# LV 4752 / 7363
# Web-Engineering/WBA
# Übung 04

Ein erstes Rails-Projekt

Sammeln erster Erfahrungen mit Rails

# Aufgabe

- Abgabe von Übungsaufgaben in Form von Dateien
  - **Wir realisieren den Gegenstand des ersten Meilensteins nun mit den Mitteln von Rails**
  - Dabei modellieren wir gründlicher und definieren geeignete „Ressourcen“
- Einschränkungen
  - Komplexität: Eine reale Anwendung benötigt mehr Funktionalität, als mit diesem Beispiel in zwei Praktikumseinheiten umsetzbar ist
  - Synchronizität mit dem Vorlesungsstoff: Auch ein kleines Vorhaben wie das vorliegende bedeutet gleichzeitigen Kontakt mit vielen Komponenten von Rails. Diese werden in der Vorlesung aber erst nach und nach behandelt, sodass diese Rails-Vorübung Vorgriffe enthält. Eigenständige Recherche ist daher erforderlich.
  - „Perfektion“: Ziel dieser Rails-Übung ist das Kennenlernen typischer Projektschritte und Entscheidungen. Diese Rails-Übung ist ein „Wegwerf-Projekt“, das Ihnen spätere „teure“ Irrtümer im eigentlichen Projekt ersparen soll – lernen Sie hier aus Fehlern; setzen Sie ggf. neu an. Perfektion ist (noch) fehl am Platz!

# Modelle

- Modelle zur Umsetzung
  - Modell „**student**“
    - Name („last_name“), Vorname („given_name“), E-Mail-Adresse („email“), Passwort („password“), Matrikelnummer („matnr“), Studiengang („program“)
  - Modell „**course**“
    - Titel der Lehrveranstaltung („title“)
    - LV-Nummer („lvnr“)
  - Modell „**context**“
    - Studiengang („program“), Semester („term“),
    - Referenzen: Lehrveranstaltung („course_id“)
  - Modell „**exercise**“
    - Aufgabennummer („ex_number“), Datei („file“), Dateityp („file_type“)
    - Referenzen: student_id, context_id
    - Virtuelles Attribut (nicht in DB!): „password“

# Modelle

- Vereinfachungen
  - Modelle „program“, „term“
    - Hier nicht ausmodelliert, i.a. aber erforderlich
    - Vereinfachung: Strings abfragen/speichern/hinterlegen
  - Zeitstempel
    - Behalten Sie die Voreinstellungen bei. Abgegebene Dateien erhalten dadurch automatisch einen Zeitstempel beim Anlegen und Ändern.

- Typisierung
  - Die meisten Felder sind von Typ „string“
  - Verwenden Sie „integer“ für matnr, lvnr und Referenzen (IDs)
  - Die abgegebene Datei sei vom Typ „binary“

# Modelle

- Migrationen
  - Modell „student“
    - Anlegen genügt – Daten werden interaktiv via Controller eingegeben
  - Modell „exercise“
    - Anlegen genügt – Daten werden interaktiv via Controller eingegeben
  - Modell „course“ (Woche 2!)
    - *Initial load* für mindestens zwei LVen per „fixture“ (aus YAML-Datei)
    - Beispiel-Datei nutzen!
  - Modell „context“ (Woche 2!)
    - *Erst nach Anlegen von „course“-Daten mit Inhalten befüllen!*
    - Es genügt ein Studiengang und ein Semester
    - *Initial load* für mindestens zwei LVen per „fixture“ (aus YAML-Datei)
    - Dabei ID-Referenzen für „course_id“ von Hand eintragen…

# Modelle

- Relationen (Woche 2!)
  - Modell „student“
    - has_many :exercises
  - Modell „course“
    - has_many :contexts
  - Modell „context“
    - belongs_to :course
    - has_many :exercises
  - Modell „exercise“
    - belongs_to :student
    - belongs_to :context

# Modelle

- Validierungen (Woche 2!)
  - Modell „student“
    - Alle Felder sind Pflichtfelder
    - Aufbau von Feld „email“ mit RegExp auf Plausibilität prüfen
  - Modell „exercise“
    - Alle Felder sind Pflichtfelder
    - Bedingung (1..15).include? ex_number.to_i erfüllt?
    - Bedingung „password“ == student.password erfüllt?

# Controller

- Benötigte Controller
  - Vorbemerkungen
    - Nicht jedes Modell benötigt einen voll ausgebauten Controller mit allen Views zur Erzeugung, Veränderung und Löschung von Modell-Objekten
    - Modelle für Stammdaten etwa können auch einmalig mit Daten geladen und später nur noch genutzt werden („initial load“). Auch manuelle Eingriffe über die Konsolen- oder Datenbankschnittstelle sind bei seltenem Bedarf ausreichend.
  - student_controller
    - Anlegen usw. von Studenten-Daten
  - exercise_controller (Abgabe-Funkt. erst in Woche 2!)
    - Dieser Controller regelt die eigentliche Abgabe

# Views

- Begrüßungsseite
  - Links zum „student“- und insbesondere zum „exercise“-Controller
  - Einfach statisch realisieren, als „index.html“ im Ordner „public“
- Views des „student_controller“
  - Am einfachsten gleich model, controller und views mit „generate scaffold“ erzeugen
  - Link zur Begrüßungsseite ergänzen (Rückweg)!
- Views des „exercise_controller“
  - Anlegen zunächst wie „student_controller“, incl. „Rückweg“
  - „new“ entspricht einer neuen Abgabe
    - Kontext per Auswahl-Liste realisieren
    - Student ebenfalls (i.a. nicht praktikabel – hier würde man z.B. die MatNr abfragen und im Controller oder gleich mit AJAX dazu die student_id suchen)
  - Passwort soll vom Controller gegen das im Datensatz vom „student“ überprüft werden

# Tests

- *Unit tests*
  - „student“: Anlegen mit lückenhaften Daten sollte scheitern
  - „exercise“: Gültige Referenzen? Datei vorhanden?

- *Functional tests*
  - Rails versteht unter „functional tests“ Tests von Controller-Funktionen
  - Wir sind in der Vorlesung noch nicht so weit, Controller-Logik zu verstehen, und werden daher die *functional tests* erst im eigentlichen Projekt realisieren

# Reihenfolge bei der Durchführung

- Legen Sie ein neues Rails-Projekt an

```
$ rails abgabe
… (Angaben über erzeugte/vorhandene Verzeichnisse/Dateien)
$ cd abgabe
$ script/server        # Web-Server auf Port 3000 starten
# Rufen Sie mit Ihrem Browser http://localhost:3000/ auf
# Folgen Sie dem Link „Rails API“

# Beenden Sie den Serverprozess wieder
$ <Ctrl-C>
```

- Erkunden Sie die entstandene Verzeichnis-Struktur
  - Hilfe: Rails API, File „README, Kap. „Description of Content“

# Woche 1

- Erzeugen Sie ein vollständiges „Gerüst“ für „student
  - Wechseln Sie wenn erforderlich in den Ordner „abgabe“, führen Sie aus:

```
$ script/generate scaffold student last_name:string (usw.)
```

- Überarbeiten Sie die generierte migration-Datei
  - Ergänzen Sie Spaltenattribute für Mussfelder und Defaults
    - Verwenden Sie 'AI-B' als Defaultwert für das Feld „course“
  - Ändern Sie bei Bedarf Tabellen-Eigenschaften

- Ausführung der Migration

```
$ rake db:migrate
```

- Ausführung der automatisch generierten (Dummy-)Tests

```
$ rake test
```

*# Functional test* scheitert, falls Mussfelder angelegt wurden und leer sind!

- Ein erster CRUD-Test für Studentendaten
  - Starten Sie wieder den Web Server

```
$ script/server
```

  - Rufen Sie folgenden URL auf: http://localhost:3000/students/
  - Legen Sie neue Studierenden-Daten an, insb. die der Team-Mitglieder
    - Was passiert, wenn Sie ein Mussfelder leer lassen?
    - Was ist aus der Default-Belegung geworden?
  - Testen Sie auch die Änderungsfunktion!
  - Legen Sie einen Testeintrag an, um ihn gleich wieder zu löschen

- Verfolgen der Vorgänge auf DB-Ebene
  - Starten Sie in einer separaten Shell eine Datenbank-Konsole

```
$ script/dbconsole
sqlite> .headers ON
sqlite> .tables        → schema_migrations students
```

- Verfolgen der Vorgänge auf DB-Ebene (Forts.)
  - Zeigen Sie die Inhalte von „students“ an

```
sqlite> select * from tables;
```

…

  - **Verfolgen Sie mit dieser Anzeige die Auswirkungen jedes Schrittes, den Sie per Web-Interface ausführen!**
  - Beenden Sie schließlich die DB-Shell;

```
sqlite> <CTRL-D>
$
```

- Manuelle Eingriffe via Konsole und ORM (Beispiel)

```
$script/console
Loading …
>> s = Student.find(1)  # Eintrag mit Id=1, Fehler sonst
>> s.email = "xy@no_such_domain.example"
>> s.save
# Neuen Datensatz anlegen:
>> s = Student.new
>> s.last_name = "Tester"
>> s.given_name = "Joe"
>> s.matnr = 100234
>> s.password = "secret"
>> s.save       # → true
>> exit
$
```

- Manuelle Massenänderungen via Konsole und ORM (Beispiel)

```
$script/console
Loading …
>> Student.find(:all) do |s|
>>   s.course = "AI-B"
>>   s.save
>> end
>> exit
$
```

- Führen Sie nun analoge Schritte zur Erzeugung des Modells „exercise“ aus
  - Behandeln Sie die Referenzen zunächst als einfache Integers

- Anlegen reiner Modelle, hier „course“ und „context“
  - Wechseln Sie wenn erforderlich in den Ordner „abgabe“, führen Sie aus:

```
$ script/generate model course title:string lvnr:integer
# Analog für „context“
```

  - Wiederholen Sie die Schritte zur Überarbeitung von „migrations“
  - Führen Sie die „migrations“ aus und anschließend die Tests

- Einrichtung der Relationen
  - Berücksichtigen Sie nun die Relationen zwischen den Tabellen Ihrer Modelle, indem Sie diese in den jeweiligen Klassen mittels „belongs_to“ bzw. „has_many“ deklarieren.
  - Beispiel aus dem Vorlesungskontext dazu:

```
class Book < ActiveRecord::Base
  has_many :authors
end
```

- Einrichtung von Fixtures
  - Entwickeln Sie Testdaten im YAML-Format für die folgenden Unit-Tests

- Aufbau erster Unit-Tests
  - Füllen Sie die Dummy-Dateien der Unit-Tests für alle 4 Modelle mit jeweils einigen Test-Methoden / Assertions an.
  - Testen Sie insbesondere die Validierungsbedingungen
    - Provozieren Sie Fehler, indem Sie bewusst fehlerhafte/unvollständige Testdaten anlegen und per assertion die dann zu erwartenden Fehler überprüfen!

- Anmerkungen
  - Die Zahl der Testdaten und Tests/Assertions ist hier noch nicht wichtig. Sie sollten zahlreich genug sein, um das Vorgehen kennen zu lernen
  - Im Projekt wird später „*test-driven development*“ erwartet, d.h. Sie sollten die Erwartungen an Ihre Modelle schon früh in Form von Assertions formulieren – idealerweise früher als die eigentliche Codierung der Funktionalität!

# Weitere Schritte

- Models
  - *Initial load* von „course“- und „context“-Fällen per Datenmigration
  - Herstellung von Relationen zwischen den Tabellen!
- Views
  - Sorgen Sie dafür, dass das Feld „password“ nur verdeckt eingegeben bzw. angezeigt wird. Hinweis: „password_field“
  - Einarbeitung der Konsequenzen der ergänzten Relationen
  - Layout-Anpassung: Navigationsbalken
- Controller
  - Funktionalität für das Abspeichern der erhaltenen Datei(en) – am Ende!
- Tests
  - Testdaten einrichten, *functional tests* zum Erfolg führen

… Wenn noch Zeit bleibt!

# Abnahme

- Es erfolgt keine Abgabe, aber eine Abnahme am Arbeitsplatz
  - Ihr Übungsprojekt soll auf einem Rechner der Linux-Cluster vorgeführt werden.

- Die Übung ist für Zweier-Teams ausgelegt
  - Bilden Sie nach Möglichkeit nun die Teams, die später auch das eigentliche Projekt gemeinsam bestreiten werden
  - Spätestens in 14 Tagen muss die Teambildung abgeschlossen sein!

- Frist
  - Am 3.12.08 beginnt die Arbeit am eigentlichen Projekt, daher soll diese Rails-Vorübung bis dahin abgeschlossen sein.
  - Die Abnahme kann erst eine Woche später erfolgen, da am 3.12. die Betreuung des Praktikums wegen der Verabschiedung von unserem Präsidenten ausfallen muss. Bitte nutzen Sie diese Woche bereits für Ihr eigentliches Projekt, nicht mehr für die Vorübung!