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

15. 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 ????? Zeilen Java-, Javascript-, HTML- und CSS-Code liegt jetzt eine fertige, nur noch zu optimierende Anwendung vor. In diesem Bericht möchten wir dabei auf implementierte und ausgelassen Features eingehen, Änderungen beschreiben und begründen, und die Implementierungsphase (einschließlich aller durchgemachten Nächte) reflektieren. Zusammenfassend ist dieser Bericht Ende einer arbeitsintensiven, aber erfolreichen Phase, die als Endprojekt ein einmalig sinnvolles, dringendnotweniges 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 gehört eine Mindestanzahl an ECTS, die ihn ihm belegt werden müssen, sowie 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 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 Modulanzahl beschränkt sind. Das heißt, dass innerhalb einer Gruppe eine Mindest- und eine Maximalanzahl an Modulen vorgeschrieben sind, um ein Studienfach erfolgreich abschließen zu können. Dies stellt zum Beispiel sicher, dass mindestens zwei Stammmodule sowie ein Proseminar belegt werden.

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: — Rückgabe: — (200 OK)

Methode	**URI**	**Beschreibung**	**Anfrage-Parameter /Rückgabewerte**
GET	/fields	Gibt alle Bereiche des zum Studenten gehörenden Studiengangs zurück	Anfrage: — 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

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.

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-Verbindungen wurden die Klassen SessionOpenFilter und SessionCloseFilter hinzugefügt, welche bei ankommender Anfrage bzw. vor abgeschickter Antwort eine Datenbank-Session erstellen 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 AuthorizationContexts 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 zur korrekten Einhaltung der JSON-Spezifikationen und ermöglichen reibungslose Umwandlung von bzw. zu ihren Modell-Gegenstücken.
Da der PDF-Export einer HTML-Generierung gewichen ist, wurde 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

3.2 Wunschkriterien

4 Review

5 Testfälle

Anhang