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

14.02.2017

Phase	**Verantwortlicher**
Pflichtenheft	Ali Bejhad
Entwurf	Janek Westfechtel
Implementierung	Clemens Naseband
Qualitätssicherung	Robin Berger
Präsentation	Jacques Huss

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument beschreibt die Implementierungsphase des PSE-Projekts „Studienplanung als Generierung von Workflows mit Compliance-Anforderungen: Planerstellung und Visualisierung“.

2 Kriterien

2.1 Musskriterien

Kriterium	Eingebaut?
Auslesen der Datenbank zu Beginn	✓
Mehrere Studienpläne pro Nutzer	✓
Generierung mit Constraints	✓
Berücksichtigung der Präferenzen	✓
Erbrachte Leistung berücksichtigen	✓
Änderungen verifizieren	✓
Studienpläne anzeigen	✓
Module werden angezeigt	✓
Unterscheidung der Module	✓

2.2 Wunschkriterien

Kriterium	Eingebaut?
Kurze Wartezeiten	(✓) kein Stresstest
Echtzeit aktualisieren	✓
Übungsschein	X
Exportieren	✓als PNG
Speichern der Einstellungen	(✓) ohne Präferenzen
Verschiedene Sprachen	(✓) fehlende Einträge
Weitere Optimierungen	X

3 Probleme und Änderungen am Entwurf

Hier werden alle Änderungen am Entwurf dokumentiert. Nicht erwähnte Klassen wurden nicht geändert. Die Klassen sind in alphabetischer Reihenfolge nach ihren Packages aufgeführt.

3.1 Model

3.1.1 studyplanning.model.data

Dies ist ein neues Package in der sich Klassen, die sich mit der Datenhaltung von Studiengänge befassen, befinden. Alle hierhin verschobenen Klassen werden die Verschiebung erwähnen.

3.1.2 Attribute

Dies ist eine neue Klasse, die wichtige Eigenschaften die ein Modul haben kann repräsentiert, hierzu gehören zur Zeit Pflichtmodule und Stammmodule. Dies ermöglicht eine viel dynamischere Verwendung des Generierungsalgorithmus, da man zur Einhaltung von Regeln des Studienganges, nur eine neue Instanz dieser Klasse erstellen muss. Diese Klasse befindet sich im neuen Package: *studyplanning.model.data*

- *public Attribute(String name, int required)* - Erstellt ein neues Attribut mit einem Namen und die Anzahl der Module die in jedem Studienplan eines Studienganges enthalten sein soll. (Bsp. 2 Stammmodule).
- *public String getName()* - Gibt den Namen des Attributes zurück.
- *public int getRequieredModule()* - Gibt die Anzahl der benötigten Module zurück.

3.1.3 ChoiceCategoryPreference

Die Klasse hat sich den Änderungen an der Superklasse ModulePreference angepasst und deren Methoden implementiert.

3.1.4 ChoiceModulePreference

Die Klasse hat sich den Änderungen an der Superklasse ModulePreference angepasst und deren Methoden implementiert. Bestehende Methoden haben sich nicht geändert.

3.1.5 ConstraintType

Folgende Methoden wurden entfernt:

- *public static ConstraintType valueOf(String name)* - Die Datenbank wurde so angepasst, dass man direkt über den Index auf den Constraint kommt und nicht mehr über den Namen.

- *public static ConstraintType[] values()* - Diese Methode war redundant da man über das Values Field auf alle Elemente zugreifen kann.

Folgende Methode wurde hinzugefügt:

- *public boolean isBiDirectional* - Diese Methode gibt an ob ein Constraint in beiden affektierten Module stehen sollte. (Bspw. braucht HM 1 nicht zu Wissen, dass es von HM2 benötigt wird)

3.1.6 DataIO

Im Verlauf der Implementierung ist aufgefallen, dass die Verwendung von UUID für Benutzer viel zu lang sind. Insbesondere deshalb wenn man bedenkt, dass man die Benutzer ID auch „leicht“ von Hand übertragen können soll. Dementsprechend wurden die Schnittstellen angepasst.

Folgende Methode wurde hinzugefügt:

- *public boolean deleteWorkflow(UUID id)* - Mit dieser Methode wird ein Workflow aus der Datenbank mit all seinen Referenzierungen gelöscht. Zurückgegeben wird ob das Löschen erfolgreich war.

- *public StudySubject getDataForStudySubject(String subject)* - Gibt den Studiengang mit dem angegebenen Index an, falls vorhanden.

- *public Collection<StudySubject> getAvailableSubjects()* - Gibt alle Studiengänge die das System ausgelesen hat.

3.1.7 DataSet

Durch die erweiterte Datenbank (siehe Email vom 11.01.2017) hat ein Studiengang für Bereiche und Vertiefungen eigene Klassen. Dementsprechend wurde die Klasse geändert. Diese Klasse befindet sich nun im neuen *Package studyplanning.model.data.*

Der Konstruktor hat sich geändert:

- *public DataSet(Iterable<SubjectSection> sections)* - Das DataSet enthält nun alle Bereiche in dem Konstruktor, da diese vorher ausgelesen werden müssen.

Folgende Methoden wurden entfernt:

- *public Collection<Module> getCompulsaryModules()* - Da verpflichtende Module nun ein Attribut ist, wird diese Methode nicht mehr benötigt.

- *public Collection<Module> getModulesInCategory(String category)* - Diese Methode wurde in *g*etSection(String) umbenannt. Da Bereiche nun eine eigene Klasse haben.

Folgende Methoden wurden hinzugefügt:

- *public Module getModule(String name)* - Gibt das Modul mit dem angegeben Namen zurück.

- *public SubjectSection getSection(String name)* - Gibt den Bereich mit dem angegebenen Namen zurück.

- *public SubjectSpecialization getSpecialization(String name)* - Gibt die Vertiefung mit dem angegeben Namen zurück.

- t*public Collection<SubjectSection> getSubjectSections()* - Gibt alle Bereiche zurück.

- *public Iterable<SubjectSpecialization> getSubjectSpecializations()* - Gibt alle Vertiefungen zurück.

3.1.8 IECTSCap

Da viele neue Klassen eine mindest- und maximale Anzahl an ECTS angeben muss, wurde entschieden ein Interface hinzuzufügen. Diese Klasse befindet sich im neuen Package

Package studyplanning.model.data.

Folgende Methoden wurden hinzugefügt:

- *boolean doesModuleCount(Module module)* - Gibt an, ob das gegebene Modul zur ECTS Grenze des Objekts hinzuzählt.
- *float getMaxAllowedECTS()* - Gibt die maximale Anzahl an ECTS, die erreicht werden darf.
- *float getRequiredECTS()* - Gibt die Mindestanzahl an ECTS an, die erreicht werden muss.
- *public static IECTSCap unionOf(Iterable<? extends IECTSCap> caps)* - Gibt ein neues IECTSCap Objekt zurück, das die Vereinigung der gegebenen IECTSCaps darstellt. Hierbei wird das Modul akzeptiert, wenn es von einem der gegebenen IECTSCaps akzeptiert wird und die minimale und maximale ECTS sind die minimale bzw. maximale ECTS der IECTSCaps zusammen addiert.

3.1.9 ModuleEvaluation → Evaluation

Die Klasse wurde in Evaluation umbenannt, da sie nun generische Evaluierung unterstützt. Sie übernimmt in diesem Fall die Bewertung von Modulen.

Die Klasse hat nun folgende Methoden:

- *public void addValue(K obj, int value)* - Erhöht den Wert der Instanz um den angegeben Wert.
- *public Iterator<K> getValuedObjects()* - Gibt einen Iterator in absteigender Reihenfolge der bewertenden Instanzen zurück.
- *public int getValue(K obj)* - Gibt den Wert der angegeben Instanz zurück.
- *public void initValues(Iterable<K> objs)* - Initialisiert die Evaluierung und gibt allen Instanzen den Wert 0.
- *public void setValue(K obj, int value)* - Setzt den Wert der Instanz auf den gegeben Wert.

3.1.10 MistakeAttribute

Durch die Einführung der Attribute, ist es nun möglich auch Fehler zu machen, die diese nicht vollständig berücksichtigen. Diese Klasse erbt von **Mistake** und implementiert deren Methoden.

Folgender Konstruktor ist hinzugefügt worden.

- *public MistakeAttribute(String localizationKey, Attribute attrib)* - Erschafft eine neue Instanz, falls beim Verifizieren ein Fehler auftritt.

Folgende Methoden sind hinzugekommen:

- *public String getLocalizedMessage(Locale language)* - Gibt eine Zeichenkette, die den genauen Fehler, in der gewählten Sprache ausgibt.

- *public Attribute getViolatedAttribute()* - Gibt das betroffenen Attribut zurück.

3.1.11 Mistakes

Im Verlauf der Implementierung sind noch einige andere Fehlermöglichkeiten aufgefallen, durch die geeignete Wahl in der Entwurfsphase mussten nur neue Methoden hinzugefügt werden.

Folgende Methoden wurden geändert:

- *public static Mistake getMissingECTS(SubjectSection subject)* - Hat nun einen Parameter, aus dem man herauslesen kann, in welchem Bereich ECTS fehlen.

- *public static Mistake getTooManyECTS(SubjectSection subject)* - Die gleiche Änderung wurde auch hier übernommen.

- *public static Mistake getMissingECTS(SubjectSection subject)* - Die gleiche Änderung wurde auch hier übernommen.

Folgende Methoden wurden hinzugefügt:

- *public static Mistake getAddModuleMistake(Module module)* - Wird verwendet, falls beim Hinzufügen eines Moduls ein Fehler entstanden ist.

- *public static Mistake getLoadingWorkflowFailedMistake(UUID WorkflowID)* - Wird verwendet, falls beim Laden eines Workflows ein Fehler entstanden ist.

- *public static Mistake getLoadingWorkflowsMistake()* - Wird verwendet, falls beim Laden der Workflows eines Benutzers ein Fehler entstanden ist.

- *public static Mistake getMistakeTooFewAttribs(Attribute attrib)* - Wird verwendet, falls ein Attribut nicht berücksichtigt wurde (bspw. Es fehlen die Stammmodule).

- *public static Mistake getRemoveModuleMistake(Module module)* - Wird verwendet, falls das Entfernen eines Moduls aus einem Workflow ein Fehler entstanden ist.

- *public static Mistake getWrongSemesterType(Module module)* - Wird verwendet, falls versucht wird eine Sommersemester Veranstaltung in ein Wintersemester oder andersherum einzufügen.

3.1.12 Module

Durch einige Anpassungen an der Logik musste auch diese Klasse an vielen Stellen geändert werden. Diese Klasse befindet sich nun im neuen *Package studyplanning.model.data*. Der Konstruktor wurde geändert.

- *public Module(String name, int id, SemesterType semester, float ects, Attribute... attribs)* - Das Modul nimmt jetzt auch die ID, die sie in der Datenbank hat entgegen um später leichter auf diese zuzugreifen. ECTS können auch Dezimalzahlen sein, weshalb der Datentyp angepasst wurde. Außerdem wird nicht mehr anhand eines boolean entschieden ob ein Modul verpflichtend ist, sondern ob er ein bestimmtes Attribut hat (bspw. Pflicht oder Stammmodul).

Folgende Methode wurde entfernt:

- *public boolean isCompulsory()* - Da diese Eigenschaft anhand von Attributen ablesbar ist, wurde diese Methode obsolet.

Folgende Methoden wurden hinzugefügt:

- *public void addConstraint(Constraint constraint)* - Fügt ein Constraint hinzu, sobald dieser aus der Datenbank ausgelesen wird.

- *public ImmutableCollection<Attribute> getAttributes()* - Gibt alle Attribute dieses Moduls zurück.

- *public int getID()* - Gibt die ID des Moduls zurück.
- *public SubjectSection getSection()* - Gibt den Bereich dieses Moduls zurück.
- *public SubjectSpecialization getSpecialization()* - Gibt die Vertiefung dieses Moduls zurück.
- *void setSection(SubjectSection section)* - Setzt den Bereich dieses Moduls.
- *void setSpecialization(SubjectSpecialization spec)* - Setzt die Vertiefung dieses Moduls.

3.1.13 ModulePreference

Die Klasse hatte sich in den Parametern angepasst durch die Änderungen von **Evaluation**.

Folgende Methoden wurden hinzugefügt:

- *public abstract ImmutableCollection<Module> getAffectedModules()* - Gibt eine Collection an Modulen, die von einer Präferenz betroffen sind, zurück.
- *public abstract SubjectSection getSection()* - Gibt die SubjectSection zurück, in der die Module dieser Preference sind.

3.1.14 ModuleWrapper

Die Klasse wurde in das Package *studyplanning.model.data* verschoben.

Folgende Methode wurde hinzugefügt:

- *public void setHasMistakes(boolean value)* - Setzt ob das Modul in dem Workflow einen Fehler hat.

3.1.15 Preferences

Der Konstruktor wurde geändert:

- *public Preferences(int preferredDuration, int preferredECTSPerSemester, int currentSemester, Collection<ModulePreference> preferences)* - Erhält Informationen über die Präferenzen des Benutzers, darunter:

Attribut Name	Aufgabe
preferredDuration	Die Länge des Studiums
preferredECTSPerSemester	Die Anzahl der ECTS pro Semester
currentSemester	Das Startsemester
preferences	Die modulgebundene Präferenzen

3.1.16 Semester

Folgende Methode hat sich geändert:

- *public float getECTSPoints(IECTSCap cap)* - Gibt die Summe aller ECTS Punke in diesem Semester zurück, die die gegebene IECTSCap akzeptiert.

Folgende Methode wurde hinzugefügt:

- *public int getNumber()* - Gibt zurück, um das wievielte Semester es sich in dem Workflow handelt.

3.1.17 SemesterType

Folgende Methode wurde entfernt:

- *public static SemesterType[] values()* - Die Methode war redundant, da man über ein public Field darauf zugreifen kann.

Folgende Methode wurde hinzugefügt:

- *public boolean matches(SemesterType type)* - Überprüft ob diese Instanz und die Übergebene im gleichen Semester stattfinden.

3.1.18 StudySubject

Die Klasse implementiert von von IECTSCap und deren Methoden. Folgende Methoden wurden hinzugefügt:

- *public void addAttribute(Attribute attrib)* - Fügt dem Studiengang das gegebene Attribut hinzu.

- *public void addSubjectSection(SubjectSection section)* - Fügt dem Studiengang den gegebenen Bereich hinzu.

- *public Attribute getAttribute(String name)* - Gibt das Attribut mit den angegeben Identifizierer zurück.

- *public Collection<Attribute> getAttributes()* - Gibt alle Attribute dieses Studienganges zurück.

- *public int getID()* - Gibt die ID, die der Studiengang in der Datenbank hat, zurück.

- *public Collection<SubjectSection> getSections()* - Gibt alle Bereiche dieses Studienganges zurück.

3.1.19 SubjectSection

Diese Klasse ist neu hinzugekommen und beschreibt einen Bereich eines Studienganges. Jeder Bereich besteht aus Spezialisierungen. Diese Klasse befindet sich in dem neuen Package studyplanning.model.data. Die Klasse implementiert von von IECTSCap und deren Methoden.

Folgende Methoden wurden eingefügt:

- *public void addSubjectSpecialization(SubjectSpecialization spec)* - Fügt dem Bereich eine Spezialisierung hinzu.

- Zusätzliche Getter wurden hinzugefügt:

Signatur	**Rückgabewert**
Iterable<Module> getModules()	Alle Module dieses Bereiches
String getName()	Der Name dieses Bereiches
SubjectSpecialization getSubjectSpecialization(String name)	Die angegebene Vertiefung
Collection<SubjectSpecialization> getSubjectSpecializations()	Alle Vertiefungen dieses Bereiches

3.1.20 SubjectSpecialization

Diese Klasse ist neu hinzugekommen und beschreibt eine Vertiefung, die Teile eines Bereiches sind. Diese Klasse befindet sich in dem neuen Package studyplanning.model.data.

Folgende Methoden wurden eingefügt:

- *public void addModule(Module mod)* - Fügt der Vertiefung das gegebene Modul zurück.
- *public Collection<Module> getModules()* - Gibt alle Module dieser Vertiefung zurück.
- *public String getName()* - Gibt den Namen der Vertiefung zurück.

3.1.21 WorkflowOperations

Hier wurden alle Parameter, die einen String für den Namen des Moduls entgegen nahmen, durch das Modul Objekt ersetzt.

3.1.22 WorkflowTasks

Folgende Methode wurde geändert:

- *public static Collection<Mistake> generate(Workflow workflow, StudySubject subject, Preferences prefs)* - Es wurde ein Parameter für die Präferenzen hinzugefügt.

3.2 View

3.2.1 HTMLBuilder

Die Klasse hat inzwischen nur eine Methode, die als Fabrikmethode agiert und andere Methoden aufruft. Dementsprechend wurden alle anderen Methoden auf *private* gesetzt. Folgende Methoden haben sich geändert:

- *public String printPage(StudySubject studySubject, Languages languages, String localeName, int userID, Workflow workflow, UUID workflowID, Collection<UUID>*

loadedWorkflows, Collection<StudySubject> subjects, Collection<Mistake> mistakes) - Diese Methode wurde hinzugefügt und agiert als Fabrikmethode, die andere Methoden aufruft.

3.2.2 ImageBuilder

Diese Klasse wurde neu hinzugefügt und dient als Hilfsklasse um Workflows als Bilder zu exportieren. Sie erbt genauso wie ViewBuilder von HTTPServlet und implementiert die abstrakten Methoden.

Folgende Methode wurde hinzugefügt:

- *public BufferedImage getImageFor(Workflow wf, Locale locale, boolean valid)* - Gibt ein generiertes Bild zurück.

3.2.3 JavaScriptBuilder

Die Klasse hat inzwischen nur eine Methode, die als Fabrikmethode agiert und andere Methoden aufruft. Dementsprechend wurden alle anderen Methoden auf *private* gesetzt.

Folgende Methode wurde hinzugefügt:

- *public String printJavaScript(int userID, UUID workflowID, String studySubjectName, Collection<UUID> loadedWorkflows)* - Fabrikmethode, die Java-Script erzeugt.

3.2.4 Languages

Folgende Methode wurde hinzugefügt:

- *public Locale[] getStoredLanguages()* - Gibt alle „Sprachen“ zurück.

3.2.5 ViewConfig

Die Klasse wurde neu hinzugefügt und sorgt für die Initialisierung des Programmes. Sie implementiert **javax.servlet.ServletContextListener** und deren Methoden. Folgende Getter wurden hinzugefügt:

Signatur	Rückgabewert
Controller getController()	Die Controller Instanz
static ViewConfig getInstance()	Die eigene Instanz
Languages getLanguages()	Gibt die Language Instanz zurück

3.3 Controller

3.3.1 controller.commands.iosubcommands

Das Paket wurde neu hinzugefügt. In dem Paket befinden sich die vom IOCommand aufrufbaren Subcommands.

3.3.2 Command

Jedes Command hat nun einen festgelegten Namen. Die Änderungen wurden in allen Command-Kind-Klassen auch übernommen.

Folgende Methoden wurden geändert:

- *public Command(String... name)* - Der Konstruktor nimmt nun einen Namen als Parameter entgegen.

- *public abstract Response execute(Map<String, String> input,Controller controller)* - Die execute Methode nimmt nun eine Map von String Schlüssel-Wert Paaren, und die ausführende Controller Instanz als Parameter entgegen.

Folgende Methoden wurden hinzugefügt:

- *public String getName()* - Gibt den Namen des Commands zurück

3.3.3 CommandException

Neu hinzugefügte Exception-Klasse, die im Falle eines Fehlers während der Ausführung der Commands geworfen wird.

Folgender Konstruktor wurde hinzugefügt:

- *public CommandException(String message)* - Erstellt eine neue CommandException mit einer gegebenen Message

3.3.4 CommandRegistry

Die CommandRegistry wurde neu hinzugefügt und koordiniert die Ausführung der einzelnen **Commands**.

Folgende Konstruktoren wurden hinzugefügt:

- *public CommandRegistry(String key)* - Erstellt eine neue Instanz, mit einem anegegbenen Schlüssel String

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> args, Controller controller)* - Nimmt die aufgespaltene Nachricht der View entgegen und führt das angegebene Command aus

- *public void registerCommand(Command command)* - Registriert ein neues Command

- *public Command getCommand(String name)* - Getter Methode für registrierte Commands

3.3.5 Controller

Dem Konstruktor des Controllers muss keine **ViewBuilder** Instanz mehr übergeben werden.

Folgende Methoden wurden hinzugefügt:

- *public getDataIO()* - Erlaubt die Übergabe der DataIO Schnittstelle

- *public getWorkflowOperations()* - Erlaubt die Übergabe der WorkflowOperations Schnittstelle

3.3.6 DuplicateWorkflowCommand

Die Klasse wurde neu hinzugefügt und ermöglicht es einen Workflow zu kopieren. Sie befindet sich im Package *controller.commands.iosubcommands*.

- *public DuplicateWorkflowCommand()* - Erstellt eine neue Instanz

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> input, Controller controller)* - Dupliziert den gegebenen Workflow

3.3.7 FlushWorkflowCommand

Die Klasse wurde neu hinzugefügt und ermöglicht es einen Workflow neu aufzusetzen. Somit muss der Nutzer nicht jedes Modul einzeln entfernen. Sie befindet sich im Package *controller.commands*. Folgende Konstruktoren wurden hinzugefügt:

- *public FlushWorkflowCommand()* - Erstellt eine neue Instanz

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> input, Controller controller)* - Ersetzt den Workflow durch einen neuen, leeren Workflow

3.3.8 InputParser

Alle Methoden, außer der ParseInput()-Methode, nehmen nun als Parameter eine String-Function aus Schlüssel-Werte Paaren entgegen.

Folgende Methoden wurden hinzugefügt:

- *public static Map<String, String> parseInput(String)* - Übersetzt den gegebenen String zu einer Map mit String Schlüssel Werte Paaren

Folgende Methoden wurden geändert:

- *public static int parseUserID()* - Rückgabewert von UUID zu Int geändert

- *public static Workflow parseWorkflow(Function<String, String> input, DataIO dataio)* - Nimmt eine **DataIO** Instanz entgegen

- *public static StudySubject parseStudySubject(Function<String, String> input, DataIO dataio)*

- *public static Preferences parsePreferences(Function<String, String> input, StudySubject studySubject)*

Folgende Methoden wurden entfernt:

- *public static Workflow parseWorkflow()*

3.3.9 Response

Die Response beinhaltet keinen HTML-Code mehr. Allerdings ist es der Response nun möglich die **UserID**, **WorkflowID** und eine Sammlung an **WorkflowIDs** zu speichern. Dies ist notwendig um dem Nutzer das speichern der Workflows zu ermöglichen. Der Konstruktor wurde den neuen Variablen entsprechend angepasst.

Folgende Methoden wurden hinzugefügt:

- *public UUID getWorkflowID()* - Getter Methode für die UUID des aktuellen Workflows

- *public int getUserID()* - Getter Methode für die ID des aktuellen Nutzers

- *public Collection<UUID> getWorkflows()* - Getter-Methode für eine Sammlung aller UUIDs der Workflows des aktuellen Nutzers

- *public void setWorkflows(Collection<UUID> workflows)* - Setter-Methode für eine Sammlung aller UUIDs der Workflows des aktuellen Nutzers

Folgende Methoden wurden geändert:

- *public Response(Workflow workflow, Collection<Mistake> mistakes, UUID workflowID, int userID)* - Erweiterung des Konstruktors mit den Parametern workflowID und userID

Folgende Methoden wurden entfernt:

- *public getHTMLCode()*

- *public Controller(Workflow,Collection,int)*

3.3.10 Util

Die Klasse wurde neu hinzugefügt und beinhaltet Funktionalitäten, die in keine der anderen Schnittstellen passen. Sie lädt config-Dateien und überprüft ob Debugging aktiviert ist. Debugging gilt als aktiviert wenn Assertions angeschaltet sind.

Die Klasse befindet sich im Package *controller*.

Folgende Methoden wurden hinzugefügt:

- *public static boolean debugModeEnabled()* - Überprüft ob Debugging aktiviert ist
- *public static Properties loadConfigFile(String name)* - Lädt die angegebe Config-Datei

3.3.11 IOCommand

Die Klasse wurde angepasst und beinhaltet nun eine eigene **CommandRegistry**. Die Funktionalität wurde grösstenteils in die neu hinzugefügten SubCommands ausgelagert.

Folgende Methoden wurden geändert:

- *public Response execute(Map<String, String> input, Controller controller)* - Führt nun das angegebene subCommand aus

3.3.12 MoveModuleCommand

Die Klasse erbt von Command, und wurde neu hinzugefügt um Module zwischen Semestern zu verschieben. Folgende Konstruktoren wurden hinzugefügt:

- *public MoveModuleCommand()* - Erstellt eine neue Instanz

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> input, Controller controller)* - Verschiebt ein Modul vom einen Semester ins andere.

3.3.13 GetWorkflowsCommand

Die Klasse wurde neu hinzugefügt, erbt von Command und befindet sich im iosubcommands-package. Sie wird verwendet um die UUIDs der vom Nutzer erstellten Workflows zu laden. Folgende Konstruktoren wurden hinzugefügt:

- *public GetWorkflowsCommand()* - Erstellt eine neue Instanz

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> input, Controller controller)* - Lädt die verschiedenen WorkflowIDs die einem Nutzer zugeordnet sind

3.3.14 LoadWorkflowCommand

Die Klasse wurde neu hinzugefügt, erbt von Command und befindet sich im iosubcommands-package. Sie wird verwendet um den Workflow eines Nutzers zu laden. Folgende Konstruktoren wurden hinzugefügt:

- *public LoadWorkflowCommand()* - Erstellt eine neue Instanz

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> input, Controller controller)* - Lädt den Workflow eines Nutzers

3.3.15 NewUserCommand

Die Klasse wurde neu hinzugefügt, erbt von Command und befindet sich im iosubcommands-package. Sie legt einen neuen Nutzer an und gibt dem Nutzer einen neuen leeren Workflow. Folgende Konstruktoren wurden hinzugefügt:

- *public NewUserCommand()* - Erstellt eine neue Instanz

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> input, Controller controller)* - Erstellt einen neuen Nutzer und einen neuen Workflow

3.3.16 NewWorkflowCommand

Die Klasse wurde neu hinzugefügt, erbt von Command und befindet sich im iosubcommands-package. Sie legt einen neuen Workflow an und speichert diesen. Folgende Konstruktoren wurden hinzugefügt:

- *public NewWorkflowCommand()* - Erstellt eine neue Instanz

Folgende Methoden wurden hinzugefügt:

- *public Response execute(Map<String, String> input, Controller controller)* - Erstellt einen neuen Workflow