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

Implementierungsbericht

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

14. Februar 2017

# Inhaltsverzeichnis

# 1. Einleitung

Der Implementierungsbericht des Projekts „Studienplanung als Generierung von Workflows mit Compliance-Anforderungen: Planerstellung und Visualisierung“ beschreibt die Umsetzung des zuvor im Pflichtenheft und Entwurf spezifizierten Projekts. In über 14000 Zeilen Java-, Javascript-, HTML- und CSS-Code (Leerzeilen und Kommentare nicht eingerechnet) liegt jetzt eine fertige, nur noch zu optimierende Anwendung vor. In diesem Bericht möchten wir dabei auf implementierte und ausgelassene Features eingehen, Änderungen beschreiben und begründen, sowie die Implementierungsphase reflektieren. Zusammenfassend ist dieser Bericht Ende einer arbeitsintensiven, aber erfolgreichen Phase, die als Endprojekt ein einmalig sinnvolles, dringend notwendiges und jedem Studenten zu empfehlendes Studienplanungshilfsmittel zur Verfügung stellt.

# 2. Änderungen am Entwurf

## 2.1. Änderungen an der Datenbasis

Um die gestellte Moduldatenbank in unseren Entwurf zu integrieren, haben wir diese um *Fields* (Bereiche) und *Rule-Groups* erweitert.

*Fields* modellieren Bereiche eines Studiengangs. Zu jedem Bereich gehören eine Mindestanzahl an ECTS, die in diesem Bereich belegt werden muss. Ist das Attribut *choosable* gesetzt, so verfügt ein Bereich auch über Vertiefungsfächer. Damit kann der Benutzer bei der Generierung in jedem auswählbaren Bereich ein Vertiefungsfach angeben, aus welchem der Generierer bevorzugt Module wählen soll. Diese Änderung sorgt unter anderem dafür, dass die nötigen 180 ECTS für den Bachelor-Abschluss nicht hartkodiert werden müssen, sondern aus der Summe der ECTS der zu einem Studiengang gehörigen Bereiche berechnet werden können.

*Rule-Groups* beschreiben Gruppen von Modulen, die bezüglich der Modulanzahl beschränkt sind. Das heißt, dass innerhalb einer Gruppe eine Mindest- und eine Maximalanzahl an Modulen vorgeschrieben ist, um ein Studienfach erfolgreich abschließen zu können. Dies stellt zum Beispiel sicher, dass mindestens zwei Stammmodule sowie ein Proseminar belegt werden.

Diese Änderungen ermöglichen eine stärkere Abstraktion der Bedingungen eines Studiengangs, was die Modellierung von unterschiedlichen Studiengängen vereinfacht.

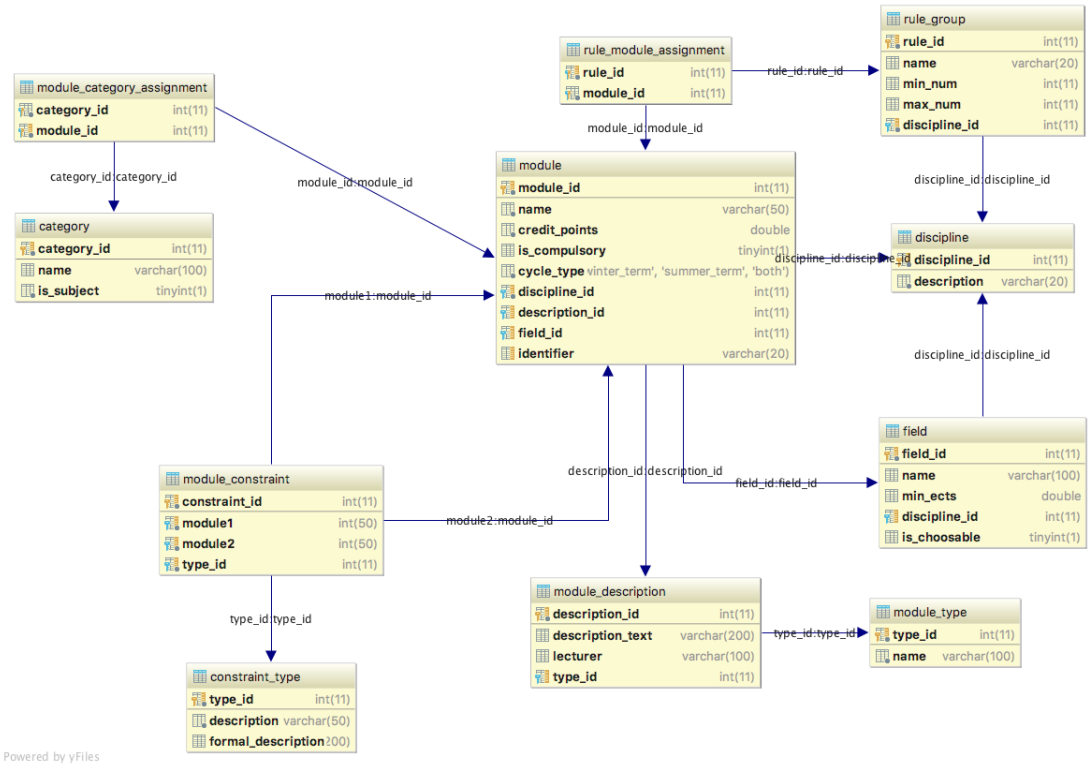


Abbildung 1: Übersicht über die Struktur der Moduldatenbank

## 2.2. Änderungen an der REST-Schnittstelle

Nachfolgend eine Auflistung aller atomaren Werte und JSON-Datenklassen, deren Definitionen sich geändert haben.

### JSON-Atome

| Bezeichner | Erklärung |
|---|---|
| ⟨*Modul-Turnus*⟩ | ∈ {`"WT"`, `"ST"`, `"both"`}. |
| ⟨*Semester-Typ*⟩ | ∈ {`"WT"`, `"ST"`}. |
| ⟨*Semester-Zahl*⟩ | Zahl des aktuellen Semesters des Benutzers. |
| ⟨*Filter-URI-Identifier*⟩ | String, mit welchem der Filter in GET-Parametern identifiziert wird. |

### JSON-Datenklassen

⟪*Student*⟫ = {
    `"discipline"`: ⟪*Studienfach* [id]⟫,
    `"study-start"`: ⟪*Studienbeginn*⟫,
    `"passed-modules"`: [⟪*Modul* [id, name, creditpoints, lecturer, semester]⟫, ...],
    `"current-semester"`: ⟨*Semester-Zahl*⟩
}

⟪*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*⟩,
    `"compulsory"`: ⟨**`true`**|**`false`**⟩,
    `"description"`: ⟨*Modul-Beschreibung*⟩,
    `"constraints"`: [⟪*Constraint*⟫, ...]
}

⟪*Filter*⟫ = {
    `"id"`: ⟨*Filter-ID*⟩,

```
    "name": ⟨Filter-Name⟩,
    "uri-name": ⟨Filter-URI-Identifier⟩,
    "default-value": ⟨Filter-Default⟩,
    "tooltip": ⟨Filter-Tooltip⟩,
    "specification": ⟨⟨Filter-Eigenschaften⟩⟩
}
```

```
⟨⟨Studienplan⟩⟩ = {
    "id": ⟨Studienplan-ID⟩,
    "status": ⟨Studienplan-Status⟩,
    "creditpoints-sum": ⟨Studienplan-Gesamt-Creditpoints⟩,
    "name": ⟨Studienplan-Name⟩,
    "modules": [⟨⟨Modul [id, name, semester, creditpoints, cycle-type, lecturer,
        preference]⟩⟩, ...],
    "violations": [⟨⟨Constraint⟩⟩, ...],
    "field-violations": [⟨⟨Field [name, min-ects]⟩⟩, ...],
    "rule-group-violations": [⟨⟨Rule-Group⟩⟩, ...],
    "compulsory-violations": [⟨⟨Modul [name]⟩⟩, ...]
}
```

```
⟨⟨Field⟩⟩ = {
    "id": ⟨Field-ID⟩,
    "name": ⟨Field-Name⟩,
    "min-ects": ⟨Field-Mindest-ECTS⟩,
    "categories": [⟨⟨Kategorie⟩⟩, ...],
}
```

```
⟨⟨Rule-Group⟩⟩ = {
    "name": ⟨Rule-Group-Name⟩,
    "min-num": ⟨Rule-Group-Modul-Mindestanzahl⟩,
    "max-num": ⟨Rule-Group-Modul-Höchstanzahl⟩
}
```

```
⟨⟨ModulesResult⟩⟩ = {
    "modules": [⟨⟨Modul [id, name, creditpoints, lecturer, cycle-type]⟩⟩, ...]
}
```

```
⟨⟨ModuleResult⟩⟩ = {
    "module": ⟨⟨Modul [id, name, categories, cycle-type, creditpoints, lecturer,
        compulsory, description, constraints]⟩⟩
}
```

```
⟨⟨StudentPutRequest⟩⟩ = {
```

```
    "student": ⟪Student [discipline, study-start, passed-modules]⟫
}
```

```
⟪PlanResult⟫ = {
    "plan": ⟪Studienplan [id, name, status, creditpoints-sum, modules]⟫
}
```

```
⟪PlanPutRequest⟫ = {
    "plan": ⟪Studienplan [id, name, modules[id, semester]]⟫
}
```

```
⟪PlanPutResult⟫ = {
    "plan": ⟪Studienplan [id, name, modules[id, semester], status]⟫
}
```

```
⟪PlanModulesResult⟫ = {
    "modules": [⟪Modul [id, name, creditpoints, lecturer, cycle-type, preference]⟫,
        ...]
}
```

```
⟪PlanModuleResult⟫ = {
    "module": ⟪Modul [id, name, categories, cycle-type, creditpoints, lecturer,
        preference, compulsory, description, constraints]⟫
}
```

```
⟪PlanVerificationResult⟫ = {
    "plan": ⟪Studienplan [id, status, violations, field-violations,
        rule-group-violations, compulsory-violations]⟫
}
```

```
⟪PlanProposalResult⟫ = {
    "plan": ⟪Studienplan [status, modules]⟫
}
```

```
⟪FieldsResult⟫ = {
    "fields": [⟪Field⟫, ...]
}
```

#### REST-Zugriffsstruktur

Hinzugefügt:

| Methode | URI | Beschreibung | Anfrage-Parameter /Rückgabewerte |
|---|---|---|---|
| GET | / | Abfrage, ob der REST-Service antwortet | Anfrage: —<br>Rückgabe: — (200 OK) |
| GET | /fields | Gibt alle Bereiche des zum Studenten gehörenden Studiengangs zurück | Anfrage: —<br>Rückgabe: ⟪*FieldsResult*⟫ |

Entfernt:

- POST /plans/⟨*Plan-ID*⟩ (Plan duplizieren):
  Kann mit DELETE, POST und PUT erreicht werden (dabei geht aber der Verifikationsstatus verloren, s.u.).
- GET /subjects:
  Rückgabe in GET /fields verfügbar.

Geändert:

- GET /modules: ⟪*ModulesResult*⟫ geändert (s.o.)
- GET /modules/⟨*Module-ID*⟩: ⟪*ModuleResult*⟫ geändert (s.o.)
- PUT /plans/⟨*Plan-ID*⟩ (Plan ersetzen):
  Nimmt jetzt nur noch Namen und Modulbelegung entgegen, der Verifikationsstatus wird auf „nicht verifiziert“ gesetzt.
  Anfrage: ⟪*PlanPutRequest*⟫ (geändert, s.o.)
  Rückgabe: ⟪*PlanPutResult*⟫ (geändert, s.o.)
- GET /plans/⟨*Plan-ID*⟩: ⟪*PlanResult*⟫ geändert (s.o.)
- GET /plans/⟨*Plan-ID*⟩/modules: ⟪*PlanModulesResult*⟫ geändert (s.o.)
- GET /plans/⟨*Plan-ID*⟩/modules/⟨*Modul-ID*⟩: ⟪*PlanModuleResult*⟫ geändert (s.o.)
- GET /plans/⟨*Plan-ID*⟩/verification: ⟪*PlanVerificationResult*⟫ geändert (s.o.)
- GET /plans/⟨*Plan-ID*⟩/proposal/⟨*Zielfunktion-ID*⟩:
  Da sich die Modellierung der Vertiefungsfächer geändert hat (siehe Kap. 2.1), werden auch die GET-Parameter wie folgt angepasst:
  Anfrage-Parameter:
  max-semester-ects=⟨*Semester-ECTS-Maximum*⟩
  fields=⟨*Field-ID*⟩,...,⟨*Field-ID*⟩
  (Auflistung gewählter Bereiche)
  field-*k*=⟨*Vertiefungsfach-ID*⟩
  (für jeden gewählten Bereich mit ID *k*)
  Rückgabe: ⟪*PlanProposalResult*⟫ (geändert, s.o.)

- GET /plans/⟨*Plan-ID*⟩/pdf:
  Zurückgeschickt wird ein Dokument vom Typ "text/html". Der Nutzer kann dieses entweder manuell in ein PDF umwandeln (von gängigen Browsern unterstützt) oder direkt ausdrucken.
- PUT /student:
  Beim Ersetzen der Studenteninformationen werden die bestandenen Module aus allen Plänen gelöscht, in denen sie vorkommen. Alle Pläne werden weiterhin als „nicht verifiziert" markiert.
  Anfrage: ⟪*StudentPutRequest*⟫ (geändert, s.o.)
- GET /student:
  Es wird zusätzlich die ⟨*Semester-Zahl*⟩ zurückgeschickt (s. ⟪*Student*⟫, geändert)

## 2.3. Client-Änderungen

In der gesamten Client-Implementierung wurden Attributnamen den JSON-Definitionen angepasst, um die Standard-parse- und toJSON-Methoden nutzen zu können und Übersichtlichkeit durch Einheitlichkeit herzustellen.

### 2.3.1. View-Paket

#### subview

- NotFoundPage eingefügt

ComparisonPage

- aus zeitlichen Gründen entfernt

Header

- Attribute
  - hinzugefügt
    tagName
    events
    template
- Methoden
  - hinzugefügt
    initialize
    render
    goHome

showProfile
logoutUser

LoginPage

- Attribute
  - hinzugefügt
    tagName
    template

MainPage

- Attribute
  - hinzugefügt
    planCollection
    tagName
    template
    events
- Methoden
  - hinzugefügt
    initialize

PlanEditPage

- Attribute
  - hinzugefügt
    planHeadBar
    planView
    isSignUp
  - umbenannt
    sidebar -> moduleFinder
    passedModules -> model
- Methoden
  - hinzugefügt
    planHeadBar
    planView
    isSignUp
  - entfernt
    hideModuleDetails
    showModuleDetails
    close

ProfilPage

- Attribute
  - hinzugefügt
    isSidebar
    model
    planHeadBar
    isPreferencable
- Methoden
  - hinzugefügt
    saveModules

WizardPage

- Attribute
  - hinzugefügt
    events
- Methoden
  - hinzugefügt
    onFinish

**Filter-Paket** FilterComponent

- Attribute
  - hinzugefügt
    className
- Methoden

ModuleFilter

- Attribute
  - hinzugefügt
    filterCollection
    events
- Methoden
  - hinzugefügt
    showFilterSettings

RangeFilter

- Attribute
  - entfernt
    min
    max

- Methoden
  - hinzugefügt
    updateVal

SelectFilter

- Attribute
  - entfernt
    options
- Methoden
  - hinzugefügt
    dropDownChange

TextFilter

- Methoden
  - hinzugefügt
    textChange

**Uielement-Paket** In der Klasse ModuleBox wurde folgendes geändert:

- voteUp() wurde als Listener für das Ereignis einer positiven Präferenz hinzugefügt
- voteDown() wurde als Listener für das Ereignis einer negativen Präferenz hinzugefügt
- isPreferencable gibt an, ob das Modul „präferierbar“ ist
- isDraggable gibt an, ob das Modul in ein Semester gezogen werden kann
- isPassedPlanModule gibt an, ob das Modul in einem Plan liegt, welcher bestandene Module anzeigt. Dies hat zur Folge, dass bestandene Module ebenfalls gezogen oder gelöscht werden können.

In der Klasse ModuleFinder wurde folgendes geändert:

- isPlaced wurde entfernt, da die Funktion "Module, welche im Plan vorhanden sind auszublendenäktuell nicht vorhanden ist.
- isSidebar wurde entfernt, da die Frage, ob der ModuleFinder eine Sidebar ist oder nicht, nun ausschließlich über CSS gelöst wird und der Javascript-Code des ModuleFinders hierauf keinen Einfluss nimmt

In der Klasse ModuleInfoSidebar wurde folgendes geändert

- Methode onChange() wurde umbenannt in reload()

In der Klasse ModuleList wurde folgendes geändert:

- modules wurde in moduleCollection umbenannt
- isDraggable wurde als Attribut hinzugefügt
- isRemovable wurde als Attribut hinzugefügt
- onChange() wurde in reload() umbenannt

In der Klasse NotificationBox wurde folgendes geändert:

- blurOut() wurde entfernt, da die onClose() die Funktionalität bereits implementiert

Die Klasse PassedModulePlan ist nicht länger notwendig.
In der Klasse Plan wurde folgendes geändert:

- align wurde gelöscht, da nicht notwendig
- onChange() wurde in reload() umbenannt
- isAddable wurde hinzugefügt (ob dem Plan semester hinzugefügt wurden)
- isPreferencable wurde dem Plan hinzugefügt (ob Module präferiert werden dürfen)
- isPassedPlan wurde dem Plan hinzugefügt (ob dies ein Plan aus bestandenen Modulen ist)

In der Klasse PlanListElement wurde folgendes geändert:

- setChecked(:string) zum Setzen des Checkbox-Werts hinzugefügt
- isChecked():boolean zum Erhalten des Checkbox-Werts hinzugefügt
- delete() heißt nun deletePlan() (delete ist in Javascript ein Keyword)
- duplicate() heißt nun duplicatePlan()
- export() heißt nun exportPlan()
- show() heißt nun showPlan()

Es wurde die Klasse ProfileHeadBar hinzugefügt, welche die folgenden Methoden hat:

- savePlan() speichert die bestandenen Module
- deleteUser() löscht den Nutzer
- goHome() leitet den Nutzer zurück zur Hauptseite

In der Klasse Semester wurde folgendes geändert:

- isPassedSemester besagt, ob das Semester bereits vergangen ist
- isPassedPlan besagt, ob der Plan ein Plan mit ausschließlich bestandenen Modulen ist
- isPreferencable besagt, ob die Module präferierbar sind

#### Uipanel-Paket

- In GenerationWizardComponent2 und in SignUpWizardComponent2 werden die onChange()-Methoden nicht gebraucht, da sie die fertigen Seiten moduleFinder beziehungsweise ProfilPage nutzen, und diese selbstständig triggern und speichern.
- GenerationWizardComponent3 nutzt die neu eingeführte FieldCollection und benötigt neben onChange() zwei weitere Methoden um auf das Betätigen der Slider zu reagieren.
- Die Klasse SignUpWizardComponent1 benötigt eine weitere onChange-Methode, da sie zwei Events enthält, sowie eine weitere Methode beginning() zur gekapselten Erstellung der Studienstart-Auswahl-Daten.

PlanList

- Attribute
  - hinzugefügt
    planListElements
- Methoden
  - hinzugefügt
    comparePlans
    deletePlans
    checkAllCheckboxChange

### 2.3.2. Storage-Paket

Keine Änderungen

### 2.3.3. Router-Paket

In der Klasse MainRouter wurde geändert:

- showLoading() sowie hideLoading() wurde hinzugefügt zum Anzeigen/Verstecken einer Ladeanzeige
- notFound() wurde hinzugefügt zum Anzeigen einer „Seite existiert nicht “-Seite
- logoutPage() wurde hinzugefügt um den Logut zu vollziehen

### 2.3.4. Model-Paket

#### Paket modules.*

- In der Klasse ModuleCollection wurde die Methode containsModule(moduleId) eingefügt, um testen zu können, ob ein Modul bereits in einer Modulcollection enthalten ist.
- Der Klasse Preferences wurde die Methode onChange angefügt, um auf Änderungen der Preferenz reagieren zu können.

**Paket plans.*** In der Klasse Plan wurden einige Methoden hinzugefügt:

- retrieveProposedPlan() um generierten Plan zu erhalten
- getEctsSum() zur Berechnung der ECTS Punkte im Plan
- addModule() um einem Semester ein neues Modul hinzuzufügen.
- Auf loadVerification() wurde hingegen verzichtet, da diese Information permanent im zugehörigen VerificationResult gespeichert ist, und bei Änderung in diesem Objekt neu geladen wird. Die Änderung des Change-Events von VerificationResult wird ebenfalls an Plan weitergeleitet.

In der Klasse ProposedPlan wurde hinzugefügt:

- setInfo() zum setzten eines Objekts vom Typ ProposalInformation, das die Informationen über die Wünsche eines Nutzers bei der Generierung kapselt

Es wurde die Klasse RuleGroup neu hinzugefügt, die eine rule-group kapselt, die bei der Verifizierung überprüft wird.

In der Klasse Semester wurde hinzugefügt:

- getEctsSum() zur Berechnung der ECTS-Summe in einem Semester
- onChange() Zur Weiterleitung des „change“-Events an SemesterCollection und Plan

In der Klasse SemesterCollection wurde hinzugefügt:

- addModule(m:Module) fügt dem Semester m.get('semester') das Modul m hinzu
- push(:Semester) fügt der SemesterCollection ein neues Semester hinzu.
- getEctsSum() berechnet die Summe der ECTS-Punkte aller Semester

Wie oben erwähnt wurde ebenfalls die neue Klasse VerificationResult eingeführt, die die Antwort einer Verifikations-Anfrage an den Server kapselt.

**System-Paket** Klasse Field wurde eingefügt, um eine Option, die der Nutzer zur besseren Generierung auswählen kann, zu repräsentieren. Auch wurde die Klasse FieldCollection, welche alle existenten Optionen zur Generierung enthält, eingefügt. Die Klasse SearchCollection arbeitet nun doch nicht mit LazyLoading sondern lädt alle Module direkt, da sich dies bereits als schnell genug herausgestellt hat.

**User-Paket** In der Klasse SessionInformation wurden folgendes geändert

- loggedIn wurde entfernt, da der Zustand von SessionInformation nun implizit über den Wert von access_token berechnet wird
- getLoginUrl() wurde hinzugefügt. Diese Methode berechnet die Login URL
- isLoggedIn() wurde hinzugefügt. Diese Methode prüft ob access_token vorhanden ist (und somit ob die Person eingloggt ist)

## 2.4. Server-Änderungen

### 2.4.1. Model-Paket

**Paket moduledata.*** Es wurden die Klassen Field und RuleGroup hinzugefügt, um die Modelländerungen abzubilden. Im ModuleDao-Interface wurden einige Bequemlichkeitsmethoden zum Finden von Kategorien, Bereichen, Vertiefungsfächern u. ä. hinzugefügt. Eine ConditionQueryConverter-Klasse unterstützt nun die Umwandlung von Condition-Objekten in SQL-Statements, die dann mit Hibernate an die Datenbank geschickt werden können.

### 2.4.2. Filter-Paket

Das Filter-Paket wurde grundlegend umgestaltet: Ein Filter enthält mittlerweile nur die Daten, die zum Filtern benötigt werden (wie z.B. konkret selektierte Elemente). Clientbezogene Daten (wie beispielsweise zulässige Intervallschranken oder eine Liste aller Auswahlelemente) liegen ausschließlich in den Filter-Deskriptoren. Diese kümmern sich auch um De- und Serialisierung von Filtern bei Anfragen.

**Paket condition** Da sich die jOOQ-Condition-Klasse als nicht sonderlich weiterverwertbar bzw. handhabbar erwies, wurde sie durch eine eigene Condition-Klasse ersetzt,

welche die drei nötigen Bedingungstypen als Binäroperatoren modelliert. Jeder Filter liefert nun eine Liste von Conditions zurück. Dadurch lassen sich SQL-Statements einfacher aus den Condition-Objekten zusammenbauen.

### 2.4.3. Generation-Paket

Die NodeWithoutOutput-Klasse wurde entfernt (da wir zwischen den Knoten mit und ohne Kindern nicht zu unterscheiden brauchen). Dafür wurde die Klasse NodesList hinzugefügt (zur Speicherung der Knoten).

### 2.4.4. Pluginmanager-Paket

Auf Implementierung eines Plugin-Systems wurde verzichtet. Standard-Generierer und -Verifizierer sind somit fest in die Software eingebaut (können aber nach wie vor später durch Black-Boxes ersetzt werden). GenerationManager und VerificationManager enthalten also fest verdrahtet unsere Nachbildungen, da nicht davon auszugehen ist, das eine Änderung des Generierungstools im laufenden Betrieb stattfindet.

### 2.4.5. Verification-Paket

Durch die sich geänderten Anforderungen wegen der Modelländerungen wurde VerificationResult um Attribute zum Festhalten von Field-, Rule-Group- und Compulsory-Violations erweitert und die Verifizierung entsprechend angepasst.

### 2.4.6. REST-Paket

Zur Verwaltung von Datenbank-Sessions wurden die Klassen SessionOpenFilter und SessionCloseFilter hinzugefügt, welche bei ankommender Anfrage bzw. vor abgeschickter Antwort eine Datenbank-Session eröffnen bzw. wieder schließen. Damit ist gewährleistet, dass während der Verarbeitung einer Anfrage stets auf die Datenbank zugegriffen werden kann und DAOs stets verfügbar und einsatzbereit sind.

Ferner wurde die Annotation @AuthorizationNeeded hinzugefügt, die eine REST-Ressourcenklasse bzw. deren Methoden derart markiert, dass Jersey beim Verarbeiten einer Anfrage mithilfe des AuthorizationRequestFilters User-Informationen bereitstellen kann, die zur Abwicklung einer solchen Anfrage notwendig sind. Eine

AuthorizationContextFactory-Klasse hilft dabei beim Injizieren eines Authorization-Contexts in die Ressource.

Die Klasse MyObjectMapperProvider konfiguriert einen ObjectMapper, der die De-/Serialisierung abwickelt. ValidationConfigContextResolver aktiviert BeanValidation, welche die Überprüfung speziell annotierter Attribute beim Deserialisieren von JSON-Objekten veranlasst.

**Paket authorization.endpoint** Die Klasse GrantTypeFactory dient nun als Multiplexer für die verschiedenen GrantTypes.

**Paket resources.*** In diesem Paket liegen die REST-Ressourcen-Klassen. Mehrere Ressourcen-Klassen wurden aus technischen Gründen als Sub-Ressourcen in Oberklassen eingegliedert, bspw. PlanVerificationResource in PlansResource. Es sind – in Zusammenhang mit den Änderungen in 2.2 – neue Ressourcen-Klassen hinzugekommen, welche auf neue Anfragen reagieren, wie z. B. FieldsResource oder MainResource.
Im Sub-Paket „json“ befinden sich Klassen zur De- und Serialisierung von JSON-Datenklassen, sogenannte Data-Transfer-Objects (DTOs). Diese dienen der korrekten Einhaltung der JSON-Spezifikationen und ermöglichen eine reibungslose Umwandlung von bzw. zu ihren Modell-Gegenstücken.
Da der PDF-Export einer HTML-Generierung gewichen ist, wurden hierfür die Klassen DisplayablePlan und PlanConverter angelegt, welche mithilfe der Apache VelocityEngine und einem Report-Template die HTML- und CSS-Ausgabe generieren.

### 2.4.7. Sonstiges

Die Klassen HibernateSessionFactoryListener und SessionCloseListener wurden hinzugefügt, um beim Starten bzw. Beenden des Webservices die Datenbankverbindung einmalig herzustellen bzw. zu beenden.

In der Klasse Utils befinden sich einige Hilfsmethoden, u.a. zur kontrollierten DAO-Nutzung.

# 3. Implementierte Features

## 3.1. Musskriterien

- Webbasierte Oberfläche weitgehend benutzerorientiert, Optimierung in Qualitätssicherung
- manuelle Studienplanbearbeitung, Speichern, Löschen und Verifizieren von Studienplänen weitgehend fehlerfrei möglich
- Login, Modulbewertung, Modulansicht und Generierung implementiert
- Modulare Implementierung gewährleistet Erweiterbarkeit

## 3.2. Wunschkriterien

- Studienpläne können (um-)benannt, als HTML-Dokument exportiert und dupliziert werden
- Auf die Einbindung des Login über den Shibboleth Identity Provider wurde aus Gründen des Bürokratieaufwands verzichtet
- Module können gefiltert werden und eine Detailansicht wurde implementiert
- beliebige Zielfunktionen können, sofern sie vom Server bereitgestellt werden, genutzt werden
- die Vergleichsansicht wurde auf Grund des hohen Implementierungsaufwandes bei wenig Gewinn an Funktion vorerst nicht implementiert
- auf einen „Rückgängig-Button“ wurde auf Grund des hohen Aufwands bei kaum existentem Komfortgewinn für den Nutzer verzichtet
- Studienpläne können nicht mit anderen Nutzern geteilt werden

# 4. Testfälle

In der Implementierungsphase wurden bereits erste Integrationstests durchgeführt. So wurden fast alle möglichen Nutzeraktionen einmal durchgespielt, um eine eine grundlegende Qualität der Implementierung zu gewährleisten.

## 4.1. Server

Auf dem Server wurden noch nicht für alle Klassen Unittests erstellt, da es sich hauptsächlich um Klassen zur Darstellung von REST-Schnittstellen und simple Java-Objekte handelt, die nur Getter und Setter besitzen. Die tatsächliche Funktionalität kann erst

in Verbindung mit einer Datenbank getestet werden. Diese Integrationstests werden in Qualitätssicherungsphase stattfinden.

Unittests:

| **Testklasse** | **Beschreibung** | **Status** |
|---|---|---|
| CategoryTest | Getter und Setter für `Category` getestet | ERFOLGREICH |
| DisciplineTest | Getter und Setter für `Discipline` getestet | ERFOLGREICH |
| SemesterTest | Test der Semester-Abstandsberechnung zwischen:<br>• zwei Wintersemestern<br>• zwei Sommersemestern<br>• zwischen Winter- und Sommersemester<br>• zwischen Sommer- und Wintersemester<br>Test der compareTo-Methoden | ERFOLGREICH |
| StandardVerifierTest | Testet, ob der `StandardVerifier` folgendes erkennt:<br>• fehlende Pflichtmodule<br>• Verletzung von Rule-Group-Bedingungen<br>• Verletzung von Bereichsbedingungen (Fields) | ERFOLGREICH |
| StandardVerifierConstraintTest | Testet ob der die Verletzung folgender `ConstraintTypes` in unterschiedlichen Ausartungen erkennt:<br>• Verletzung von Überlappung<br>• Verletzung von Plan-Zusammengehörigkeit<br>• Verletzung von Semester-Zusammengehörigkeit<br>• Verletzung von Voraussetzungen<br>Dadurch wurden auch die `isValid`-Methoden der entsprechenden `ConstraintTypes` getestet | ERFOLGREICH |

| | | |
|---|---|---|
| SimpleGeneratorTest | Testet einzelne Methoden des `SimpleGenerator`. Ein Plan mit einem einzelnen `ModuleEntry` wird:<br>• anhand der Benutzer-Präferenzen und `Constraints` verschiedener Art vervollständigt und modifiziert<br>• anhand einer Zielfunktion optimiert<br>Dafür werden sog. Mock-Objekte benutzt, die nur zum Testen erstellt wurden, d.h. bei der echten Verwendung des Systems werden diese nicht benutzt werden, sondern stattdessen die Daten aus der Datenbank. Das Verhalten von Methoden der Klassen `ModuleDao` und `Plan` wurde hierfür mittels Mockito angepasst. | ERFOLGREICH |
| NodesListTest | Testet die topologische Sortierung der Klasse `NodesList`. | ERFOLGREICH |
| VerificationManager | Getter für `Verifier` getestet. | ERFOLGREICH |

## 4.2. Client

Auf der Client-Seite wurde mit Hilfe der Frameworks Karma und Jasmine.js getestet. Während der Implementierungsphase wurde insbesondere das Modell mit Unit- und Integration-Tests überprüft. Dies war notwendig, da zu diesem Zeitpunkt noch nicht auf eine REST-Schnittstelle zu gegriffen werden konnte. Bei diesen Tests erreichten wir für die Modell-Klassen eine Testabdeckung von ca. 65 Prozent. Weitere Tests werden in der Qualitätssicherungsphase folgen. Für die Integrationstests war es die Zielsetzung, dass jede Anfrage bzw. Antwort an bzw. vom Server durch einen Testfall abgedeckt ist. Dieses Ziel wurde erreicht, wobei alle Tests erfolgreich laufen.

| **Testfall** | **Fragestellung** | **Status** |
|---|---|---|
| ModuleCollection-Initialisierung | Wird eine ModuleCollection für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |

| | | |
|---|---|---|
| ModuleConstraint-Initialisierung | Wird ein ModuleConstraint für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |
| Module-Initialisierung | Wird ein Modul für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |
| Preference-Initialisierung | Wird eine Preference für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |
| Plan-Initialisierung | Wird ein Plan für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |
| Discipline-Initialisierung | Wird eine Discipline für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |
| Filter-Initialisierung | Wird ein Filter für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |
| LanguageManager-Funktion | Gibt der Language Manager die richtigen Textbausteine zurück? | ERFOLGREICH |
| Notification-Collection-Initialisierung | Gibt getInstance() das richtige Objekt zurück? | ERFOLGREICH |
| ObjectiveFunctionCollection-Initialisierung | Wird eine ObjectiveFunctionCollection für gegebene Daten erfolgreich initialisiert? | ERFOLGREICH |
| TemplateManager-Funktion | Gibt der TemplateManager die richtige HTML-Ausgabe zurück? | ERFOLGREICH |
| CookieSync-Funktion | Speichert/Lädt CookieSync ein Modell erfolgreich? | ERFOLGREICH |
| OAuthSync-Funktion | Übergibt OAuthSync beim Speichern/Laden die richtigen Header? | ERFOLGREICH |
| MainView-Funktion | Lädt der MainView die richtigen Views und zeigt diese an? | ERFOLGREICH |

Tabelle 4: Geschriebene Unit-Tests

# 5. Review

## 5.1. Einhaltung des Implementierungsplans

Die Aufteilung des Teams in ein Server- und ein Client-Entwicklungsteam wurde beibehalten und erwies sich als sehr hilfreich, da sich jedes Teammitglied auf ein Teilgebiet konzentrieren konnte.

### 5.1.1. Server

Die Einrichtung der Entwicklungsumgebung lief ohne Probleme. Die Umsetzung vom Datenbankzugriff brauchte zunächst ein paar Tage mehr Zeit als veranlagt, was allerdings nicht zu Problemen führte, da die Schnittstellen klar definiert waren. Auch bei der Entwicklung der REST-Schnittstellen kam es zu minimalen Verzögerungen.
Mit der Implementierung der Filterarchitektur wurde bereits früher begonnen, sodass diese von der Datenbankzugriffsschicht verwendet werden konnte, allerdings wurde eine Änderung der Condition-Struktur nötig, wie in Kapitel 2.4.2 beschrieben, weshalb der Zugriff auf Module über Filter erster später möglich war.
Die Implementierungsreihenfolge der weiteren Komponenten wurde vertauscht. So wurden zunächst Report-Generierung und der Authorization-Endpoint implementiert, was beides zusammen innerhalb von zwei Tagen erledigt wurde. Erst daraufhin wurde mit der Implementierung der Verifizierung begonnen, was ebenfalls nur zwei Tage benötigte.
Die Generierung hingegen wurde fortlaufend über die ganze Dauer der Implementierungsphase entwickelt.
Die letzten vier Tage wurden für erste Integrationstests von Server und Client genutzt, um den Mehraufwand in der Qualitätssicherungsphase gering zu halten. Hierbei wurden ein paar Probleme in Bezug auf das Zusammenspiel von Datenbank- und REST-Schnittstelle sichtbar, welche es zu beheben galt. Näheres hierzu findet sich in Kapitel 5.2.

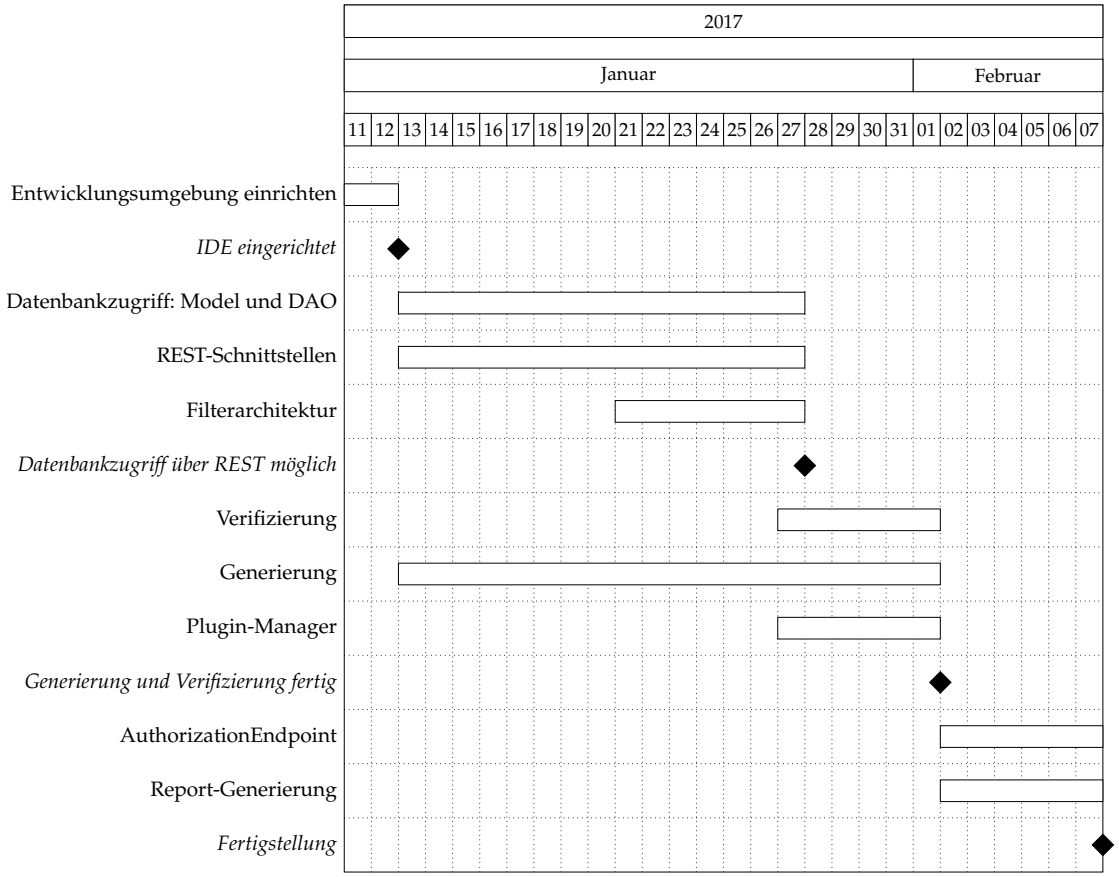


Abbildung 2: Ursprünglicher Implementierungsplan Server

### 5.1.2. Client

Aufgrund der heterogenen Struktur von Entwicklungsumgebungen im Javascript-Umfeld erwies sich zunächst die Einrichtung der Test- und Entwicklungsumgebung als eine größere Herausforderung als zunächst angenommen. Dies verzögerte den Entwicklungsprozess um zwei Tage. Aufgrund der steilen Lernkurve beim Erlernen von JavaScript verzögerte sich dann auch die erste Phase, in welcher die Modell-Klassen implementiert wurden um ca. vier bis fünf Tage. Es erwies sich hierbei als hilfreich, für den Client bereits zu diesem Zeitpunkt Integrationstests mit einem REST-Webservice-Mock zu schreiben, um so sicherstellen zu können, dass die Modell-Klassen der Spezifikation entsprechen. Dies führte zu einer deutlich höheren Stabilität des Modells, beanspruchte allerdings gleichzeitig zwei weitere Tage.
Da jedoch die Views trotz vorhandener Schnittstellen nicht ohne funktionierende Modell-Klassen getestet werden konnten, konnte mit diesen insgesamt erst ca. sieben Tage später begonnen werden. Dank der guten Einarbeitung während der Modell-Implementierung war es jedoch trotzdem möglich, den Zeitplan einzuhalten und die View rechtzeitig fertigzustellen. Bei der Implementierung der View zeigte sich, dass es sinnvoller ist, nach der „Top-Down“-Methode zu entwickeln und die Implementierung mit dem MainRouter sowie den Subviews zu beginnen. Durch diese Veränderung im Implementierungsplan ließen sich die View-Komponenten von Anfang an exzellent testen.
Insgesamt wurde das Projekt rechtzeitig und erfolgreich fertiggestellt – nicht zuletzt durch den Test der Kommunikation zwischen Client und Server, welcher in den letzten vier Tagen der Implementierungsphase durchgeführt wurde, um eine grundlegende Stabilität des Systems zu sichern.

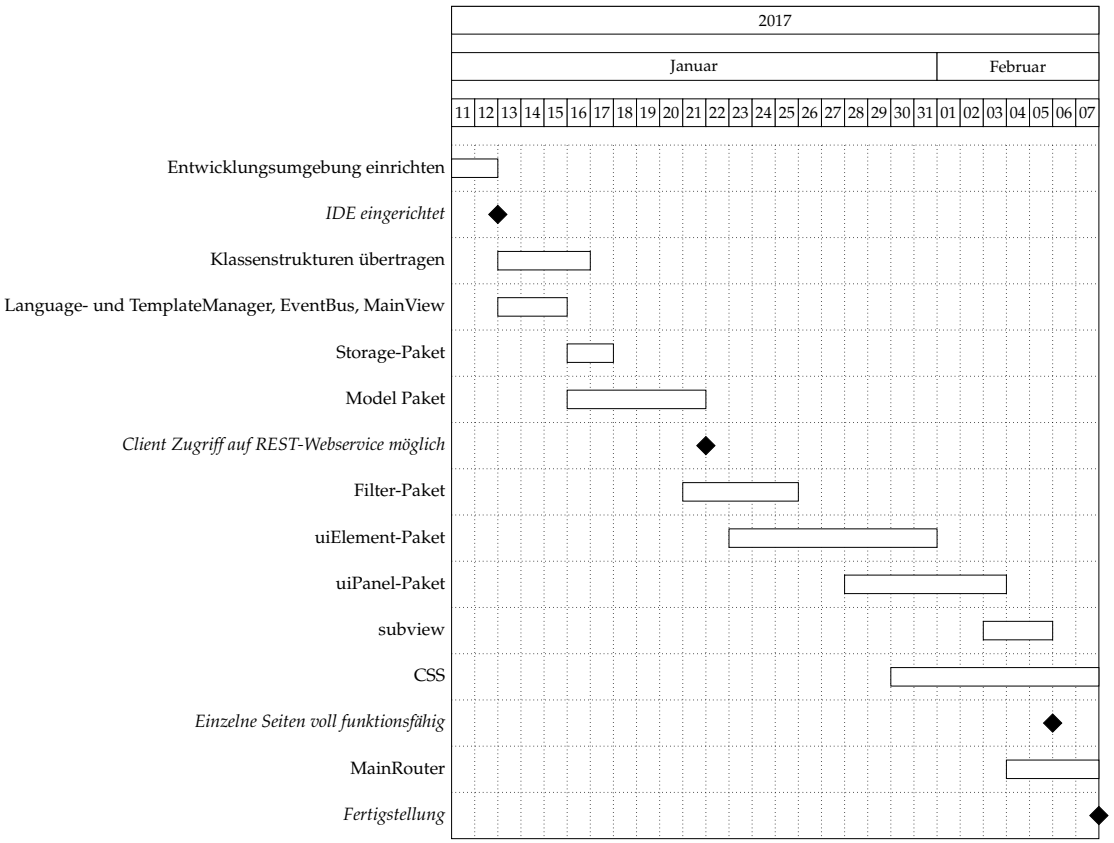


Abbildung 3: Ursprünglicher Implementierungsplan Client

## 5.2. Unerwartete Probleme

Selbstverständlich funktioniert bei der Implementierung eines Softwareprodukts nicht immer alles problemlos. Im Folgenden sind die Hürden dargestellt, die sich uns im Laufe der Implementierung stellten.

Zunächst ist anzumerken, dass keine Probleme durch Entwurfsbestandteile, welche sich als unpraktikabel erwiesen, aufgetreten sind. Der Entwurf musste nur minimal überarbeitet werden.
Die Client-Entwicklung wurde anfangs durch die steile Lernkurve beim Erlernen von JavaScript ausgebremst, da der Großteil des Teams bislang nur mit Java gearbeitet hatte – was sich aber nach dem Überwinden der ersten Hürden schnell löste. Auch stellte die Einrichtung der Client-Entwicklungsumgebung eine kleine Hürde dar.
Ein größeres Problem bestand im Zusammenspiel von REST- und Datenbankschnittstelle: Da das von uns verwendete Object-Relational-Mapping-Tool *Hibernate* sehr stark sogenanntes „Lazy Loading“[1] einsetzt, ist nach dem Schließen der Datenbankverbindung ein Nachladen von assoziierten Relationen von aus der Datenbank geladenen Instanzen nicht mehr möglich. So erschien es zunächst nötig, die Datenbankverbindung erst nach dem Antworten auf die Clientanfrage zu schließen. Die JAX-RS-Implementierung *Jersey* konstruiert Antworten auf Anfragen jedoch erst, nachdem alle Trigger, die nach dem Zusammenbauen der Antwort diese manipulieren können, aufgerufen wurden, weshalb es nicht einfach möglich war, mit einem solchen Trigger die Datenbankverbindung zu schließen. Da es jedoch keinen späteren Zeitpunkt gibt, zu dem das Schließen der Verbindung veranlasst werden kann, mussten sogenannte *Data-Transfer-Objects* (DTOs) hinzugefügt werden, die bereits vor dem Schließen der Datenbankverbindung alle Antwortdaten laden und zwischenspeichern, um dann serialisiert werden zu können. Dies konnte das Problem vollständig lösen.
Hinzu kam weiterhin, dass am Wochenende vor unserem Abgabetermin die komplette Internetanbindung des Wohnheims, in welchem vier der sechs Teammitglieder wohnen, ausfiel, was die Verwendung von Versionsverwaltung und Recherchen erheblich erschwerte.

## 5.3. Resümee

Abschließend kommen wir zu der Erkenntnis, dass wir direkt zu Beginn die Schnittstellen zwischen Client und Server hätten mocken sollen. Viele Probleme sind uns erst bei der Integration aufgefallen. Zudem hätten wir mehr kommunizieren müssen: Denn dadurch, dass zunächst jeder für sich selbst entwickelt hat, sind manche Probleme erst

[1] Das heißt, zu einem geladenen Objekt assoziierte Datenbankeinträge aus anderen Tabellen werden erst beim Zugriff auf die entsprechenden Getter nachgeladen

zuletzt aufgefallen. Durch bessere Kommunikation hätte man auch interne Fristen besser einhalten können.

Als sinnvoll erwiesen hat sich die Trennung von Client- und Server-Entwicklungsteam, da so eine klare Aufgabenverteilung im Team bestand. Außerdem hat jeder von uns sehr viele neue praktische Erkenntnisse im Bereich der Web-App-Implementierung gewonnen.

# Anhang

## A. Virtuelle Maschine

Die Virtuelle Maschine (VM) kann mit VirtualBox importiert werden. Es muss nur der Webbrowser innerhalb der VM geöffnet werden.

## B. Client

Firefox und Chrome müssen auf dem System bereits installiert sein!

### B.1. Linux (Ubuntu)

**Es könnte sein, dass die Befehle als root ausgeführt werden müssen (`sudo [Befehl]`)**

1. Installiere Apache:

```
apt-get install apache2
```

2. Ersetze Zeile mit `DocumentRoot [...]` in

```
/etc/apache2/sites-enabled/000-default.conf
```

durch

```
DEFINE WEBAPPROOT "[Pfad zu Repository]/Implementierung
   /Client/WebApp"
Include [Pfad zu Repository]/Implementierung/Client/
   Conf/apache.conf
```

3. Ordnerberechtigungen: `chmod -R +x [Pfad zu Repository]`

4. Apache Service neustarten: `service apache2 restart`

5. Installiere Ant `apt-get install ant`

6. Installiere ruby `apt-get install ruby`
7. Installiere npm `apt-get install npm`
8. Installiere grunt via npm `npm install -g grunt`
9. Erstelle Verknüpfung: `ln -s /usr/bin/nodejs /usr/bin/node`
10. Enable mod-rewrite `ln -s /etc/apache2/mods-available/rewrite.load /etc/apache2/mods-enabled/rewrite.load`

Im Verzeichnis WebApp:

1. Installiere sass via gem `gem install sass`
2. `npm install`
3. `npm install -g karma-cli`

Brackets installieren

1. Brackets ppa hinzufügen `add-apt-repository ppa:webupd8team/brackets`
2. `apt-get update`
3. Brackets installieren `apt-get install brackets`
4. Öffne Brackets und, wenn erwünscht, installiere die Plugins
5. *Annotate* und *Brackets Git*

## B.2. Windows

1. Lade `http://www.microsoft.com/de-de/download/details.aspx?id=29` herunter und installiere es
2. Lade Apache 2.4.25 (wenn 64-Bit-Betriebssystem, dann unbedingt 64-Bit-Version) von `http://www.apachehaus.com/cgi-bin/download.plx` herunter
3. Extrahiere das Verzeichnis Apache24 in C:\oder äquivalent

4. Setze in `C:/Apache24/conf/httpd.conf` in der Zeile `Define SRVROOT[...]` den Wert `Define SRVROOT "[Verzeichnis von Apache24]"`

   (wenn in C:\Apache24, setze `Define SRVROOT C:\Apache24`)

5. Eventuell Zeile mit `LoadModule rewrite_module modules/mod_rewrite.so` (o.ä.)

6. Ersetze in httpd.conf (siehe 4.) die Zeilen

```
DocumentRoot "${WEBAPPROOT}/htdocs"
<Directory "${WEBAPPROOT}/htdocs">
#
# Possible values for the Options directive are "None", "
   All",
# or any combination of:
#   Indexes Includes FollowSymLinks SymLinksifOwnerMatch
   ExecCGI MultiViews
#
# Note that "MultiViews" must be named *explicitly* --- "
   Options All"
# doesn't give it to you.
#
# The Options directive is both complicated and important.
    Please see
# http://httpd.apache.org/docs/2.4/mod/core.html#options
# for more information.
#
Options Indexes FollowSymLinks

#
# AllowOverride controls what directives may be placed in .
   htaccess files.
# It can be "All", "None", or any combination of the
   keywords:
#   Options FileInfo AuthConfig Limit
#
AllowOverride None

#
# Controls who can get stuff from this server.
#
Require all granted
</Directory>
```

(oder so ähnlich – wichtig ist das /htdocs, darauf kann man sich verlassen) durch

```
DEFINE WEBAPPROOT "[Pfad zu Repository]/Implementierung/
   Client/WebApp"
Include "[Pfad zu Repository]/Implementierung/Client/Conf/
   apache.conf"
```

Aktiviere mod_rewrite in der httpd.conf.

7. Öffne CMD als Administrator in Verzeichnis `[...]/Apache24/bin/` führe aus: `httpd -k install` (dies installiert Apache 2.4 als Service)

8. Öffne ApacheMonitor (befindet sich in `[...]/Apache24/bin/`) und starte Apache 2.4 (ggf. alles bejahen was kommt).
   **Achtung:** Hierfür darf kein Programm den Port 80 blockieren – insbesondere Skype macht das in der Standardeinstellung gerne mal!

9. Öffne im Webbrowser `http://localhost`: Ein Fenster mit dem Inhalt `It works: Studyplan Client` sollte sich öffnen

1. Installiere Brackets von `http://brackets.io/`

2. Öffne Brackets und installiere die Plugins *Annotate* und, wenn erwünscht, *Brackets Git*

3. Installiere ant von `http://ant.apache.org/bindownload.cgi` und füge das bin-Verzeichnis der PATH-Umgebungsvariable hinzu

4. Installiere nodejs von `https://nodejs.org/en/download/` und wähle bei Installation aus, dass die PATH-Variable gesetzt werden soll

5. Installiere `https://github.com/petetnt/brackets-sass-lint#readme` mit der brackets-npm-registry

6. Für SASS/SCSS benötigen wir Ruby, deshalb installieren wir es von `http://rubyinstaller.org/` und fügen es während der Installation dem PATH hinzu

7. Öffne CMD als Administrator in Verzeichnis `[Pfad zu Repository]/Implementierung/Client/WebApp`

8. Führe aus: `gem install sass` (Das installiert sass)

9. `npm install`

10. `npm install -g karma-cli`

### B.3. Kompilieren

1. Kompilieren mit

```
ant -DAPI_DOMAIN=[server_ip]:[port]/studplan/rest
```

2. Mögliche Flags:

```
- DAPI_DOMAIN -> server url
- DAPI_KEY -> client_id
- DAPI_SCOPE -> scope bei Login-Anfrage
- DAPI_TOKEN -> default access token
- DDEBUG_ALWAYS_LOGIN -> beim Start eingeloggt mit
   default access token
```

## C. Server einrichten

### C.1. Datenbank einrichten

- beide Datenbanken aus den Dumps `moduledata_dump.sql` und `userdata_dump.sql` mit MySQL erzeugen
- in der Nutzer-Datenbank folgendes Statement ausführen

```
INSERT INTO `rest_client`
VALUES (1, 'key-26hg02lsa',
    'secret-jg921tjg0', '.*',
    'http://localhost/processLogin');
```

- unter `studyplan_server/src/main/resources` die Dateien `moduledata.cfg.xml` und `userdata.cfg.xml` bearbeiten und in Zeile 11 die Datenbankverbindung anpassen (connection.url, connection.username und connection.password müssen geändert werden)

```
<!-- Database connection settings -->
<property name="connection.driver_class">
  com.mysql.jdbc.Driver
</property>
<property name="connection.url">
  jdbc:mysql://path/to/database
```

```
</property>
<property name="connection.username">username</property>
<property name="connection.password">password</property>
```

Listing 1: Auszug aus *data.cfg.xml

## C.2. Tomcat einrichten

- *Apache Tomcat 8.5* herunterladen oder direkt aus der ZIP-Datei entpacken (im letzteren Fall ist Tomcat bereits konfiguriert)
- ansonsten müssen folgende Dateien im Ordner `conf` von Tomcat angepasst werden, hier kann man sich an den Konfigurationen der fertig gepackten Variante orientieren:
  - `context.xml`: bei `<context>` sollte `reloadable` auf `true` gesetzt werden (dies ermöglicht einen Redeploy ohne Neustart des Servers)
  - `server.xml`: hier kann in Zeile 69 der Port eingestellt werden auf dem Tomcat läuft, in unserem Fall 9999
  - `tomcat-users.xml`: Hier muss die Role `student` und entsprechende Nutzer die dieser angehören hinzugefügt werden. Nur Nutzer die hier hinterlegt sind, können sich auf dem System einloggen.
    Dies ist das Äquivalent zu einem auf dem Shibboleth Identity Provider registrierten Nutzer, er ist berechtigt unsere Anwendung zu nutzen, hat aber zunächst kein Benutzerkonto.
- Wurde die fertig konfigurierte Installation verwendet, so sind bereits die Nutzer *max_mustermann* und *peter_schmidt* hinterlegt, beide mit dem Passwort `123`.

## C.3. Server kompilieren und ausführen

- Voraussetzung ist eine korrekt eingerichtete Installation von *Apache Maven*
- Über die Kommandozeile in das Verzeichnis `studplan_server` navigieren
- `mvn package` ausführen
- im Verzeichnis `target` findet sich die Datei `studyplan.war`. Diese muss nun in das `webapp`-Verzeichnis der Tomcat-Installation kopiert werden.
- der Server kann nun über das Skript `startup.sh` (bzw. `.bat`) im `bin`-Verzeichnis der Tomcat-Installation gestartet werden
- der Studyplan-Applikationsserver läuft nun unter `http://localhost:9999/studyplan`
- angehalten wird Tomcat über das `shutdown`-Skript im bin-Verzeichnis
- Alternativ kann der Server auch ohne lokale Tomcat-Installation über das Tomcat-Maven-Plugin gestartet werden. Hierfür muss im Verzeichnis `studyplan_server` der Befehl `mvn tomcat7:run` ausgeführt werden

# D. Bedienungsanleitung

StudyPlan ist eine Web-App, mit welcher ein Benutzer sein Studium einfach, schnell und sicher planen kann. Dafür muss er sich registrieren und ein paar Informationen bezüglich seines Studiums (Studienfach, Studienbeginn, bestandene Module...) eingeben, damit die richtigen Informationen für ihn geladen werden können.

Nach der Anmeldung wird er auf die Hauptseite weitergeleitet. Dort kann er seine Pläne verwalten (erstellen, duplizieren, als HTML-Dokument exportieren und löschen).

Will der Benutzer einen Plan erstellen, so klickt er auf den Erstellen-Button und gibt einen Namen ein. Die Planansicht-Seite ist eine übersichtliche Oberfläche mit einem Plan auf der linken Seite und einer Modul-Liste auf der Rechten.

Die bereits bestandenen Module bzw. vergangenen Semester sind schon im Plan eingebettet bzw. ausgefüllt. So kann der Benutzer neue Semester einfügen und Module mit Drag-and-Drop aus der Modul-Liste in den Semestern des Plans ablegen. Die ECTS-Anzahl der Semester und des ganzen Plans werden somit aktualisiert.

Die Modul-Liste stellt verschiedene Filter zur Verfügung, mit deren Hilfe man gewünschte Module einfach anhand verschiedener Kriterien finden kann. Dazu gehören der Modulname, der Turnus, die Veranstaltungsart, die Kategorie, das gewünschte ECTS-Intervall und, ob es sich um ein Pflicht- oder Wahlmodul handelt.

Der Benutzer hat auch die Möglichkeit, Module positiv oder negativ zu bewerten (die Bewertungen werden auch bei der Generierung berücksichtigt).
Klickt er auf den Überprüfen-Button, wird der derzeitige Plan anhand der Modul-Constraints und der Bereichs- und Rule-Group-Regeln verifiziert. Wenn Constraint-Verletzungen gefunden werden, werden diese in einem Dialogfenster angezeigt und die fehlerhaften Module rot umrahmt. Damit kann der Benutzer den Plan an die Vorgaben des Modulhandbuchs anpassen.

Der Vervollständigen-Button dient zur Vervollständigung eines bereits existierenden Plans mit sinnvollen Modulen. Klickt der Benutzer darauf, wird er auf den Generierungs-Wizard weitergeleitet. Dort kann er die Maximalanzahl an ECTS pro Semester sowie die Zielfunktion auswählen, anhand welcher der Plan optimiert werden soll. Wählt der Benutzer beispielsweise die Zielfunktion für eine minimale Semesteranzahl, wird der Plan so vervollständigt, dass er möglichst wenig Semester enthält.
Nach erfolgreicher Vervollständigung wird der neue Plan angezeigt und dem Nutzer angeboten, den Plan zu übernehmen, zu verwerfen, oder unter neuem Namen zu speichern. Dies kann der Nutzer in der rechten Seitenleiste auswählen.

Über die Kopfzeile der Seite kann der Benutzer jederzeit sein Profil bearbeiten, auf die Planliste zugreifen oder sich abmelden.