizards

## **Package client.view**

### **class MainView**

**Beschreibung** Die MainView kapselt die Benutzeroberfläche, sie kann mittels gegebener Funktionen mit beliebigem Inhalt in Form von Objekten des Typs Backbone.View befüllt werden.

### **Attribute**

- private BackboneView **content**  
Die View, welche im Content-Bereich angezeigt wird.
- private BackboneView **header**  
Die View, welcher im Header angezeigt wird.
- private NotificationCentre **notificationCentre**  
Das NotificationCentre, welches die Anzeige von Benachrichtigungen übernimmt.
- private SessionInformation **sessionInformation**  
Die Informationen zur Session

### **Konstruktoren**

- public **MainView()**

### **Methoden**

- public void **setContent** (BackboneView content, Object options)  
Mit dieser Funktion kann man den Content-Bereich des MainViews setzen.

**Parameter**

**content** Der zu setzende Content-Bereich

**options** Objekt, mit welchem der Header initialisiert werden soll

- public void **setHeader**(BackboneView header, Object options)  
Mit dieser Methode kann man den Header der MainView setzen

**Parameter**

**header** Der zu setzende Header

**options** Objekt, mit welchem der Header initialisiert werden soll

**Package client.view.subview****class ComparisonPage**

**Beschreibung** Klasse, welche die Vergleichsseite kapselt

**Attribute**

- private ComparisonView **comparisonView**  
Das ComparisonView-Element, welches angezeigt wird

**Konstruktoren**

- public **ComparisonPage**(Object params)

**class Header**

**Beschreibung** Klasse, welche den Header der Seite kapselt

**Attribute**

- private SessionInformation **sessionInformation**  
Das SessionInformation-Objekt, welches zur Anzeige des Headers benötigt wird

**Konstruktoren**

- public **Header**(Object params)

**class LoginPage**

**Beschreibung** Klasse, welche die Login-Seite kapselt

**Konstruktoren**

- public **LoginPage**(Object params)

## **class MainPage**

**Beschreibung** Klasse, welche die Hauptseite kapselt

### **Attribute**

- `private PlanList planList`  
Die anzuseigende Plan Liste

### **Konstruktoren**

- `public MainPage (Object params)`

### **Methoden**

- `public void addPlan()`  
Methode zum Hinzufügen eines Plans

## **class PlanEditPage**

**Beschreibung** Klasse, welche die Plan-Bearbeitungsansicht kapselt

### **Attribute**

- `private PlanHeadBar headBar`  
Kopfzeile, die angezeigt wird
- `private Plan plan`  
Plan, der angezeigt wird
- `private BackboneView sidebar`  
Sidebar, welche angezeigt wird (in der Hauptansicht der ModuleFinder)

### **Konstruktoren**

- `public PlanEditPage (Object params)`

### **Methoden**

- `public void hideModuleDetails()`  
Methode zum Schließen einer ModuleInfoSidebar
- `public void showModuleDetails (Module module)`  
Methode zum Aufrufen einer ModuleInfoSidebar

### **Parameter**

`module` Modul, zu welchem die Info-Sidebar angezeigt werden soll

## **class ProfilePage**

**Beschreibung** Klasse, welche die Profil-Seite kapselt

### **Attribute**

- private PassedModulePlan **passedModules**  
Ein Plan mit bestandenen Modulen
- private ModuleFinder **sidebar**  
Die ModuleFinder-Sidebar

### **Konstruktoren**

- public **ProfilePage**(Object params)

### **Methoden**

- public void **close()**  
Leitet zurück zur Hauptseite ( MainPage ) und speichert die Änderungen.
- public void **hideModuleDetails()**  
Methode zum Schließen einer ModuleInfoSidebar
- public void **showModuleDetails**(Module module)  
Methode zum Aufrufen einer ModuleInfoSidebar

#### **Parameter**

**module** Modul, zu welchem die Info-Sidebar angezeigt werden soll

## **class WizardPage**

**Beschreibung** Diese Klasse kapselt die Anzeige von Wizards in einem State-Machine-Pattern. curView wird bei der Initialisierung gesetzt. WizardPage kann deshalb für jedwede Art von Wizards verwendet werden.

WizardPage implementiert Backbone.Event, sodass man sich als Observer der WizardPage registrieren kann. Hier kann man auf das wizardComplete-Event reagieren, welches signalisiert, dass der Wizard beendet ist.

### **Attribute**

- public WizardComponent **curView**  
Die aktuell angezeigte WizardComponent

### **Attribute**

- public WizardComponent **curView**  
Die aktuell angezeigte WizardComponent

## Konstruktoren

- public **WizardPage**(Object params)

## Methoden

- public void **next()**

Methode, welche die nächste Seite des Wizards aufruft oder diesen beendet.

## Package server.filter

### interface AttributeFilter

**Beschreibung** Repräsentiert einen Filter für ein bestimmtes Module-Attribut.

## Methoden

- public FilterDescriptor **getDescriptor()**  
Liefert die zum AttributeFilter gehörende Filterbeschreibung.  
**Rückgabe** die Filterbeschreibung
- public FilterType **getFilterType()**  
Liefert den Filter-Typ des AttributeFilters.  
**Rückgabe** der Filter-Typ

### class CategoryFilter

**Beschreibung** Repräsentiert einen Kategorie-Wahlfilter mit den Modulkategorien als Wahlmöglichkeiten.

## Konstruktoren

- public **CategoryFilter**(int selection)  
Erzeugt einen neuen Kategorie-Wahlfilter mit gegebener festgelegter Auswahl.

### Parameter

**selection** die Nummer des ausgewählten Elements

## Methoden

- public FilterDescriptor **getDescriptor()**
- public List<String> **getItems()**

### class CompulsoryFilter

**Beschreibung** Repräsentiert einen Pflicht-/Wahlmodul-Auswahlfilter mit Filterung nach Pflicht-, Wahlmodulen oder beidem als Wahlmöglichkeiten.

## Konstruktoren

- public **CompulsoryFilter**(int selection)  
Erzeugt einen neuen Pflicht-/Wahlmodul-Auswahlfilter mit gegebener festgelegter Auswahl.

### Parameter

**selection** die Nummer des ausgewählten Elements

## Methoden

- public FilterDescriptor **getDescriptor**()
- public List<String> **getItems**()

## class ContainsFilter

**Beschreibung** Repräsentiert einen Textsuch-Attribut-Filter.

### Attribute

- protected String **substring**  
Der Suchstring.

## Konstruktoren

- protected **ContainsFilter**(String substring)  
Erzeugt einen neuen Textsuch-Filter mit gegebenem Suchstring.

### Parameter

**substring** der Suchstring

## Methoden

- public Condition **getCondition**()  
Liefert eine Filterbedingung, die das Vorkommen des Substrings im Attributwert fordert.

**Rückgabe** die Filterbedingung als jOOQ-Condition-Objekt

- public abstract FilterDescriptor **getDescriptor**()
- public FilterType **getFilterType**()

- public String **getSubstring**()  
Liefert den Substring, nach welchem gefiltert werden soll.

**Rückgabe** der Substring

## class CreditPointsFilter

**Beschreibung** Repräsentiert einen ECTS-Intervall-Filter.

## Konstruktoren

- public **CreditPointsFilter**(int lower, int upper, int min, int max)  
Erzeugt einen neuen ECTS-Intervall-Filter mit gegebenen Schranken.

## Parameter

- lower** untere Schranke des Filters
- upper** obere Schranke des Filters
- min** minimale untere Schranke des Filters
- max** maximale obere Schranke des Filters

## Methoden

- public FilterDescriptor **getDescriptor**()

## class CycleTypeFilter

**Beschreibung** Repräsentiert einen Turnus-AuswahlfILTER mit Filterung nach Winter-, Sommersemester oder beidem als Wahlmöglichkeiten.

## Konstruktoren

- public **CycleTypeFilter**(int selection)  
Erzeugt einen neuen Turnus-AuswahlfILTER mit gegebener festgelegter Auswahl.

## Parameter

- selection** die Nummer des ausgewählten Elements

## Methoden

- public FilterDescriptor **getDescriptor**()
- public List<String> **getItems**()

## class DisciplineFilter

**Beschreibung** Repräsentiert einen Fachrichtungs-Wahlfilter mit den Fachrichtungen als Wahlmöglichkeiten.

## Konstruktoren

- public **DisciplineFilter**(int selection)  
Erzeugt einen neuen Fachrichtungs-AuswahlfILTER mit gegebener festgelegter Auswahl.

## Parameter

- selection** die Nummer des ausgewählten Elements

## Methoden

- public FilterDescriptor **getDescriptor()**
- public List<String> **getItems()**

## interface Filter

**Beschreibung** Repräsentiert einen Filter für Module über eine Filterbedingung.

## Methoden

- public Condition **getCondition()**  
Gibt die Filterbedingung als jOOQ-Condition-Objekt zurück.
- Rückgabe** die Filterbedingung

## enum FilterDescriptor

**Beschreibung** Beschreibungen der nach außen sichtbaren Filterklassen für den Client.

## Enum-Konstanten

- **CATEGORY**  
Beschreibt den Kategorie-Wahlfilter.
- **COMPULSORY**  
Beschreibt den Pflicht-/Wahlveranstaltungs-Wahlfilter.
- **CREDIT\_POINTS**  
Beschreibt den ECTS-Intervall-Filter.
- **CYCLE\_TYPE**  
Beschreibt den Turnus-Wahlfilter.
- **DISCIPLINE**  
Beschreibt den Fachrichtungs-Wahlfilter.
- **NAME**  
Beschreibt den Modulnamen-Textfilter.
- **TYPE**  
Beschreibt den Modultyp-Wahlfilter.

## Methoden

- public String **attributeName()**  
Liefert den Namen des Module-Attributs, nach welchem durch beschriebenen Filter gefiltert werden soll.  
**Rückgabe** der Attribut-Name
- public AttributeFilter **defaultFilter()**  
Liefert ein Default-Objekt des beschriebenen Filtertyps.  
**Rückgabe** das Default-Objekt
- public String **filterName()**  
Liefert den Namen des Filters (für die Benutzeroberfläche).

**Rückgabe** den Filternamen

- `public int id()`  
Liefert die ID des Filters.

**Rückgabe** die ID des Filters

- `public Field<Object> toField()`  
Gibt ein jOOQ-Field-Objekt zurück, das des Filters attributeName kapselt.

**Rückgabe** das jOOQ-Field-Objekt

- `public JSONObject toJson()`  
Liefert eine JSON-Präsentation des beschriebenen Filters für den Client.

**Rückgabe** eine JSON-Präsentation des beschriebenen Filters

- `public String tooltip()`  
Liefert den zum Filter gehörenden Tooltip (für die Benutzeroberfläche).

**Rückgabe** das Tooltip zum Filter

- `public static FilterDescriptor valueOf(String name)`
- `public static FilterDescriptor values()`

## enum FilterType

**Beschreibung** Aufzählung der verschiedenen AttributeFilter-Typen.

### Enum-Konstanten

- **CONTAINS**  
Repräsentiert den Filtertyp ContainsFilter (siehe ContainsFilter).
- **LIST**  
Repräsentiert den Filtertyp ListFilter (siehe ListFilter).
- **RANGE**  
Repräsentiert den Filtertyp RangeFilter (siehe RangeFilter).

### Methoden

- `public abstract JSONObject defaultJsonValue(AttributeFilter defaultFilter)`  
Liefert eine JSON-Präsentation der Werte des übergebenen Default-Filters.

#### Parameter

`defaultFilter` das Filter-Objekt mit Default-Werten

**Rückgabe** die Werte des Default-Filters

- `public abstract JSONObject toJsonSpecification(AttributeFilter defaultFilter)`  
Liefert den Spezifikations-Abschnitt der JSON-Präsentation des übergebenen Default-Filters.

### Parameter

**defaultFilter** der Default-Filter

**Rückgabe** die Spezifikation des Filters als JSON-Objekt

- public static FilterType **valueOf**(String name)
- public static FilterType **values**()

## class ListFilter

**Beschreibung** Repräsentiert einen Listenauswahl-Attribut-Filter.

### Attribute

- protected int **selection**  
Die Nummer des ausgewählten Elements.

### Konstruktoren

- protected **ListFilter**(int selection)  
Erzeugt einen neuen Auswahlfilter mit gegebener festgelegter Auswahl.

### Parameter

**selection** die Nummer des ausgewählten Elements

### Methoden

- public Condition **getCondition**()  
Liefert eine Filterbedingung, die die Übereinstimmung des untersuchten Attributswertes mit der festgelegten Auswahl (selection) fordert.

**Rückgabe** die Filterbedingung als jOOQ-Condition-Objekt

- public abstract FilterDescriptor **getDescriptor**()
- public FilterType **getFilterType**()
- public abstract List<String> **getItems**()  
Liefert alle Wahlmöglichkeiten dieses Auswahl-Filters als Strings.

**Rückgabe** die Wahlmöglichkeiten des Auswahl-Filters

- public int **getSelection**()  
Liefert die Nummer des selektierten Elements.

**Rückgabe** die Element-Nummer

## class MultiFilter

**Beschreibung** Bündelt mehrere Filter zu einem Einzigen mittels UND-Verknüpfung der Filterbedingungen.

## Konstruktoren

- public **MultiFilter**(List<Filter> filters)  
Erzeugt einen neuen MultiFilter aus gegebenen Unterfiltern.

### Parameter

**filters** die Unterfilter, die gebündelt werden sollen

## Methoden

- public Condition **getCondition**()  
Gibt die UND-Verknüpfung der Filterbedingungen der gebündelten Filter als Filterbedingung zurück, oder eine konstant wahre Filterbedingung, falls filters leer ist.  
**Rückgabe** Die neue Filterbedingung als jOOQ-Condition-Objekt
- public List<Filter> **getFilters**()  
Gibt alle Filter zurück, die von diesem MultiFilter gebündelt werden.  
**Rückgabe** die gebündelten Filter

## class NameFilter

**Beschreibung** Représentiert einen Modulnamen-Textsuchfilter.

## Konstruktoren

- public **NameFilter**(String substring)  
Erzeugt einen neuen Modulnamen-Textsuchfilter mit gegebenem Suchstring.  
**Parameter**  
**substring** der Suchstring

## Methoden

- public FilterDescriptor **getDescriptor**()

## class RangeFilter

**Beschreibung** Représentiert einen Intervall-Beschränkungs-Filter für ganzzahlige Attribute.

## Attribute

- protected int **lower**  
Die untere Schranke des Filters.
- protected int **max**  
Die maximale untere Schranke des Filters.

- `protected int min`  
Die minimale untere Schranke des Filters.
- `protected int upper`  
Die obere Schranke des Filters.

## Konstruktoren

- `protected RangeFilter(int lower, int upper, int min, int max)`  
Erzeugt einen neuen Intervall-Filter mit gegebenen Schranken.

### Parameter

- `lower` untere Schranke des Filters
- `upper` obere Schranke des Filters
- `min` minimale untere Schranke des Filters
- `max` maximale obere Schranke des Filters

## Methoden

- `public Condition getCondition()`  
Liefert eine Filterbedingung, die vom Attributwert das Einhalten der festgelegten Intervall-Grenzen fordert.

**Rückgabe** die Filterbedingung als jOOQ-Condition-Objekt

- `public abstract FilterDescriptor getDescriptor()`
- `public FilterType getFilterType()`
- `public int getLower()`  
Liefert die festgelegte untere Schranke des Filters.

**Rückgabe** die untere Schranke

- `public int getMax()`  
Liefert den Maximalwert der oberen Schranke des Filters.

**Rückgabe** den Maximalwert der oberen Schranke

- `public int getMin()`  
Liefert den Mindestwert der unteren Schranke des Filters.

**Rückgabe** der Mindestwert der unteren Schranke

- `public int getUpper()`  
Liefert die festgelegte obere Schranke des Filters.

**Rückgabe** die obere Schranke

## class TrueFilter

**Beschreibung** Stellt einen Filter dar, der alle Module zulässt (und dessen Filterbedingung daher konstant wahr ist).

## Konstruktoren

- `public TrueFilter()`  
Erzeugt einen neuen TrueFilter.

## Methoden

- `public Condition getCondition()`  
Gibt eine konstant wahre Filterbedingung zurück.  
**Rückgabe** die Filterbedingung als jOOQ-Condition-Objekt

## `class TypeFilter`

**Beschreibung** Repräsentiert einen Modultyp-WahlfILTER mit den Modultypen als Wahlmöglichkeiten.

## Konstruktoren

- `public TypeFilter(int selection)`  
Erzeugt einen neuen Modultyp-WahlfILTER mit gegebener festgelegter Auswahl.

### Parameter

`selection` die Nummer des ausgewählten Elements

## Methoden

- `public FilterDescriptor getDescriptor()`
- `public List<String> getItems()`

## `Package server.generation`

### `interface Generator`

**Beschreibung** Das Interface Generator bietet die allgemeine Struktur eines Generierers.

## Methoden

- `public Plan generate(PartialObjectiveFunction objectiveFunction, Plan currentPlan, ModuleDao moduleDAO)`  
Die Methode generate() generiert einen vollständigen, optimierten und korrekten Studienplan. Hierzu nimmt sie einen angefangenen Studienplan entgegen, vervollständigt diesen zunächst nach System-Constraints und Zufall und optimiert ihn dann mit Hilfe der übergebenen Zielfunktion objectiveFunction. Die benötigten Module werden über den übergebenen ModuleDao erreicht.

**Parameter**

- **objectiveFunction** Die Zielfunktion, anhand welcher optimiert werden soll
- **currentPlan** der bereits bestehende Plan
- **moduleDAO** die Module

**Rückgabe** Zurückgegeben wird ein vollständiger, korrekter und optimierter Studienplan vom Typ Plan

## **Package server.generation.objectivefunction**

**Siehe auch:**

- ObjectiveFunction

### **class AtomObjectiveFunction**

**Beschreibung** AtomObjectiveFunction ist eine Teilzielfunktion, die nur eine bestimmte Eigenschaft berücksichtigt.

**Konstruktoren**

- public AtomObjectiveFunction()

**Methoden**

- public abstract double evaluate(Plan plan)

### **class AverageObjectiveFunction**

**Beschreibung** Es wird bei der Auswertung der Teilzielfunktionen immer der Durchschnitt genommen.

**Konstruktoren**

- public AverageObjectiveFunction()

**Methoden**

- public double evaluate(Plan plan)

### **class MinimalECTSAtomObjectiveFunction**

**Beschreibung** Je geringer die Gesamtanzahl der ECTS in einem Studienplan, desto besser ist die Bewertung von MinimalECTSAtomObjectiveFunction. Zu beachten ist jedoch, dass sobald ein gewisser Schwellwert unterschritten wird, alle Studienpläne die Bestnote erhalten.

## Konstruktoren

- public **MinimaleCTSAtomObjectiveFunction()**  
Setzt den Schwellwert auf 0, der Studienplan mit den wenigsten ECTS wird somit immer am besten bewertet.
- public **MinimaleCTSAtomObjectiveFunction(int threshold)**

## Parameter

**threshold** ist die Menge an ECTS, ab der diese Funktion den Bestwert zurückgibt  
– eine niedrigere ECTS-Zahl verbessert die Bewertung also nicht.

## Methoden

- public double **evaluate(Plan plan)**

## **class MinimalSemestersAtomObjectiveFunction**

**Beschreibung** Minimiert nach der Semesteranzahl. Ein Studienplan mit einem Semester erhält demzufolge die beste Bewertung.

## Konstruktoren

- public **MinimalSemestersAtomObjectiveFunction()**

## Methoden

- public double **evaluate(Plan plan)**

## **class MinimalStandardAverageDeviation**

↪**ECTSAtomObjectiveFunction**

**Beschreibung** Minimiert die durchschnittliche Standardabweichung pro Semester des Plans. Ideal sind deshalb Pläne, bei denen jedes Semester genau gleich viele ECTS beinhaltet.

## Konstruktoren

- public **MinimalStandardAverageDeviationECTSAtomObjectiveFunction()**

## Methoden

- public double **evaluate(Plan plan)**

## **class ModulePreferencesAtomObjectiveFunction**

**Beschreibung** Optimiert nach den Nutzerbewertungen. Ein Plan ist ideal, wenn alle Nutzerbewertungen berücksichtigt wurden, also alle positiv bewerteten Module im Studienplan zu finden sind und kein negativ bewertetes.

### **Konstruktoren**

- public **ModulePreferencesAtomObjectiveFunction()**

### **Methoden**

- public double **evaluate(Plan plan)**

## **class MultiplicationObjectiveFunction**

**Beschreibung** Multipliziert die Ergebnisse der Auswertung der Teilzielfunktionen auf.

### **Konstruktoren**

- public **MultiplicationObjectiveFunction()**

### **Methoden**

- public double **evaluate(Plan plan)**

## **class ObjectiveFunction**

**Beschreibung** Eine Zielfunktion dient zur Bündelung von Teilzielfunktionen, die alle gemeinsam ausgewertet werden.

### **Konstruktoren**

- public **ObjectiveFunction()**

### **Methoden**

- public void **add(PartialObjectiveFunction objective)**  
Fügt eine Teilzielfunktion zu dieser Zielfunktion hinzu.

#### **Parameter**

**objective** die hinzuzufügende Teilzielfunktion

- public abstract double **evaluate(Plan plan)**
- public Collection<PartialObjectiveFunction> **getSubFunctions()**  
Gibt subFunctions zurück.

**Rückgabe** die Collection subFunctions, bestehend aus PartialObjectiveFunctions

- public PartialObjectiveFunction **remove**(PartialObjectiveFunction objective)  
Entfernt eine Teilzielfunktion von dieser Zielfunktion

#### Parameter

**objective** die zu entfernende Zielfunktion

**Rückgabe** die entfernte Zielfunktion

## interface PartialObjectiveFunction

**Beschreibung** PartialObjectiveFunction beschreibt eine Teilzielfunktion. Da jede Zielfunktion Teil einer anderen Zielfunktion sein kann, ist jede Zielfunktion auch eine Teilzielfunktion.

#### Methoden

- public double **evaluate**(Plan plan)  
Bewertet einen Studienplan und gibt dementsprechend eine Fließkommazahl zwischen 0 und 1 zurück.

#### Parameter

**plan** der zu bewertende Plan

**Rückgabe** Wert zwischen 0 und 1 als Bewertung für den Plan, wobei ein Plan mit der Bewertung 1 ein idealer Plan ist.

## class ThresholdObjectiveFunction

**Beschreibung** Wertet alle Teilzielfunktionen erst ab der Überschreitung des Schwellwertes aus. Mittelmäßige Studienpläne können auf diese Weise äußerst schlechte Bewertungen erhalten. Bessere Studienpläne heben sich so stärker heraus.

#### Konuktoren

- public **ThresholdObjectiveFunction**()  
Setzt den Schwellwert auf 0.5
- public **ThresholdObjectiveFunction**(int threshold)  
Setzt den Schwellwert auf den angegebenen Wert

#### Parameter

**threshold** Der Schwellwert

#### Methoden

- public double **evaluate**(Plan plan)

## **Package server.generation.standard**

### **class Node**

**Beschreibung** Die abstrakte Klasse Node stellt einen Knoten eines Graphen dar.

#### **Konstruktoren**

- public **Node()**

#### **Methoden**

- public void **addParent**(Node node)

Die Methode addParent fügt der Liste parents den übergebenen Node hinzu.

##### **Parameter**

**node**

- public void **addSubNode**(Node node)

Die Methode addSubNode fügt dem innersten Knoten den übergebenen Knoten als inneren Knoten hinzu.

##### **Parameter**

**node** der neue innerste Knoten.

- public Module **getModule()**

Die Methode gibt das Modul zurück, welches der Knoten darstellt.

**Rückgabe** Das Modul, welches der Knoten repräsentiert.

- public Collection<Node> **getParents()**

Gibt die Liste parents zurück.

**Rückgabe** parent-moduls

- public Node **getSubNodes()**

Gibt eventuell enthaltene weitere Knoten zurück.

**Rückgabe** enthaltene Knoten

### **class NodeWithoutOutput**

**Beschreibung** Die Klasse NodeWithoutOutput erbt von Node und stellt Blätter der Graphenstruktur dar (bzw. Module, die nicht Voraussetzung für ein anderes Modul des Studienplans sind).

#### **Konstruktoren**

- public **NodeWithoutOutput()**

## **class NodeWithOutput**

**Beschreibung** Die Klasse NodeWithOutput erbt von Node und stellt einen inneren Knoten der Graphenstruktur dar, also ein Modul, welches Voraussetzung für andere Module ist.

### **Konstruktoren**

- public **NodeWithOutput()**

### **Methoden**

- public void **addChild(Node node)**

Die Methode addChild für der Liste children den übergebenen Knoten hinzu, zu dem der Knoten eine Ausgangskante hat.

#### **Parameter**

**node** der hinzuzufügende Knoten

- public Collection<Node> **getChildren()**

Die Methode getChildren gibt eine Liste mit allen Knoten zurück, zu denen der Knoten Ausgangskanten hat.

#### **Rückgabe** children-moduls

- public Collection<Node> **removeChild(Node node)**

Die Methode removeChild löscht den übergebenen Knoten aus der Liste children vom Typ Node. Zurückgegeben wird die dann aktuelle Liste Children.

#### **Parameter**

**node** der zu löschenende Knoten

**Rückgabe** die jetzt aktuelle Liste children.

## **class SimpleGenerator**

**Beschreibung** Die Klasse SimpleGenerator, welche das Interface Generator implementiert, stellt einen konkreten Generator dar. Dieser nutzt die Graphenstruktur, die Zielfunktion und die Credit-Point-Gewichtsfunktion.

### **Konstruktoren**

- public **SimpleGenerator()**

### **Methoden**

- public Plan **generate(PartialObjectiveFunction objectiveFunction, Plan currentPlan, ModuleDao moduleDAO)**

## **class WeightFunction**

**Beschreibung** Die Klasse WeightFunction stellt eine Credit-Point-Gewichtsfunktion dar. Sie bestimmt also die Anzahl an Credit-Points eines Knotens und aller seiner innerer Knoten.

### **Konstruktoren**

- `public WeightFunction()`

### **Methoden**

- `public int getWeight(Node node)`

Die Methode `getWeight` errechnet die Punktanzahl eines Knoten (`node`) und aller innerer Knoten dieses Knotens, addiert sie und gibt diese Ganzzahl zurück.

#### **Parameter**

`node` der Knoten, dessen Credit-Point-Anzahl bestimmt werden soll.

**Rückgabe** die Gesamtanzahl an Credit-Points des Knotens.

## **Package server.model**

## **class HibernateUtil**

**Beschreibung** Fabrik zur Erzeugung von SessionFactories. Diese sind das Hibernate-Äquivalent zu Datenbankverbindungen.

### **Konstruktoren**

- `HibernateUtil()`

### **Methoden**

- `static SessionFactory getModuleDataSessionFactory()`

Erzeugt die SessionFactory zum Zugriff auf die Modul-Datenbank aus der entsprechenden Konfigurationsdatei

**Rückgabe** die SessionFactory

- `static SessionFactory getUserDataSessionFactory()`

Erzeugt die SessionFactory zum Zugriff auf die Nutzer-Datenbank aus der entsprechenden Konfigurationsdatei

**Rückgabe** die SessionFactory

## **Package server.model.moduledata**

### **class Category**

**Beschreibung** Modelliert eine Modul-Kategorie.

#### **Konstruktoren**

- public **Category()**

#### **Methoden**

- public int **getCategoryId()**  
**Rückgabe** gibt die eindeutige Kategorie-ID zurück
- public String **getName()**  
**Rückgabe** gibt den Namen der Kategorie zurück
- public boolean **isSubject()**  
**Rückgabe** gibt zurück, ob es sich bei der Kategorie um ein Vertiefungsfach handelt

### **enum CycleType**

**Beschreibung** Turnus eines Moduls

#### **Enum-Konstanten**

- **BOTH**  
Modul wird in jedem Semester angeboten
- **SUMMER\_TERM**  
Modul wird nur im Sommersemester angeboten
- **WINTER\_TERM**  
Modul wird nur im Wintersemester angeboten

#### **Methoden**

- public static CycleType **valueOf**(String name)
- public static CycleType **values()**

### **class Discipline**

**Beschreibung** Modelliert ein Studienfach

#### **Konstruktoren**

- public **Discipline()**

## Methoden

- public String **getDescription()**  
**Rückgabe** gibt die Fachrichtungsbeschreibung zurück
- public int **getDisciplineId()**  
**Rückgabe** gibt die eindeutige Studienfach-ID zurück

## class Module

**Beschreibung** Modelliert ein Modul.

## Konstruktoren

- public **Module()**

## Methoden

- public List<Category> **getCategories()**  
**Rückgabe** gibt die Kategorien, denen das Modul angehört, zurück
- public List<ModuleConstraint> **getConstraints()**  
**Rückgabe** gibt die Abhängigkeiten des Moduls zu anderen Modulen zurück
- public int **getCreditPoints()**  
**Rückgabe** gibt die ECTS-Zahl des Moduls zurück
- public CycleType **getCycleType()**  
**Rückgabe** gibt den Turnus des Moduls zurück
- public Discipline **getDiscipline()**  
**Rückgabe** gibt den Studiengang, dem das Modul angehört, zurück.  
Ein Modul ist immer eindeutig einem Studiengang zugeordnet. Wird ein Modul in der Realität für mehrere Studiengänge angeboten, so handelt es sich jeweils um unterschiedliche Module, denn in einem anderen Studiengang können Modulabhangigkeiten ggf. variieren.
- public String **getIdentifier()**  
**Rückgabe** gibt den eindeutigen Identifier-String zurück
- public ModuleDescription **getModuleDescription()**  
**Rückgabe** gibt die Modul-Beschreibung des Moduls zurück
- public int **getModuleId()**  
**Rückgabe** gibt die eindeutige Modul-ID zurück
- public String **getName()**  
**Rückgabe** gibt die Modulbezeichnung zurück
- public boolean **isCompulsory()**  
**Rückgabe** gibt zurück, ob es sich um ein Pflichtmodul handelt

## **class ModuleDescription**

**Beschreibung** Modelliert eine Modulbeschreibung.  
Eine Modulbeschreibung kann mehreren Modulen zugeordnet sein.

### **Konstruktoren**

- public **ModuleDescription()**

### **Methoden**

- public int **getDescriptionId()**  
**Rückgabe** gibt die eindeutige Beschreibungs-ID zurück
- public String **getDescriptionText()**  
**Rückgabe** gibt den Beschreibungstext zurück
- public String **getLecturer()**  
**Rückgabe** gibt den Dozenten zurück
- public ModuleType **getModuleType()**  
**Rückgabe** gibt den Modul-Typ zurück

## **enum ModuleOrientation**

**Beschreibung** Beschreibt die Richtung eines Moduleconstraints bei der Übergabe an die verify(first, second, orientation)-Methode von ModuleConstraintType.

### **Enum-Konstanten**

- **LEFT\_TO\_RIGHT**

Für ein gegebenes Tupel an Parametern (first, second) für die verify-Funktion mit gegebenen Constraint-Modulen modul1 und modul2 gilt:

first := modul1  
second := modul2

- **RIGHT\_TO\_LEFT**

Für ein gegebenes Tupel an Parametern (first, second) für die verify-Funktion mit gegebenen Constraint-Modulen modul1 und modul2 gilt:

second := modul1  
first := modul2

### **Methoden**

- public static ModuleOrientation **valueOf(String name)**
- public static ModuleOrientation **values()**

## **class ModuleType**

**Beschreibung** Modelliert einen Modultyp, wie beispielsweise Vorlesung, Seminar, etc.

### **Konstruktoren**

- public **ModuleType()**

### **Methoden**

- public String **getName()**  
**Rückgabe** gibt den Namen des Typs zurück
- public int **getTypeId()**  
**Rückgabe** gibt die eindeutige Typ-ID zurück

## **Package server.model.moduledata.constraint**

### **class ModuleConstraint**

**Beschreibung** Diese Klasse modelliert eine Abhängigkeit zwischen zwei Modulen.

### **Konstruktoren**

- public **ModuleConstraint()**

### **Methoden**

- public ModuleConstraintType **getConstraintType()**  
**Rückgabe** gibt den Typ der Abhängigkeit zurück

#### **Siehe auch:**

- ModuleConstraintType
- public Module **getFirstModule()**  
**Rückgabe** gibt das erste Modul der Abhängigkeitsrelation zurück
- public Module **getSecondModule()**  
**Rückgabe** gibt das zweite Modul der Abhängigkeitsrelation zurück

### **class ModuleConstraintType**

**Beschreibung** Modelliert eine Abhängigkeitstyp einer Modulabhängigkeit

## Konstruktoren

- public **ModuleConstraintType**()

## Methoden

- public String **getDescription**()

**Rückgabe** gibt die textuelle Beschreibung der Abhängigkeit zurück

- public String **getFormalDescription**()

**Rückgabe** gibt die Abhängigkeitsbeschreibung in Form eines logischen Ausdrucks zurück

- public abstract boolean **isValid**(ModuleEntry first, ModuleEntry second, ModuleOrientation orientation)

Überprüft, ob für zwei gegebene Moduleinträge die Abhängigkeit dieses Typs erfüllt ist.

### Parameter

**first** der erste Moduleintrag (Subjekt des aktuellen Verifikationsschritts)

**second** der zweite Moduleintrag (Zweites zusätzlich geladenes Modul des aktuellen Verifikationsschritts)

**orientation** Richtung, in die die Relation überprüft werden soll.

Jedes Constraint besitzt eine Richtung. Diese ist über die Eintragung der Module in Modul<sub>1</sub> und Modul<sub>2</sub> gegeben. Wenn nun gilt: first := Modul<sub>1</sub> UND second := Modul<sub>2</sub>, so ist die Richtung LEFT\_TO\_RIGHT; wenn aber gilt: first := Modul<sub>2</sub> und second := Modul<sub>1</sub>, so ist die Richtung RIGHT\_TO\_LEFT.

Dieses zusätzliche Attribut ist notwendig, da Informationen über zwei gerichtete Relationen übergeben werden müssen:

1. Welches Modul aktuell untersucht wird und welches „nur“ das zweite Modul des Constraints ist
2. Welches Modul im Constraint Modul<sub>1</sub> und welches Modul<sub>2</sub> ist  
(1) wird über die Anordnung der Parameter übergeben, (2) über den Parameter orientation.

**Rückgabe** Ergebnis der Überprüfung

- public void **setDescription**(String description)

### Parameter

**description** die textuelle Beschreibung

- public void **setFormalDescription**(String formalDescription)

### Parameter

**formalDescription** der logische Ausdruck

## **class OverlappingModuleConstraintType**

**Beschreibung** Modelliert eine zeitliche Abhangigkeit zwischen zwei Modulen:  
Die beiden Module konnen nicht im gleichen Semester belegt werden, da die Veranstaltung zur gleichen Zeit stattfindet.

### **Konuktoren**

- public **OverlappingModuleConstraintType()**

### **Methoden**

- public boolean **isValid**(ModuleEntry first, ModuleEntry second,  
ModuleOrientation orientation)

## **class PlanLinkModuleConstraintType**

**Beschreibung** Modelliert eine Zusammenhangsabhangigkeit zwischen Modulen:  
Es mussen immer beide Module im Plan enthalten sein – oder keines der beiden.

### **Konuktoren**

- public **PlanLinkModuleConstraintType()**

### **Methoden**

- public boolean **isValid**(ModuleEntry first, ModuleEntry second,  
ModuleOrientation orientation)

## **class PrerequisiteModuleConstraintType**

**Beschreibung** Modelliert eine Voraussetzungsabhangigkeit zwischen Modulen:  
Das erste Modul ist Voraussetzung fur das zweite Modul.

### **Konuktoren**

- public **PrerequisiteModuleConstraintType()**

### **Methoden**

- public boolean **isValid**(ModuleEntry first, ModuleEntry second,  
ModuleOrientation orientation)

## **class SemesterLinkModuleConstraintType**

**Beschreibung** Modelliert eine Zusammenhangsbeziehung zwischen zwei Modulen: Beide Module müssen im gleichen Semester belegt werden.

### **Konstruktoren**

- public **SemesterLinkModuleConstraintType()**

### **Methoden**

- public boolean **isValid**(ModuleEntry first, ModuleEntry second, ModuleOrientation orientation)

## **Package server.model.moduledata.dao**

### **class HibernateModuleDao**

**Beschreibung** Ein konkretes ModulDao, welches die Datenbankverbindung über Hibernate herstellt. Es kann nur auf Module, Kategorien und Vertiefungsfächer des im Konstruktur angegebenen Studiengangs zugreifen.

### **Konstruktoren**

- public **HibernateModuleDao**(Discipline discipline)  
Initialisiert ein neues DAO für den angegebenen Studiengang

#### **Parameter**

**discipline** der Studiengang

### **Methoden**

- public List<Category> **getCategories()**
- public List<Discipline> **getDisciplines()**
- public Module **getModuleById**(String id)
- public List<Module> **getModulesByFilter**(Filter filter)
- public List<Module> **getModulesByFilter**(Filter filter, int start, int end)
- public Module **getRandomModuleByFilter**(Filter filter)
- public List<Category> **getSubjects()**

## **class ModuleAttributeNames**

**Beschreibung** Kapselt die für die Filterarchitektur nötigen Module-Attribut-Namen als Stringkonstanten. Diese werden bei der Anwendung der Filter in den ModuleDao-Methoden für die eindeutige Identifizierung von Module-Eigenschaften benutzt.

### Siehe auch:

- Filter
- ModuleDao

### Attribute

- public static final String **CATEGORY**  
Stringkonstante, die die Kategorie-Eigenschaft eines Moduls repräsentiert.
- public static final String **CREDIT\_POINTS**  
Stringkonstante, die die ECTS-Eigenschaft eines Moduls repräsentiert.
- public static final String **CYCLE\_TYPE**  
Stringkonstante, die die Turnus-Eigenschaft eines Moduls repräsentiert.
- public static final String **DISCIPLINE**  
Stringkonstante, die die Fachrichtungs-Eigenschaft eines Moduls repräsentiert.
- public static final String **IS\_COMPULSORY**  
Stringkonstante, die die Wahl-/Pflicht-Veranstaltungs-Eigenschaft eines Moduls repräsentiert.
- public static final String **MODULE\_TYPE**  
Stringkonstante, die die Modultyp-Eigenschaft eines Moduls repräsentiert.
- public static final String **NAME**  
Stringkonstante, die die Modulnamens-Eigenschaft eines Moduls repräsentiert.

### Konstruktoren

- public **ModuleAttributeNames**()

## interface ModuleDao

**Beschreibung** Data-Access-Object zum Zugriff auf die Modul-Datenbank.

### Methoden

- public List<Category> **getCategories**()  
**Rückgabe** gibt eine Liste der verfügbaren Kategorien zurück
- public List<Discipline> **getDisciplines**()  
**Rückgabe** gibt eine Liste der verfügbaren Studiengänge zurück
- public Module **getModuleById**(String id)  
**Parameter**  
**id** der String-Identifier des zu suchenden Moduls  
**Rückgabe** das Modul mit dem entsprechenden Identifier, null wenn kein Modul gefunden
- public List<Module> **getModulesByFilter**(Filter filter)  
Sucht alle Module, die den angegebenen Filterkriterien entsprechen, und gibt diese zurück

**Parameter**

**filter** der Modulfilter

**Rückgabe** die Modulliste

- public List<Module> **getModulesByFilter**(Filter filter, int start, int end)

Sucht alle Module, die den angegebenen Filterkriterien entsprechen, und gibt die Einträge Nr. start bis end zurück.

**Parameter**

**filter** der Modulfilter

**start** Start-Index

**end** End-Index

**Rückgabe** die Modulliste

- public Module **getRandomModuleByFilter**(Filter filter)

Gibt ein zufälliges Modul, welches den angegebenen Filterkriterien entspricht, zurück

**Parameter**

**filter** der Modulfilter

**Rückgabe** das Modul

- public List<Category> **getSubjects()**

**Rückgabe** gibt eine Liste der verfügbaren Vertiefungsfächer zurück

**class ModuleDaoFactory**

**Beschreibung** Factory zur ModuleDao-Erzeugung

**Konstruktoren**

- public **ModuleDaoFactory()**

**Methoden**

- public static ModuleDao **getModuleDao**(Discipline discipline)

**Parameter**

**discipline** der Studiengang

**Rückgabe** liefert das für die verwendete Datenbankschnittstelle benötigte DAO zurück

Das DAO wird mit dem übergebenen Studiengang initialisiert.

## **Package server.model.userdata.authorization**

### **class AuthorizationContext**

**Beschreibung** Modelliert einen Authorisierungskontext.

Er enthält die benötigten Informationen für einen autorisierten Benutzer, wie in Kapitel 4.2 beschrieben

#### **Konstruktoren**

- public **AuthorizationContext()**

#### **Methoden**

- public String **getAccessToken()**  
**Rückgabe** gibt das Access-Token zurück
- public Date **getExpiryDate()**  
**Rückgabe** gibt das Verfallsdatum des Access-Tokens zurück
- public String **getRefreshToken()**  
**Rückgabe** gibt das Refresh-Token zurück
- public RESTClient **getRestClient()**  
**Rückgabe** gibt den zugehörigen REST-Client zurück
- public AuthorizationScope **getScope()**  
**Rückgabe** gibt die Berechtigung des Nutzers zurück
- public User **getUser()**  
**Rückgabe** gibt den Nutzer zurück
- public void **setAccessToken**(String accessToken)

#### **Parameter**

**accessToken** das Access-Token

- public void **setExpiryDate**(Date date)

#### **Parameter**

**date** das Verfallsdatum

- public void **setRefreshToken**(String refreshToken)

#### **Parameter**

**refreshToken** das Refresh-Token

- public void **setRestClient**(RESTClient client)

#### **Parameter**

**client** der REST-Client

- public void **setScope**(AuthorizationScope scope)

#### **Parameter**

**scope** die Berechtigung

- public void **setUser**(User user)

### Parameter

**user** der Nutzer

## enum AuthorizationScope

**Beschreibung** Berechtigungen, die ein Nutzer anfragen kann

### Enum-Konstanten

- **STUDENT**

Berechtigung Student. Kann alle Kernfunktionen nutzen.

### Methoden

- public static AuthorizationScope **valueOf**(String name)
- public static AuthorizationScope **values**()

## class RESTClient

**Beschreibung** Modelliert einen Klienten, der auf die REST-Schnittstelle zugreifen kann, wie in Kapitel 4.2 beschrieben.

### Konstruktoren

- public **RESTClient**()

### Methoden

- public String **getApiKey**()  
**Rückgabe** gibt die eindeutige ID des Klienten zurück
- public String **getApiSecret**()  
**Rückgabe** gibt die nur dem Klienten bekannte Kennung zurück
- public String **getOrigin**()  
**Rückgabe** gibt die Domain, von welcher aus der Client auf Ressourcen zugreifen kann, als regulären Ausdruck zurück.
- public URL **getRedirectUrl**()  
**Rückgabe** gibt die Weiterleitungs-URL zurück
- public List **getScopes**()  
**Rückgabe** gibt eine Liste aller Berechtigungen zurück, die vom Client angefragt werden können
- public void **setApiKey**(String apiKey)

### Parameter

- **apiKey** die ID des Klienten
- public void **setApiSecret**(String apiSecret)

### **Parameter**

**apiSecret** die Kennung

- public void **setOrigin**(String origin)

### **Parameter**

**origin** der reguläre Ausdruck

- public void **setRedirectUrl**(URL redirectUrl)

### **Parameter**

**redirectUrl** die Weiterleitungs-URL

## **Package server.model.usermodel.dao**

### **class AbstractSecurityProvider**

**Beschreibung** Diese Klasse ermöglicht die Authentifizierung, indem sie Methoden zur Access-Token-Generierung sowie zur Abfrage von Authentifizierungs-Kontexten bereitstellt.

#### **Konstruktoren**

- public **AbstractSecurityProvider**()

#### **Methoden**

- public abstract AuthorizationContext **generateAuthorizationContext**(User user)

Erzeugt zu einem Nutzer einen neuen Authentifizierungs-Kontext inklusive Access-Token und speichert diesen in der Datenbank.

Die Gültigkeitsdauer wird auf 4 Stunden gesetzt.

### **Parameter**

**user** der User

**Rückgabe** den generierten Authentifizierungs-Kontext

- public abstract AuthorizationContext **getAuthorizationContext**(String accessToken)

Liefert zu einem Access-Token den entsprechenden Authentifizierungs-Kontext.

Ist kein Kontext vorhanden, so wird null zurückgegeben.

### **Parameter**

**accessToken** der Access-Token

**Rückgabe** der Authentifizierungs-Kontext

- public static final AbstractSecurityProvider **getSecurityProviderImpl**()

**Rückgabe** gibt eine konkrete SecurityProvider-Implementierung zurück

## **class HibernatePlanDao**

**Beschreibung** Ein konkretes PlanDao, welches die Datenbankverbindung über Hibernate herstellt.

### **Konstruktoren**

- **HibernatePlanDao ()**

### **Methoden**

- public void **deletePlan**(Plan plan)
- public Plan **getPlanById**(String id)
- public void **updatePlan**(Plan plan)

## **class HibernateUserDao**

**Beschreibung** Ein konkretes UserDao, welches die Datenbankverbindung über Hibernate herstellt.

### **Konstruktoren**

- **HibernateUserDao ()**

### **Methoden**

- public void **deleteUser**(User user)
- public User **findUser**(User user)
- public void **updateUser**(User user)

## **interface PlanDao**

**Beschreibung** Data-Access-Object zum Zugriff auf Studienpläne aus der Datenbank

### **Methoden**

- public void **deletePlan**(Plan plan)  
Löscht den Plan aus der Datenbank.

#### **Parameter**

**plan** der Plan

- public Plan **getPlanById**(String id)  
Sucht einen Plan nach seinem String-Identifier.

#### **Parameter**

**id** der Identifier-String

**Rückgabe** den gefundenen Plan oder null, falls nichts gefunden

- public void **updatePlan**(Plan plan)  
Speichert alle Änderungen am Plan in der Datenbank bzw. legt ihn an, wenn noch nicht vorhanden.

#### **Parameter**

plan der Plan

## **class PlanDaoFactory**

**Beschreibung** Factory zur UserDao-Erzeugung

#### **Konstruktoren**

- public **PlanDaoFactory**()

#### **Methoden**

- public static PlanDao **getPlanDao**()

**Rückgabe** liefert das für die verwendete Datenbankschnittstelle benötigte DAO zurück

## **class SecurityProvider**

**Beschreibung** Konkrete Implementierung eines Security-Providers, der Hibernate verwendet.

#### **Konstruktoren**

- **SecurityProvider**()

#### **Methoden**

- public AuthorizationContext **generateAuthorizationContext**(User user)
- public AuthorizationContext **getAuthorizationContext**(String accessToken)

## **interface UserDao**

**Beschreibung** Data-Access-Object zum Zugriff auf Nutzer in der Datenbank

#### **Methoden**

- public void **deleteUser**(User user)  
Löscht den übergebenen Nutzer aus der Datenbank

**Parameter**

**user** der Nutzer

- public User **findUser**(User user)

**Rückgabe** Sucht nach dem entsprechenden Nutzer in der Datenbank.

So kann ein Nutzer über die ID oder seinen Nutzernamen gefunden werden.

**Parameter**

**user** der zu suchende Nutzer

**Rückgabe** der gefundene Nutzer

- public void **updateUser**(User user)

Speichert die Änderungen am übergebenen Nutzer in der Datenbank

Handelt es sich um einen neuen Nutzer, so wird dieser angelegt.

**Parameter**

**user** der Nutzer

**class UserDaoFactory**

**Beschreibung** Factory zur UserDao-Erzeugung

**Konstruktoren**

- public **UserDaoFactory**()

**Methoden**

- public static UserDao **getUserDao**()

**Rückgabe** Liefert das für die verwendete Datenbankschnittstelle benötigte DAO zurück

**Package server.model.usermodel****class ModuleEntry**

**Beschreibung** Modelliert einen Moduleintrag in einem Studienplan

**Konstruktoren**

- public **ModuleEntry**()

**Methoden**

- public Module **getModule**()

- **Rückgabe** gibt das Modul zurück
- public int **getSemester()**
- **Rückgabe** gibt die Nummer des Semesters zurück, dem der Eintrag zugeordnet wurde
- public void **setModule(Module module)**

**Parameter**

  - module** das Modul
- public void **setSemester(int semester)**

**Parameter**

  - semester** die Semesternummer

## **class ModulePreference**

**Beschreibung** Modelliert eine Modulpräferenz. Eine Modulpräferenz bezeichnet die Bewertung eines Moduls durch einen Nutzer.

### **Konstruktoren**

- public **ModulePreference()**

### **Methoden**

- public Module **getModule()**

**Rückgabe** gibt das Modul zurück, zu dem die Präferenz gehört
- public PreferenceType **getPreference()**

**Rückgabe** gibt den Typ der Präferenz zurück

#### **Siehe auch:**

- PreferenceType
- public void **setModule(Module module)**

**Parameter**

  - module** das Modul
- public void **setPreference(PreferenceType preferenceType)**

**Parameter**

  - preferenceType** der Präferenztyp

## **class Plan**

**Beschreibung** Modelliert einen Studienplan

## Konstruktoren

- public **Plan()**

## Methoden

- public int **getCreditPoints()**  
**Rückgabe** gibt die ECTS-Summe des Plans zurück
- public String **getIdentifier()**  
**Rückgabe** gibt den eindeutigen Plan-Identifier zurück
- public List<ModuleEntry> **getModuleEntries()**  
**Rückgabe** gibt alle Moduleinträge des Plans zurück
- public String **getName()**  
**Rückgabe** gibt den Namen des Plans zurück
- public int **getPlanId()**  
**Rückgabe** gibt die eindeutige Plan-ID zurück
- public PreferenceType **getPreferenceForModule**(Module module)  
Gibt für ein übergebenes Modul die Präferenz zurück.  
null, falls keine Präferenz vorhanden

### Parameter

**module** das Modul

### Rückgabe die Präferenz

- public ModulePreference **getPreferences()**  
**Rückgabe** gibt eine List der Modulpräferenzen zurück
- public User **getUser()**  
**Rückgabe** gibt den Eigentümer des Plans zurück
- public VerificationState **getVerificationState()**  
**Rückgabe** gibt den Verifizierungsstatus des Plans zurück
- public void **setCreditPoints**(int creditPoints)

### Parameter

**creditPoints** die ECTS-Summe

- public void **setIdentifier**(String identifier)

### Parameter

**identifier** der Plan-Identifier

- public void **setName**(String name)

### Parameter

**name** der Name

- public void **setPlanId**(String planId)

### Parameter

**planId** die Plan-ID

- public void **setUser**(User user)

### **Parameter**

**user** der Eigentümer

- public void **setVerificationState**(VerificationState verificationState)

### **Parameter**

**verificationState** der Verifizierungsstatus

## **enum PreferenceType**

**Beschreibung** Modelliert den Typ einer Modulpräferenz

### **Enum-Konstanten**

- **NEGATIVE**  
das Modul wurde negativ bewertet
- **POSITIVE**  
das Modul wurde positiv bewertet

### **Methoden**

- public static PreferenceType **valueOf**(String name)
- public static PreferenceType **values**()

## **class Semester**

**Beschreibung** Modelliert ein Semester

### **Konuktoren**

- public **Semester**()

### **Methoden**

- public int **getDistanceToCurrentSemester**()  
Berechnet die Anzahl an Semester, die seit diesem Semester vergangen sind (inkl. dem aktuellen)  
**Rückgabe** die Semesterzahl
- public SemesterType **getSemesterType**()  
**Rückgabe** gibt den Typ des Semester zurück

### **Siehe auch:**

- SemesterType
- public int **getYear**()  
**Rückgabe** gibt das Jahr zurück in dem das Semester begonnen hat
- public void **setSemesterType**(SemesterType semesterType)

### **Parameter**

- `semesterType` der Semestertyp
- public void `setYear`(int year)

### **Parameter**

`year` das Jahr in dem das Semester begonnen hat

## **enum SemesterType**

**Beschreibung** Modelliert Semester-Typen

### **Enum-Konstanten**

- `SUMMER_TERM`  
Sommersemester
- `WINTER_TERM`  
Wintersemester

### **Methoden**

- public static SemesterType `valueOf`(String name)
- public static SemesterType `values`()

## **class User**

**Beschreibung** Modelliert einen Nutzer.

### **Konstruktoren**

- public `User`()

### **Methoden**

- public Discipline `getDiscipline`()  
**Rückgabe** gibt den Studiengang zurück
- public List<ModuleEntry> `getPassedModules`()  
**Rückgabe** gibt eine Liste von Modul-Einträgen der bestandenen Module zurück
- public List<Plan> `getPlans`()  
**Rückgabe** gibt eine Liste der Studienpläne des Nutzers zurück
- public Semester `getStudyStart`()  
**Rückgabe** gibt das Semester des Studienstarts zurück
- public int `getUserId`()  
**Rückgabe** gibt die eindeutige ID des Nutzers zurück
- public String `getUserName`()

**Rückgabe** gibt den eindeutigen Nutzernamen zurück

- public void **setDiscipline**(Discipline discipline)

**Parameter**

discipline der Studiengang

- public void **setStudyStart**(Semester semester)

**Parameter**

semester das Semester des Studienstarts

- public void **setUserId**(int userId)

**Parameter**

userId Wert, auf den die ID gesetzt wird

- public void **setUserName**(String userName)

**Parameter**

userName der Nutzernname

## enum VerificationState

**Beschreibung** Modelliert den Verifikationsstatus eines Studienplans

### Enum-Konstanten

- **INVALID**  
der Plan enthält Fehler, d.h. Constraints sind verletzt
- **NOT\_VERIFIED**  
der Plan wurde noch nicht verifiziert
- **VALID**  
der Plan ist gültig

### Methoden

- public static VerificationState **valueOf**(String name)
- public static VerificationState **values**()

## Package server.pluginmanager

### class GenerationManager

**Beschreibung** Verwaltet den Zugriff auf das Generierungs-PlugIn. Das Generierungs-PlugIn umfasst sowohl die Generierer-Schnittstelle als auch die Zielfunktionen-Schnittstelle. Beide Schnittstellen werden mittels dieser Klasse adaptiert.

### Konstruktoren

- public **GenerationManager**()  
Erstellt einen GenerationManager.

## Methoden

- public double **evaluate**(Plan plan)

Diese Methode ruft die evaluate-Methode der PartialObjectiveFunction auf.

### Parameter

**plan** der zu bewertende Plan.

**Rückgabe** Wert zwischen 0 und 1 als Planbewertung.

- public Plan **generate**(PartialObjectiveFunction objectiveFunction, Plan currentPlan, ModuleDao moduleDAO)

Diese Methode ruft die generate-Methode des Generator auf.

### Parameter

**objectiveFunction** Die Zielfunktion, anhand der optimiert werden soll

**currentPlan** der bereits bestehende Plan

**moduleDAO** das Modul-DAO

**Rückgabe** ein vollständiger, korrekter und optimierter Studienplan.

- public Generator **getGenerator**()

Gibt den Generator zurück.

**Rückgabe** der Generator

- public Collection<PartialObjectiveFunction> **getObjectiveFunction**()

Gibt die Liste der Zielfunktionen zurück.

**Rückgabe** die Liste der Zielfunktionen

## class VerificationManager

**Beschreibung** Verwaltet den Zugriff auf das Verifizierungs-PlugIn, das die Verifizierer-Schnittstelle enthält. Diese Schnittstelle wird von dem VerificationManager adaptiert.

### Attribute

- private Verifier **Verifier**

Der Verifizierer.

### Siehe auch:

– server.verification.Verifier

### Konstruktoren

- public **VerificationManager**()

Erstellt einen VerificationManager.

## Methoden

- public Verifier **getVerifier**()

Gibt den Verifizierer zurück.

#### **Rückgabe** der Verifizierer

- public VerificationResult **verify**(Plan plan)  
Diese Methode ruft die verify-Methode des Verifier auf.

#### **Parameter**

**plan** Ein zu verifizierender Studienplan wird übergeben.

**Rückgabe** Ein VerificationResult wird als Ergebnis der Verifizierung zurückgegeben.

## **Package server.rest.authorization.endpoint**

### **class AuthorizationCodeGrant**

**Beschreibung** Diese Klasse repräsentiert einen Authorization-Code-Grant (Siehe [1, Kap. 1.3.1]). AuthorizationCodeGrant wird in der ersten Version des Systems nicht benötigt, ist aber für spätere Erweiterungen vorgesehen. Bei dem Versuch, eine Authentifizierung mittels Authorization-Code-Grant durchzuführen, wird ein Fehler zurückgegeben.

#### **Konuktoren**

- public **AuthorizationCodeGrant**()

#### **Methoden**

- public void **getLogin**(String clientId, String scope, String state)
- public void **postToken**(MultivaluedMap<StringString> params)

### **class AuthResource**

**Beschreibung** Diese Klasse repräsentiert die Authentifizierung-Ressource. Der Resource-Owner wird zurück zu dieser Ressource weitergeleitet, nachdem er den Zugriff auf die Anwendung erhalten hat.

#### **Konuktoren**

- public **AuthResource**()  
Erstellt eine Authentifizierungs-Ressource.

#### **Methoden**

- public GrantType **getGrantType**()  
Gibt den GrantType zurück.

**Rückgabe** den GrantType

- public void **getLogin**(String clientID, String scope, String state)  
GET-Anfrage: Gibt den Authorization Endpoint ([1], Kapitel 3.1) zurück.

**Parameter**

**clientID** den api\_key des Klienten.

**scope** in den ersten Versionen des Systems immer „student“.

**state** ein Schlüssel, der vom REST-Webservice in der Antwort mitgesendet wird.

**Siehe auch:**

– Kapitel 4.2 Tabelle 5.

- public void **postToken**(MultivaluedMap<StringString> params)  
POST-Anfrage: Setzt den Token Endpoint ([1], Kapitel 3.1).

**Parameter**

**params** eine mehrwertige Zuordnung

## interface GrantType

**Beschreibung** Diese Schnittstelle repräsentiert eine Fabrik zur Erstellung von Typen von Authorization Grant ([1], Kapitel 1.3)

**Methoden**

- public void **getLogin**(String clientId, String scope, String state)  
GET-Anfrage: Gibt den Authorization Endpoint (siehe [1], Kapitel 1.3) zurück.

**Parameter**

**clientId** den api\_key des Klienten.

**scope** in den ersten Versionen des Systems immer „student“.

**state** ein Schlüssel, der vom REST-Webservice in der Antwort mitgesendet wird.

**Siehe auch:**

– Kapitel 4.2 Tabelle 5.

- public void **postToken**(MultivaluedMap<StringString> params)  
POST-Anfrage: Setzt den Token Endpoint (siehe [1], Kapitel 3.1).

**Parameter**

**params** eine mehrwertige Zuordnung

## **class ImplicitGrantType**

**Beschreibung** Diese Klasse repräsentiert einen Implicit-Grant-Type ([1], Kapitel 1.3.2).

### **Konstruktoren**

- `public ImplicitGrantType()`  
Erstellt einen ImplicitGrantType.

### **Methoden**

- `public void getLogin(String clientId, String scope, String state)`
- `public void postToken(MultivaluedMap<StringString> params)`

## **class PasswordCredentialsGrant**

**Beschreibung** Diese Klasse repräsentiert einen Password-Credentials-Grant (siehe [1], Kapitel 1.3.3). PasswordCredentialsGrant wird in der ersten Version des Systems nicht benötigt, ist aber für spätere Erweiterungen vorgesehen. Bei dem Versuch, eine Authentifizierung mittels Password-Credentials-Grant durchzuführen, wird eine Fehlermeldung zurückgegeben.

### **Konstruktoren**

- `public PasswordCredentialsGrant()`  
Erstellt einen PasswordCredentialsGrant.

### **Methoden**

- `public void getLogin(String clientId, String scope, String state)`
- `public void postToken(MultivaluedMap<StringString> params)`

## **class RefreshGrant**

**Beschreibung** Diese Klasse repräsentiert einen Refresh-Grant: Beim Ablaufen des Access-Tokens schickt dieser GrantType ein Refresh-Token (siehe [1], Kapitel 1.5) als Antwort an den Klient.

### **Konstruktoren**

- `public RefreshGrant()`  
Erstellt einen RefreshGrant.

### **Methoden**

- `public void getLogin(String clientId, String scope, String state)`
- `public void postToken(MultivaluedMap<StringString> params)`

## **class UnknownGrant**

**Beschreibung** Diese Klasse repräsentiert einen unbekannten GrantType. Bei Instantiierung dieser Klasse wird eine Fehlermeldung zurückgegeben, da der GrantType ungültig ist.

### **Konuktoren**

- public **UnknownGrant()**  
Löst eine Ausnahme aus.

### **Methoden**

- public void **getLogin**(String clientId, String scope, String state)
- public void **postToken**(MultivaluedMap<StringString> params)

## **Package server.rest**

### **class AuthorizationRequestFilter**

**Beschreibung** Klasse für das Filtern von Authentifizierungs-Anfragen.

### **Konuktoren**

- public **AuthorizationRequestFilter()**  
Erstellt einen Filter für Authentifizierung-Anfragen.

### **Methoden**

- public void **filter()**

### **class FilterResource**

**Beschreibung** Diese Klasse repräsentiert die Filter-Ressource.

### **Konuktoren**

- public **FilterResource()**  
Erstellt eine Filter-Ressource.

### **Methoden**

- public List<JSONObject> **getAllFilters()**  
GET-Anfrage: Gibt eine Liste aller vorhandenen Filter zurück.  
**Rückgabe** eine Liste von Filtern.

### **class GetParameters**

**Beschreibung** Platzhalter für mehrere GET-Anfrage-Parameter.

## **class JSONObject**

**Beschreibung** Platzhalter für diverse JSON-Objekte.

## **class MainApplication**

**Beschreibung** Hilfsklasse, um Ressourcen-Klassen festzulegen.

### **Konstruktoren**

- public **MainApplication()**

## **class ModuleResource**

**Beschreibung** Diese Klasse repräsentiert die Modul-Ressource.

### **Konstruktoren**

- public **ModuleResource()**  
Erstellt eine Module-Ressource.

### **Methoden**

- public List<JSONObject> **getAllDisciplines()**  
GET-Anfrage: Gibt eine Liste der JSON-Repräsentationen aller Fachrichtungen zurück.  
**Rückgabe** eine Liste der JSON-Repräsentationen aller Fachrichtungen.
- public List<JSONObject> **getAllSubjects()**  
GET-Anfrage: Gibt eine Liste der JSON-Repräsentationen aller Vertiefungsfächer zurück.  
**Rückgabe** eine Liste der JSON-Repräsentationen aller Vertiefungsfächer.
- public JSONObject **getModule(String moduleID)**  
GET-Anfrage: Gibt eine JSON-Repräsentation des Moduls mit der gegebenen ID zurück.

### **Parameter**

**moduleID** ID des gewünschten Moduls.

**Rückgabe** eine JSON-Repräsentation des Moduls.

- public List<JSONObject> **getModules(GetParameters jsonFilter)**  
GET-Anfrage: Gibt eine Liste der JSON-Repräsentationen von Modulen, die Kriterien gegebener Filter entsprechen, zurück.

### **Parameter**

**jsonFilter** die benutzten Filter als GET-Parameter (siehe Modules-Parameter in Kap. 4.5).

**Rückgabe** eine Liste der JSON-Repräsentationen der gefilterten Module.

## **class ObjectiveFunctionResource**

**Beschreibung** Diese Klasse repräsentiert die Zielfunktion-Ressource.

### **Konstruktoren**

- `public ObjectiveFunctionResource()`  
Erstellt eine Zielfunktion-Ressource.

### **Methoden**

- `public List<JSONObject> getAllObjectiveFunctions()`  
GET-Anfrage: Gibt eine Liste mit allen vorhandenen Zielfunktionen als JSON-Objekte zurück.

**Rückgabe** Liste mit allen vorhandenen Zielfunktionen als JSON-Objekte.

- `public GenerationManager getGenerationManager()`  
Gibt den GenerationManager zurück.

**Rückgabe** der GenerationManager

## **class PlanConverterResource**

**Beschreibung** Diese Klasse repräsentiert den PDF-Export für Pläne.

### **Konstruktoren**

- `public PlanConverterResource()`  
Erstellt eine PlanConverterResource.

### **Methoden**

- `public PDF convertplanToPDF(String planID, String accessToken)`  
GET-Anfrage: Gibt einen PDF-Druck des Plans mit der gegebenen ID zurück.

#### **Parameter**

**planID** ID des zu konvertierenden Plans.

**accessToken** Das in Kap. 4.2 spezifizierte Access-Token.

**Rückgabe** der Plan als PDF exportiert.

## **class PlanGeneratorResource**

**Beschreibung** Diese Klasse repräsentiert die Plan-Generierer-Ressource.

## Konstruktoren

- `public PlanGeneratorResource()`  
Erstellt eine PlanGeneratorResource.

## Methoden

- `public JSONObject generatePlan(String planID, GetParameters jsonSettings)`  
GET-Anfrage: Gibt einen auf Basis des Plans mit der gegebenen ID generierten Plan als JSON-Objekt zurück.

### Parameter

`planID` ID des Basis-Plans.

`jsonSettings` die gesetzten Einstellungen des Plans als GET-Parameter.

**Rückgabe** den generierten Plan als JSON-Objekt.

- `public GenerationManager getGenerationManager()`  
Gibt den GenerationManager zurück.

**Rückgabe** der GenerationManager

## class PlanModulesResource

**Beschreibung** Diese Klasse repräsentiert die Plan-Modul-Ressource.

## Konstruktoren

- `public PlanModulesResource()`  
Erstellt eine PlanModulesResource.

## Methoden

- `public Filter getFilterFromRequest(MultivaluedMap<String, String> params)`  
Erzeugt aus den in params gegebenen GET-Parametern einen Filter, der alle übergebenen Kriterien erfüllt.

### Parameter

`params` Die Anfrage-Parameter, in welchen mehrere Filter sowie deren Einstellungen gegeben sind.

**Rückgabe** den erzeugten Filter.

- `public JSONObject getModule(String planID, String moduleID)`  
GET-Anfrage: Gibt eine JSON-Repräsentation des Moduls moduleID aus Sicht des Plans planID zurück.

### Parameter

`planID` ID des gerade bearbeiteten Plans.

`moduleID` ID des geforderten Moduls.

**Rückgabe** eine JSON-Repräsentation des Moduls als JSON-Objekt.

- public `JSONObject getModules(String plan_id, GetParameters jsonFilter)`  
GET-Anfrage: Gibt die Liste der JSON-Repräsentationen der Module zurück, die den Kriterien der gegebenen Filter entsprechen. Die Module sind planspezifisch bezüglich des Plans planID.

**Parameter**

- `planID` ID des gerade bearbeiteten Plans.
- `jsonFilter` die benutzten Filter als GET-Parameter.

**Rückgabe** eine Liste der JSON-Repräsentationen von Modulen.

- public `public JSONObject putModuleSemester(String planID, String moduleID, GetParameters jsonPutModule)`  
PUT Anfrage: Fügt das Modul als JSON-Objekt zum Plan mit der gegebenen ModulID bzw. PlanID hinzu.

**Parameter**

- `planID` ID des zu bearbeitenden Plans.
- `moduleID` ID des hinzuzufügenden Moduls.
- `jsonPutModule` das Modul als GET-Parameter.

**Rückgabe** JSON-Repräsentation des Moduls als JSON-Objekt.

- public void `removeModuleSemester(String planID, String moduleID)`  
DELETE-Anfrage: entfernt das Modul von dem Plan mit den gegebenen ModulID bzw. PlanID.

**Parameter**

- `planID` ID des zu bearbeitenden Plans.
- `moduleID` ID des zu entfernenden Moduls.

- public `public JSONObject setModulePreference(String planID, String moduleID, JSONObject jsonModulePreference)`  
PUT-Anfrage: Setzt eine Bewertung für das Modul mit der gegebenen ID, das zum Plan planID gehört.

**Parameter**

- `planID` ID des zu bearbeitenden Plans.
- `moduleID` ID des zu bewertenden Moduls.
- `jsonModulePreference` die zu setzende Bewertung des Moduls als JSON-Objekt.

**Rückgabe** das bewertete Modul als JSON-Objekt.

## **class PlansResource**

**Beschreibung** Diese Klasse repräsentiert die Pläne-Ressource.

## Konstruktoren

- `public PlansResource()`  
Erstellt eine PlansResource.

## Methoden

- `public JSONObject createPlan()`  
POST-Anfrage: Erstellt einen neuen Studienplan.  
**Rückgabe** der erstellte Plan als JSON-Objekt.
- `public void deletePlan(String planID)`  
DELETE-Anfrage: Löscht den Plan mit der gegebenen ID.

### Parameter

`planID` ID des zu löschenen Plans.

- `public JSONObject duplicatePlan(String planID)`  
POST-Anfrage: Dupliziert den Plan mit der gegebenen ID

### Parameter

`planID` ID des zu duplizierenden Plans.

### Rückgabe

Plan als JSON-Objekt.

- `public JSONObject editPlan(String planID, GetParameters jsonPlan)`  
PATCH-Anfrage: Bearbeitet den Plan mit der gegebenen ID.

### Parameter

`planID` ID des zu bearbeitenden Plans.

`jsonPlan` der Plan als GET-Parameter.

### Rückgabe

JSON-Objekt des bearbeiteten Plans.

- `public JSONObject getPlan(String plan_id)`  
GET-Anfrage: Gibt den Plan mit der gegebenen ID zurück.

### Parameter

`planID` ID des angefragten Plans.

### Rückgabe

der Plan als JSON-Objekt.

- `public List<JSONObject> getPlans()`  
GET-Anfrage: Gibt eine Liste aller vorhandenen Studienpläne zurück.

### Rückgabe

eine Liste aller vorhandenen Studienpläne als JSON-Objekte.

- `public JSONObject replacePlan(String planID, JSONObject jsonPlan)`  
PUT-Anfrage: Ersetzt den Plan mit der gegebenen ID mit dem übergebenen Plan .

### Parameter

`planID` ID des zu überschreibenden Plans.

`jsonPlan` der zu speichernde Plan als JSON-Objekt.

### Rückgabe

der gespeicherte Plan.

## **class PlanVerifierResource**

**Beschreibung** Diese Klasse repräsentiert die Planverifizierer-Ressource.

### **Konstruktoren**

- **public PlanVerifierResource()**  
Erstellt eine Planverifizierer-Ressource.

### **Methoden**

- **public VerificationManager getVerificationManager()**  
Gibt den Verifizierungsmanager zurück.

**Rückgabe** der VerificationManager

- **public JSONObject verifyPlan(String planID)**  
GET-Anfrage: Verifiziert den Plan mit den gegebenen ID, gibt den verifizierten Plan mitsamt Verifizierungsergebnissen zurück und speichert den Verifikationsstatus in der Datenbank.

#### **Parameter**

**planID** ID des Plans.

**Rückgabe** den verifizierten Plan als JSON-Objekt.

## **class StudentResource**

**Beschreibung** Diese Klasse repräsentiert die Student-Resource.

### **Konstruktoren**

- **public StudentResource()**  
Erstellt eine Student-Ressource.

### **Methoden**

- **public void deleteStudent()**  
DELETE-Anfrage: Löscht den Studenten.

- **public JSONObject getInformation()**  
GET-Anfrage: Gibt die Studenteninformationen zurück.

**Rückgabe** die Studenteninformationen als JSON-Objekt.

- **public JSONObject replaceInformation(JSONObject jsonStudentInformation)**  
PUT-Anfrage: Ersetzt Informationen über einen Student und löscht die Verifikationsinformationen.

#### **Parameter**

**jsonStudentInformation** die Informationen über den Student als JSON-Objekt.

**Rückgabe** Student mit neuen Informationen als JSON-Objekt.

## **Package server.verification**

### **class VerificationResult**

**Beschreibung** Die Klasse VerificationResult ist das Ergebnis einer Verifizierung.

#### **Konstruktoren**

- public **VerificationResult()**

#### **Methoden**

- public Collection<ModuleConstraint> **getViolations()**  
Gibt die verletzten Modul-Constraints zurück.  
**Rückgabe** die verletzten Modul-Constraints
- public boolean **isCorrect()**  
isCorrect prüft anhand von violations, ob der Studienplan erfolgreich verifiziert wurde.  
**Rückgabe** zurückgegeben wird false, falls der Studienplan fehlerhaft ist und true, wenn er zu einem erfolgreichen Studienabschluss führt.

## **interface Verifier**

**Beschreibung** Das Interface Verifier bietet die allgemeine Struktur eines Verifizierers.

#### **Methoden**

- public VerificationResult **verify(Plan plan)**  
Die Methode verify verifiziert einen übergebenen Studienplan. Das heißt, sie überprüft anhand der gegebenen System-Constraints, ob der Plan zu einem erfolgreichen Studienabschluss führen kann.

#### **Parameter**

**plan** Ein zu verifizierender Studienplan wird übergeben.

**Rückgabe** Ein VerificationResult wird als Ergebnis der Verifizierung zurückgegeben.

## **Package server.verification.standard**

### **class StandardVerifier**

**Beschreibung** Der StandardVerifier implementiert den Verifier. Er ist eine konkret verifizierende Klasse.

## **Konstruktoren**

- public **StandardVerifier()**

## **Methoden**

- public VerificationResult **verify(Plan plan)**

## 8. Ablaufbeschreibungen

### 8.1. Generierungs-Algorithmus

#### 8.1.1. Einstieg

Wir nehmen zunächst an, dass eine Menge an Modulen  $M$  gegeben ist, welche von der Form  $(id_{modul}, ects, constraints)$  sind. Hierbei besteht  $constraints$  aus einer Menge an Constraints der Form  $(id_{modul1}, id_{modul2}, typ)$  mit  $typ \in \{prerequisite, both, symmetry, antisymmetry\}$ . Constraints der Form  $(id_{modul1}, id_{modul2}, both)$  und  $(id_{modul2}, id_{modul1}, both)$  gelten im Folgenden als äquivalent.

Wir sehen  $\mathcal{M}$  als die Menge aller vorhandenen Module an. Wir beschreiben einen *Studienplan* als eine Menge  $P$  von Tupeln der Form  $(num_{semester}, modul)$ , wobei  $num_{semester} \in \mathbb{N} \cup \{\infty\}$ <sup>1</sup> und  $modul$  wie oben beschrieben. Eine *Familie* beschreibt eine Menge von *Studienplänen*.

Weiter existiert über der Menge aller Studienpläne  $\Psi$  eine Zielfunktion  $\sigma : \Psi \rightarrow [0, 1]$ .

#### 8.1.2. Evolutionärer Algorithmus

Der Algorithmus erhält die oben beschriebene Menge  $M$  und generiert mit Hilfe von 8.1.3 eine *Familie*  $F$  mit  $|F| = O \in \mathbb{N}$  *Studienplänen*.

Anschließend wählt er ein beliebiges  $p \in \{q \in F \mid \sigma(q) = \max\{\sigma(t) \mid t \in F\}\}$ . Mittels 8.1.4 wird auf Basis von  $p$  eine neue Familie  $F'$  mit  $|F'| = O$  erstellt, aus welcher erneut ein  $p' \in \{q \in F' \mid \sigma(q) = \max\{\sigma(t) \mid t \in F'\}\}$  gewählt wird. Der letzte Schritt wird mit  $p := p'$  dann  $R \in \mathbb{N}$  mal wiederholt. Anschließend wird das letzte generierte  $p'$  als generierter Studienplan ausgegeben.

#### 8.1.3. Zufällige Generierung

Zunächst generieren wir auf Basis der übergebenen Module  $m \in M$  einen Graphen  $G = (V, E)$  mit

$$V := M$$

$$V' := V \cup \{m \in \mathcal{M} \mid \exists n \in V : \text{es ex. ein Constraint der Form } (id_m, id_n, prerequisite), \\ (id_m, id_n, both) \text{ oder } (id_m, id_n, symmetry)\},$$

---

<sup>1</sup> $\infty$  steht hierbei für ewige Studenten, welche Module schieben wollen.

$$E := \{(m, n) \in V' \times V' \mid \text{es ex. ein Constraint der Form } (id_m, id_n, \text{prerequisite}) \\ \text{oder } (id_m, id_n, both)\}.$$

Der Prozess zur Generierung von  $V'$  wird solange weitergeführt, bis alle Abhängigkeiten enthalten sind. Bei *both* ist zu beachten, dass  $m$  oder  $m$  und  $n$  bestandene Module sein können, nicht jedoch nur  $n$ .

$$\sum_{m \in V} m.ects \geq 180 \quad (1)$$

Zunächst definieren wir die Menge  $Z = \emptyset$  aller zufällig hinzugefügten Module. Ist die Ungleichung 1 erfüllt, so kann mittels 8.1.5 der Plan fertiggestellt werden. Ist die Ungleichung nicht erfüllt, wird für ein zufälliges Modul  $m$  des gewählten Vertiefungs-Fachs gesetzt:

$$\begin{aligned} Z' &:= Z \cup \{m\} \\ V' &:= V' \cup \{m\} \\ V' &:= V' \cup \{n \in \mathcal{M} \mid \exists l \in V' : \text{es ex. ein Constraint der Form } (id_l, id_n, \text{prerequisite}), \\ &\quad (id_l, id_n, both) \text{ oder } (id_l, id_n, symmetry)\} \\ E' &:= E \cup \{(n, m) \in V' \times V' \mid \text{es ex. ein Constraint der Form } (id_n, id_m, \text{prerequisite}), \\ &\quad (id_n, id_m, both)\} \end{aligned}$$

Auch hier wird der Prozess zur Erweiterung von  $V'$  iterativ vorgenommen, bis alle Constraint-Module in  $V'$  aufgenommen wurden. Anschließend wird erneut Ungleichung (1) überprüft und wie oben beschrieben vorgegangen.

#### 8.1.4. Zufällige Modifizierung

Für einen beliebigen wie in 8.1.3 als Graph dargestellten Plan, wähle man zufällig ein Modul  $m \in Z$  und lösche dieses samt seiner nur für dieses Modul  $m$  benötigten Abhängigkeiten. Diesen Schritt führe man  $l \in \mathbb{N}$  mal aus.

Anschließend füge man – wie in 8.1.3 beschrieben – erneut Module hinzu, bis Ungleichung (1) wieder erfüllt wird.

#### 8.1.5. Kontraktion der topologischen Sortierung

Zunächst wird der Graph mittels Tiefensuche topologisch sortiert und anschließend mit unten beschriebenem Algorithmus parallelisiert. Hierbei werden die bereits gesetzten Module zunächst in die vom Nutzer vorgegebenen Semester gesetzt.

Sei also  $G = (V, E)$  ein DAG und  $s : \{1, \dots, |V|\} \rightarrow V$  seine topologische Sortierung,  $c : V \rightarrow \mathbb{Q}$  die Gewichtungsfunktion. Sei weiter  $P = V \times V$  eine symmetrische, transitive Relation von Knoten, welche nicht parallel gesetzt werden dürfen. Sei weiter  $k \in \mathbb{N}$  das Maximum pro parallelem Schritt. Weiter sei  $p : V \times \mathbb{N} \rightarrow \{\text{true}, \text{false}\}$  eine Funktion,

die angibt, ob ein gegebener Knoten in ein gegebenes Semester gesetzt werden darf (Winter- und Sommersemester-Regelung sowie (Anti-)Symmetrie-Constraints).

Das Verfahren wird in Algorithmus 1 mittels Pseudocode beschrieben.

---

#### **Algorithmus 1 : Kontraktion der topologischen Sortierung**

---

```
// Initialisiere:
bucketAllocation : array [1...|V|] of \mathbb{N}
bucketSum : array [1...|V|] of \mathbb{N}
minPos := {1, ..., 1} : array [1...|V|] of \mathbb{N}

// Iteriere über alle $v \in V$ in topologischer Sortierung
for $i := 1$ to $|V|$ do
 $v := s(i)$
 set := false
 for $j := minPos[i]$ to $|V|$ do
 if $c(v) + bucketSum[j] \leq k$
 and $\forall (v, u) \in P : bucketAllocation[s^{-1}(u)] \neq j$
 and $p(v, j)$
 then
 bucketAllocation[i] := j // setze v in Bucket j
 bucketSum[j] += c(v)
 foreach $(v, u) \in E$ do
 minPos[s-1(u)] := max{j + 1, minPos[i]}
 set := true
 break
 if not set then
 throw Exception // Knoten zu groß
```

---

## 8.2. Abläufe

In den folgenden Sequenzdiagrammen werden beispielhaft vereinfachte Umsetzungen mehrerer Anwendungsfälle gezeigt. Auf die Darstellung von Parameterübergaben wird hier verzichtet.

Die Abbildungen 28 bis 31 beschreiben den Registrierungsablauf, der Teil der Umsetzung des Anwendungsfalls „A10: Erstanmeldung“ ist.

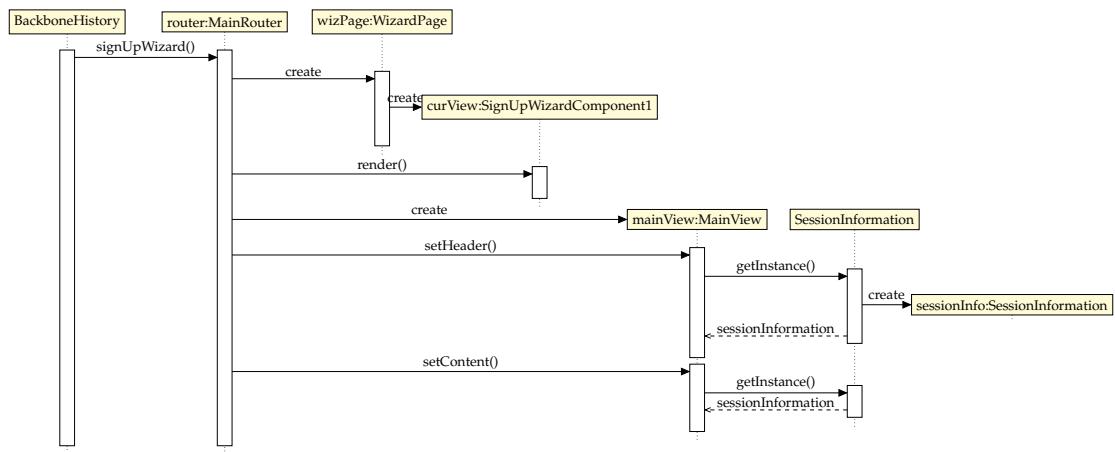


Abbildung 28: 1. Registrierungsseite anzeigen.

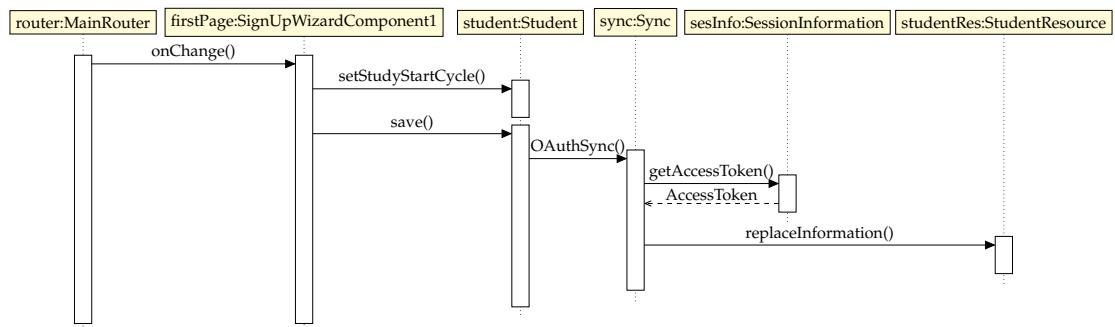


Abbildung 29: Die verarbeiteten Informationen speichern(Clientseitig)

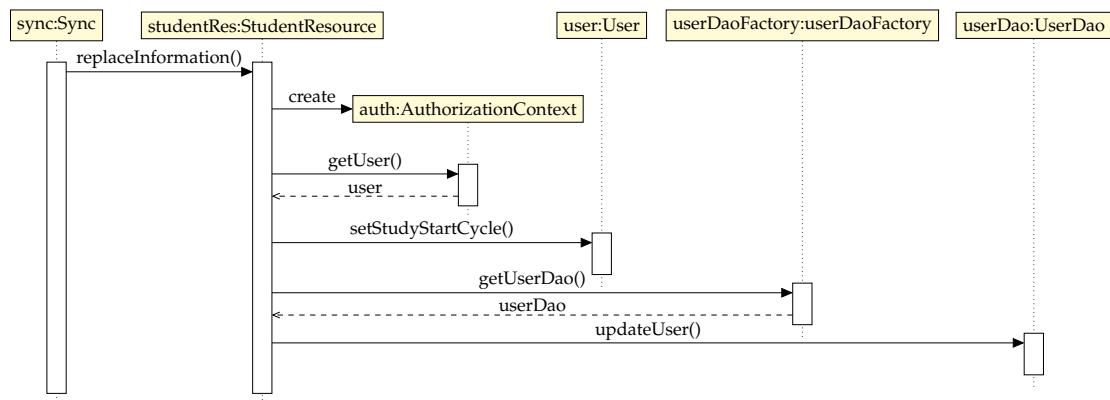


Abbildung 30: Die verarbeiteten Informationen speichern(Serverseitig)

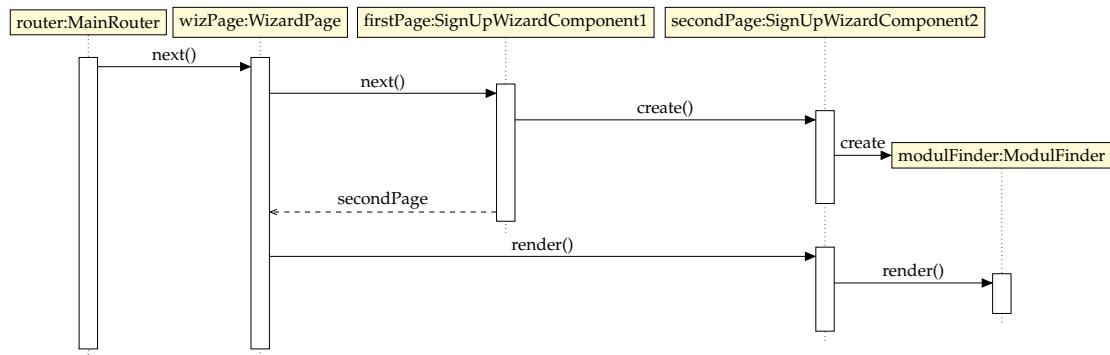


Abbildung 31: Zur zweiten Registrierungsseite wechseln

Die Abbildungen 32 bis 34 beschreiben einen Teil der Umsetzung des Anwendungsfalls „A140: Modul in Studienplan einfügen“.

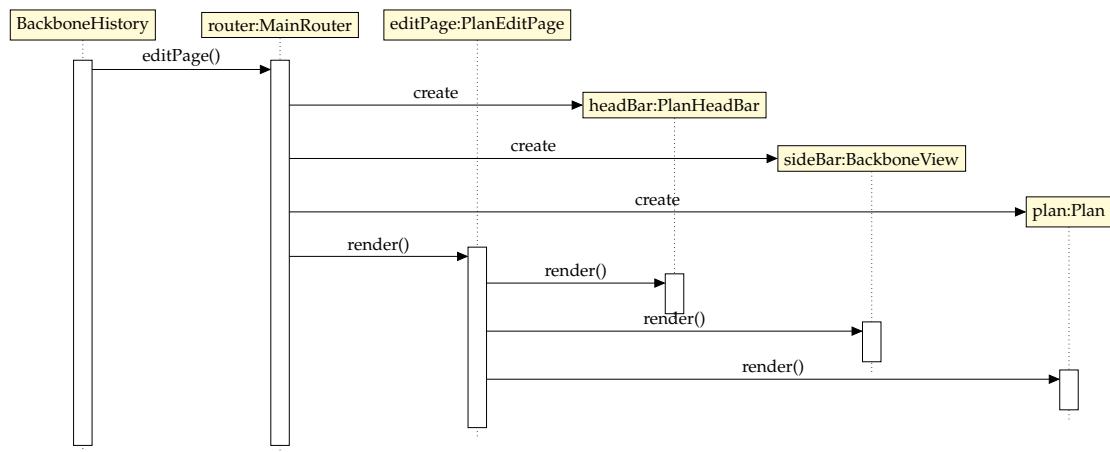


Abbildung 32: Bearbeitungsansicht anzeigen

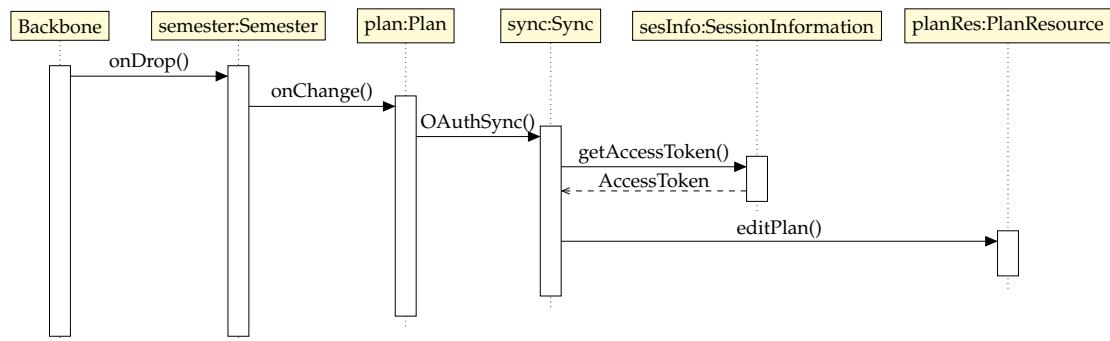


Abbildung 33: Modul im Plan einfügen: Informationen speichern(Clientseitig)

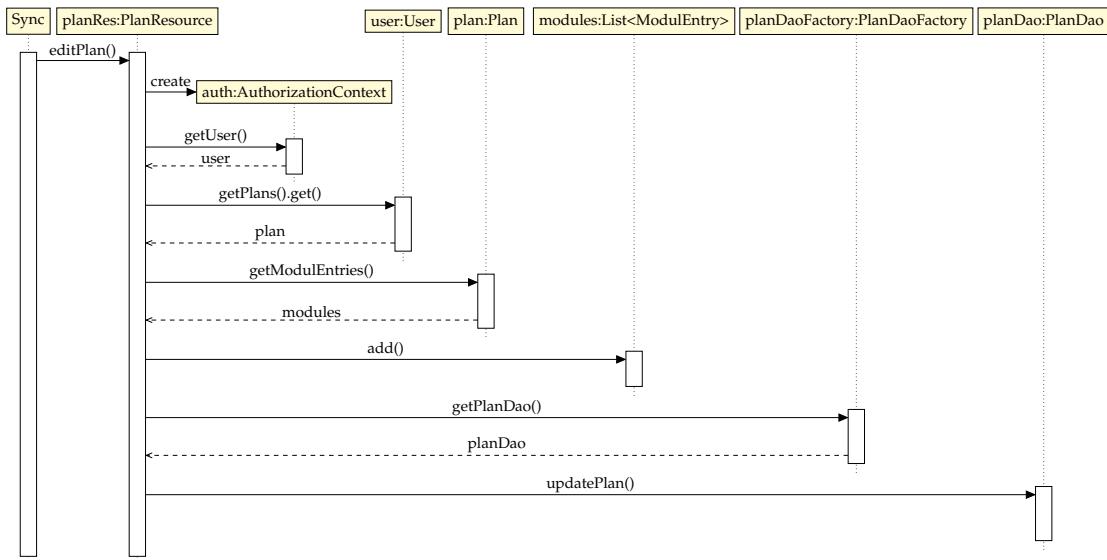


Abbildung 34: Modul im Plan einfügen: Informationen speichern(Serverseitig)

### 8.3. Beispiele für REST-Kommunikation

Im folgenden Sequenzdiagramm wird beispielhaft die Umsetzung eines Anwendungsfalls mit Fokus auf die REST-Kommunikation gezeigt (siehe Kapitel 4). GET-Parameter sind mit einem Stern (\*) versehen und werden in Kapitel 4.5 definiert.

Abbildung 35 beschreibt die REST-Kommunikations-Abfolge, welche für die Umsetzung des Anwendungsfalls „A230: Studienplan vervollständigen lassen“ notwendig ist. Hierbei wird angenommen, dass ein Vorschlag erfolgreich generiert werden kann und anschließend nicht verworfen wird.

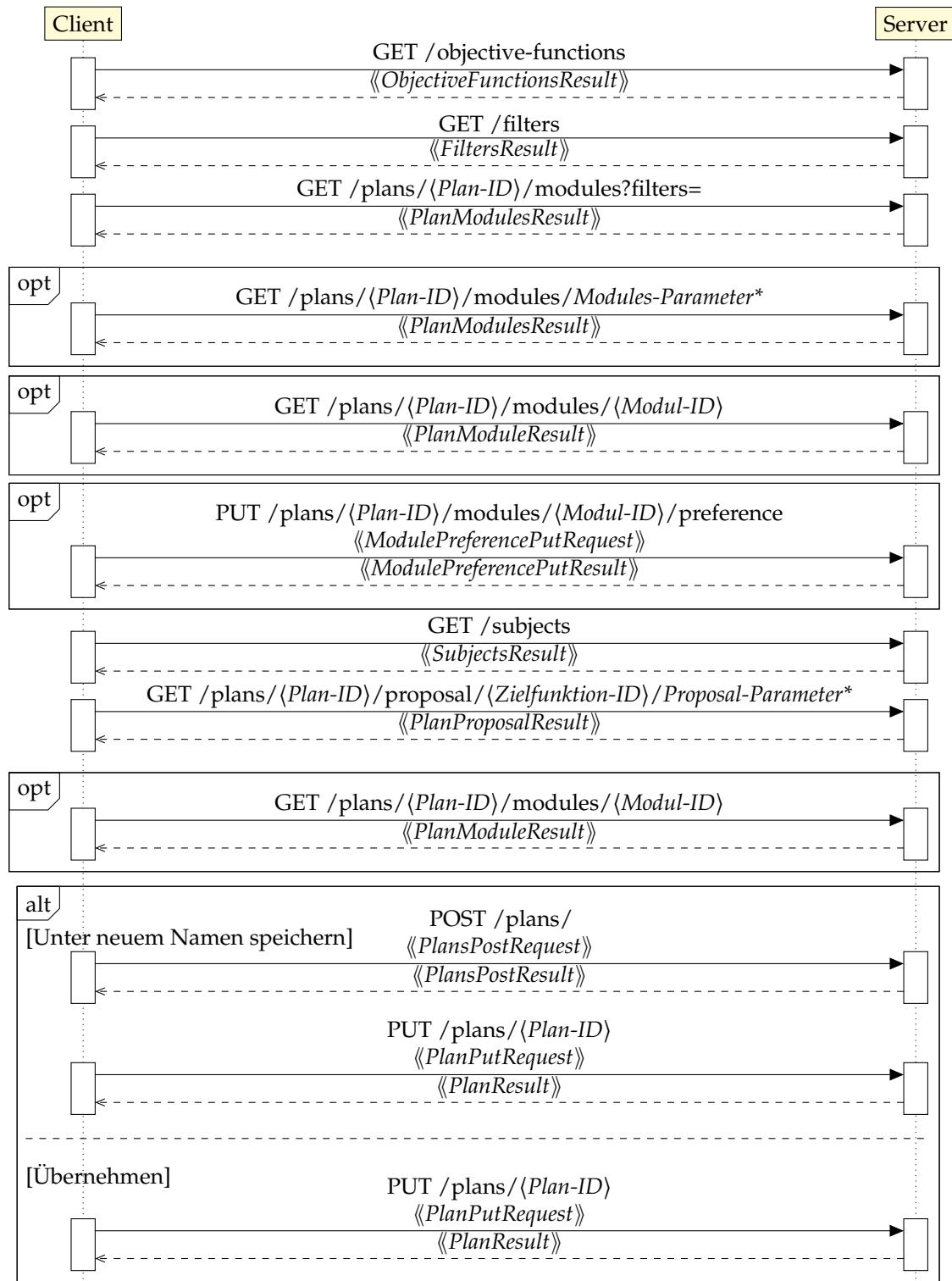


Abbildung 35: Für den Anwendungsfall „A230: Studienplan vervollständigen lassen“ nötige REST-Kommunikations-Abfolge.

## 9. Klassenindex

Es folgen der Client-Klassen-Index (Klassen in *edu.kit.informatik.studyplan.client.\**) und der Server-Klassen-Index (Klassen in *edu.kit.informatik.studyplan.server.\**). Package-Namen in Klammern stehen stets relativ zu *edu.kit.informatik.studyplan*.

### 9.1. Client-Klassen

|                             |                                                      |
|-----------------------------|------------------------------------------------------|
| ComparisonPage              | PassedModuleCollection                               |
| ComparisonView              | PassedModulePlan                                     |
| CookieModel                 | Plan ( <i>client.model.plans</i> )                   |
| Discipline                  | Plan ( <i>client.view.components.uielement</i> )     |
| DisciplineCollection        | PlanCollection                                       |
| EventBus                    | PlanEditPage                                         |
| Filter                      | PlanHeadBar                                          |
| FilterCollection            | PlanList                                             |
| FilterComponent             | PlanListElement                                      |
| GenerationWizardComponent1  | Preference                                           |
| GenerationWizardComponent2  | ProfilePage                                          |
| GenerationWizardComponent3  | ProposalInformation                                  |
| Header                      | ProposalSidebar                                      |
| LanguageManager             | ProposedHeadBar                                      |
| LoginPage                   | ProposedPlan                                         |
| MainPage                    | RangeFilter                                          |
| MainRouter                  | RegularHeadBar                                       |
| MainView                    | SearchCollection                                     |
| Module                      | SelectFilter                                         |
| ModuleBox                   | Semester ( <i>client.model.plans</i> )               |
| ModuleCollection            | Semester ( <i>client.view.components.uielement</i> ) |
| ModuleConstraint            | SemesterCollection                                   |
| ModuleFilter                | SessionInformation                                   |
| ModuleFinder                | SignUpWizardComponent1                               |
| ModuleInfoSidebar           | SignUpWizardComponent2                               |
| ModuleList                  | Student                                              |
| Notification                | SubjectDiscipline                                    |
| NotificationBox             | SubjectDisciplineCollection                          |
| NotificationCentre          | Sync                                                 |
| NotificationCollection      | TemplateManager                                      |
| OAuthCollection             | TextFilter                                           |
| OAuthModel                  | WizardComponent                                      |
| ObjectiveFunction           | WizardPage                                           |
| ObjectiveFunctionCollection |                                                      |

## 9.2. Server-Klassen

|                                       |                                        |
|---------------------------------------|----------------------------------------|
| AbstractSecurityProvider              | ModuleConstraintType                   |
| AtomObjectiveFunction                 | ModuleDao                              |
| AttributeFilter                       | ModuleDaoFactory                       |
| AuthResource                          | ModuleDescription                      |
| AuthorizationCodeGrant                | ModuleEntry                            |
| AuthorizationContext                  | ModuleOrientation                      |
| AuthorizationRequestFilter            | ModulePreference                       |
| AuthorizationScope                    | ModulePreferencesAtomObjectiveFunction |
| AverageObjectiveFunction              | ModuleResource                         |
| Category                              | ModuleType                             |
| CategoryFilter                        | MultiFilter                            |
| CompulsoryFilter                      | MultiplicationObjectiveFunction        |
| ContainsFilter                        | NameFilter                             |
| CreditPointsFilter                    | Node                                   |
| CycleType                             | NodeWithOutput                         |
| CycleTypeFilter                       | NodeWithoutOutput                      |
| Discipline                            | ObjectiveFunction                      |
| DisciplineFilter                      | ObjectiveFunctionResource              |
| Filter                                | OverlappingModuleConstraintType        |
| FilterDescriptor                      | PartialObjectiveFunction               |
| FilterResource                        | PasswordCredentialsGrant               |
| FilterType                            | Plan                                   |
| GenerationManager                     | PlanConverterResource                  |
| Generator                             | PlanDao                                |
| GetParameters                         | PlanDaoFactory                         |
| GrantType                             | PlanGeneratorResource                  |
| HibernateModuleDao                    | PlanLinkModuleConstraintType           |
| HibernatePlanDao                      | PlanModulesResource                    |
| HibernateUserDao                      | PlanVerifierResource                   |
| HibernateUtil                         | PlansResource                          |
| ImplicitGrantType                     | PreferenceType                         |
| JSONObject                            | PrerequisiteModuleConstraintType       |
| ListFilter                            | RESTClient                             |
| MainApplication                       | RangeFilter                            |
| MinimalECTSAtomObjectiveFunction      | RefreshGrant                           |
| MinimalSemestersAtomObjectiveFunction | SecurityProvider                       |
| MinimalStandardAverageDeviation       | Semester                               |
| →ECTSAtomObjectiveFunction            | SemesterLinkModuleConstraintType       |
| Module                                | SemesterType                           |
| ModuleAttributeNames                  | SimpleGenerator                        |
| ModuleConstraint                      | StandardVerifier                       |

|                            |                     |
|----------------------------|---------------------|
| StudentResource            | UserDaoFactory      |
| ThresholdObjectiveFunction | VerificationManager |
| TrueFilter                 | VerificationResult  |
| TypeFilter                 | VerificationState   |
| UnknownGrant               | Verifier            |
| User                       | WeightFunction      |
| UserDao                    |                     |

## 10. Gestrichene optionale Anforderungen

- Teilen von Studiengängen (zu viel Aufwand, verursacht viel Datenmüll oder ist aus Datenschutzgründen fraglich)
  - 1. Geteilter Studienplan wird als Snapshot gespeichert -> Datenmüll + Zusätzlicher Verwaltungsaufwand
  - 2. Es wird Lesezugriff auf den Studienplan gestattet -> Zugriffsmanager müsste implementiert werden, da sonst möglicherweise jemand nach Jahren den Studienplan einer Person liebt, die das zu diesem Zeitpunkt gar nicht mehr will.
  - 3. Studienplan kopieren -> was wenn anderes Studienfach? -> andere abgeschlossene Module
  - weitere Lösungen haben ähnliche Probleme und sind mit viel Aufwand verbunden
- Login über Shibboleth Identity Provider (kann durch Konfigurieren des Servers erreicht werden, wäre jedoch zu viel bürokratischer Aufwand)
- Modulübersicht
  - Filter
    - \* Angebotenes Semester wurde ersetzt durch (Semesterzyklus -> Sommer-, Wintersemester oder beides)
    - \* Fachrichtung und Kategorie wurden kombiniert
  - Rückgängig-Button -> zu viel Entwurfs- und Implementierungsaufwand
  - Ausgeschlossene Module wurde entfernt, da Bewertung ähnliche Funktion bietet
  - Vervollständigen eines Studienplans gibt nur einen Plan zurück, nicht mehrere (weniger Verwaltungsaufwand nach der Generierung und bessere Kompatibilität mit dem Generierungstool, das später genutzt werden soll)

## 11. Implementierungsplan

Für die Implementierung planen wir, das Team in zwei Gruppen einzuteilen. Eine Gruppe wird sich mit der Server-Entwicklung und eine mit der Client-Entwicklung be-

schäftigen. Dies fördert die Produktivität, da sich nur die Hälfte des Teams in Javascript einarbeiten muss. Zudem müssen so Entwickler nicht zwischen mehreren Programmiersprachen wechseln.

Falls es bei der Server-Entwicklung zu zeitlichen Engpässen kommt, so können Personen aus dem Client-Team am Server mitarbeiten, weil hier nur eine geringe Einarbeitungszeit nötig ist.

Im Folgenden sind vorläufige Implementierungspläne für beide Entwicklungsteams dargestellt.

### **11.1. Server-Implementierung**

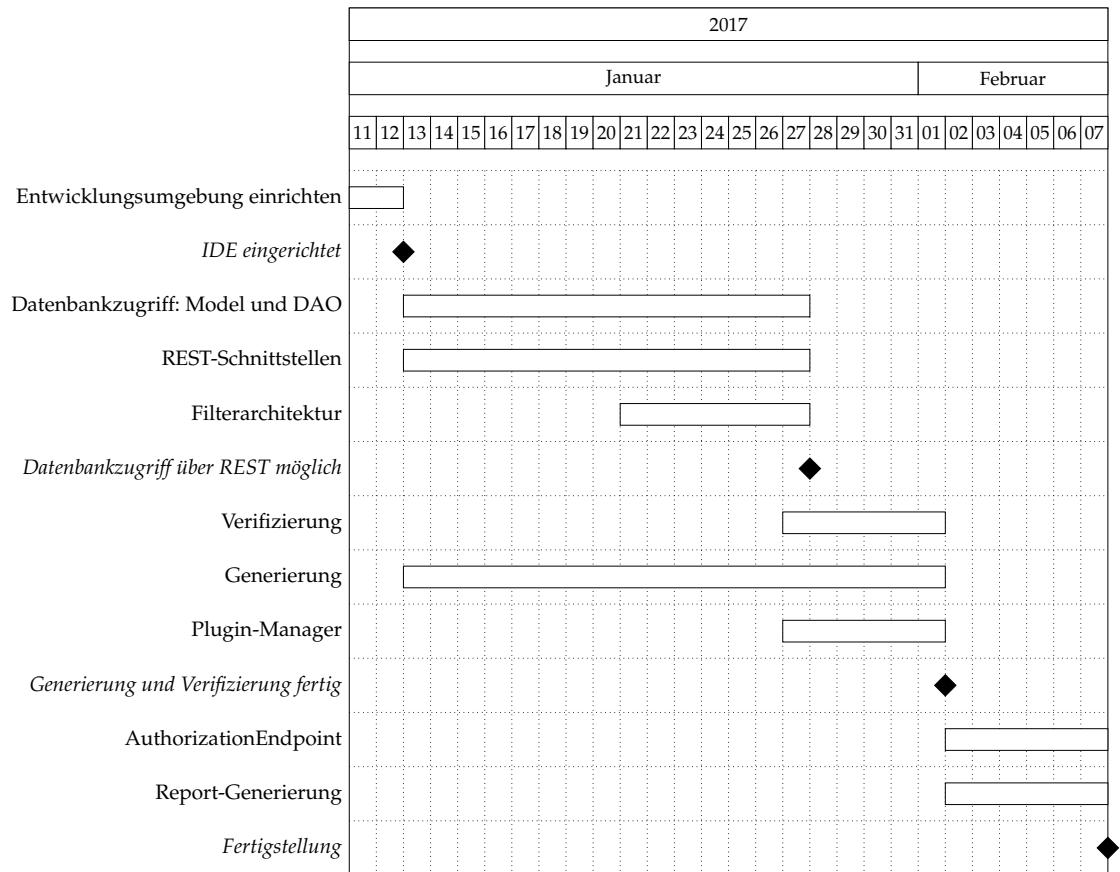


Abbildung 36: Implementierungsplan Server

## **11.2. Client-Implementierung**

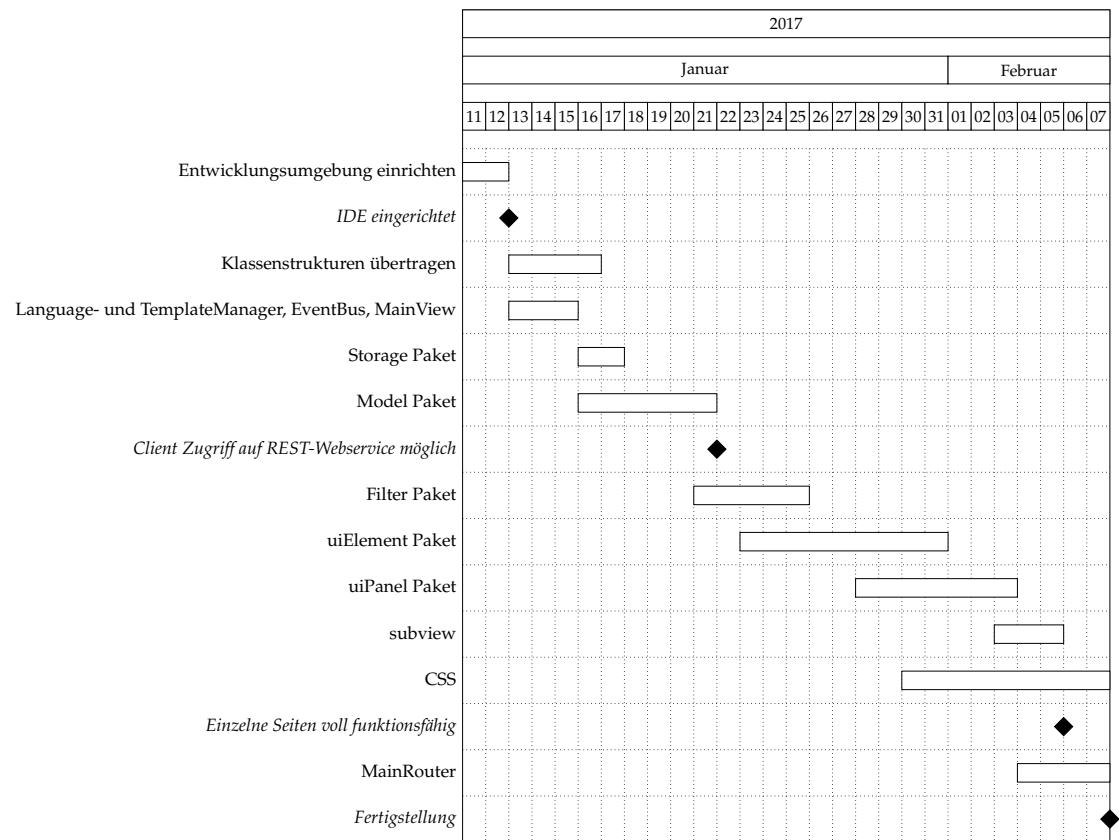


Abbildung 37: Implementierungsplan Client

# Anhang

## A. Abbildungen

## Literatur

- [1] D. Hart. *The OAuth 2.0 Authorization Framework*. <https://tools.ietf.org/html/rfc6749>. IETF. 2012.
- [2] T. Berners-Lee, R. Fielding and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. <https://tools.ietf.org/html/rfc3986>. Network Working Group. 2005.
- [3] Jeremy Ashkenas. *Backbone.js*. <http://backbonejs.org>. DocumentCloud. 2016.