Software-Architektur

Blatt 3 (Abgabe)

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 11.07.2017 15:41

Für den Schein ist in Vierergruppen eine Multiplex-Kino-Verwaltung als Web-Applikation mit dem Play Framework in der Programmiersprache Scala erstellen. Abzugeben sind dazu im zur Verfügung gestellten Git-Repository der Milestone 1, der Milestone 2 und das Release 1.0 jeweils zu den in Moodle kommunizierten Terminen.

Zum Release 1.0 ist eine Präsentation vor der gesamten Praktikumsgruppe durchzuführen.

Innerhalb der Gruppe können Sie die Arbeit sinnvoll aufteilen. Stellen Sie dabei sicher, dass von allen Gruppenmitgliedern Commits im Repo zu finden sind. Auch wenn Sie gemeinsam am Code arbeiten, ist immer der Committer für den Code verantwortlich. Arbeiten Sie dann beispielsweise abwechselnd als Committer. Wer nichts oder nur sehr wenig committed hat, bekommt keinen Schein.

Es zählt das Verhältnis am Ende bei der Abgabe von Release 1.0.

Anforderungen

Allgemein

Zu allen Abgaben muss Folgendes erfüllt sein:

- Der Code muss üblichen Standards genügen. Dies wird vom Jenkins mit Hilfe von Scalastyle überprüft. Die von Scalastyle verwendete Config-Datei finden Sie unter `https://gist.github.com/obcode/bd2e623dfc41157d8cf5`.
- Beim Compilieren darf keine Warning ausgegeben werden. Aus dem Grund sind auf dem Jenkins Warnings Fehler. Durch hinzufügen von

```
scalacOptions ++= Seq("-unchecked", "-deprecation", "-feature",
    "-Xfatal-warnings")
```

zur `build.sbt` können Sie das auch bei sich machen.

- Der Code muss mit ScalaDoc kommentiert sein (wird von Jenkins automatisiert erstellt und deployed).
- Die Datei `README.txt` muss, mit Markdown formatiert, die App, Designentscheidungen sowie die Umsetzung der u.a. architekturellen Anforderungen beschreiben (wird von Jenkins mit Pandoc erstellt und deployed).
- Der Code muss zu einem wesentlichen Teil mit Specs2-Tests abgedeckt sein (werden von Jenkins ausgeführt). Die Testabdeckung wird mit Scoverage ermittelt und muss für alle selbst geschriebenen Klassen und Objekte mindestens 80% betragen.
- Das MVC-Pattern muss konsequent umgesetzt werden.

Im Folgenden werden immer nur minimale funktionale Anforderungen an die Anwendung beschrieben. Um die Architekturanforderungen umzusetzen, sind unter Umständen zusätzliche Funktionalitäten einzubauen.

Milestone 1 — Das Multiplex-Kino

Ein Multiplex-Kino enthält mehrere Kinosäle in denen verschiedene Filme laufen. Die Kinosäle sind von 1 beginnend fortlaufend nummeriert und haben eine feste Anzahl von Sitzplätzen. Diese sind nach dem folgenden Schema nummeriert: Sitzplatz dx mit $d \in \{A, B, C, ...\}$ und $x \in \{1, 2, 3, ...\}$, wobei der Buchstabe für die Reihe und die Zahl für den Sitz innerhalb der Reihe steht, also z.B. $A4$ heisst Reihe A, Sitz Nummer 4. Implementieren Sie die notwendige Businesslogik, so dass Kinosäle erzeugt werden können.

In jedem Kinosaal läuft jeweils ein Film für einen bestimmten Zeitraum zu festgelegten Zeiten. Implementieren Sie die Businesslogik für die Filme mit allen notwendigen Eigenschaften.

Alle Daten müssen in einer dateibasierten h2-Datenbank gehalten werden.

Legen Sie Beispieldaten an (das geht beispielsweise über SQL-Kommandos in den Evolution-Dateien, oder über das Global-Objekt) und implementieren Sie Tests (z.B. können in einem Saal nicht zwei Filme gleichzeitig laufen).

Als Website erstellen Sie eine Sicht auf die Filme die an einem auswählbaren Tag laufen.

Abgabe zum Termin (siehe Moodle) durch Taggen mit “Milestone1” im zur Verfügung gestellten Git-Repository.

Milestone 2 — Die Kunden wollen buchen

Benutzer können sich registrieren und einloggen. Benutzer sind Kunden oder Mitarbeiter. Überlegen Sie sich ein sinnvolles Verfahren mit dem sichergestellt wird, dass neue Mitarbeiter erzeugt werden können, aber kein Kunde sich selbst zum Mitarbeiter machen kann.

Kunden können für eine Vorstellung (mehrere) Karten buchen. Die Buchung kann online bis 30 Minuten vor der Vorstellung storniert werden. Um die Karten an der Kasse abholen zu können, wird von Ihrer Anwendung ein eindeutiger Buchungscode erzeugt. Diesen müssen die Kunden an der Kasse wissen.

Mitarbeiter können Filme über die Website einpflegen. Stellen Sie dabei sicher, dass keine Konflikte auftreten können. Ziel ist das so benutzerfreundlich wie möglich zu machen. Z.B. wäre es toll, wenn der Mitarbeiter in Abhängigkeit vom Datum nur Kinos auswählen kann, die frei sind.

Filme oder einzelne Vorstellungen können auch wieder gelöscht werden. Gibt es bereits Buchungen, müssen diese storniert werden. Die betroffenen Kunden müssen informiert werden (per E-Mail oder ausserhalb des Systems) und es wird Ihnen ein Gutschein in Höhe des Gesamtpreises der stornierten Karten (auch wenn diese noch gar nicht bezahlt worden sind) als kleine Wiedergutmachung in Ihrem Kundenkonto hinterlegt.

Mitarbeiter können auch Karten buchen.

Es sind mindestens 3 Pattern in der gesamten Anwendung **sinnvoll** umgesetzt und in der `README.txt` beschrieben. Sie können die Pattern auch in dem Bereich umsetzen den Sie vorher schon implementiert haben (also refaktorieren).

Abgabe zum Termin (siehe Moodle) durch Taggen mit “Milestone2” im zur Verfügung gestellten Git-Repository.

Release 1.0 — Die Vorbereitung für die Kassen

Am Kassensystem werden Buchungen abgerufen, bezahlt oder storniert, und es können direkt Karten gekauft werden. Eine Viertelstunde vor dem Beginn einer Vorstellung werden alle nicht bezahlten Buchungen storniert.

Es ist nicht notwendig sich am Kassensystem als Benutzer einzuloggen. Das Kassensystem hält keine eigenen Daten sondern kommuniziert mit der bisher implementierten Anwendung über eine REST-Schnittstelle.

Implementieren und dokumentieren Sie eine REST-Schnittstelle für Ihr System über die potentielle Kassensysteme zugreifen könnten. Achten Sie insbesondere auf eine lose Kopplung (Stichwort: Hypertext/Hypermedia).

Abgabe zum Termin (siehe Moodle) durch Taggen mit “1.0-RELEASE” im zur Verfügung gestellten Git-Repository.

Kurzpräsentation

Am Ende des Semesters müssen Sie Ihre WebApp der Praktikumsgruppe präsentieren. Sie haben dazu **10 Minuten** Zeit. Legen Sie den Schwerpunkt Ihrer Präsentation auf die verwendeten Pattern in Ihrer Implementierung. Führen Sie den Zugriff auf die REST-Schnittstelle auf dem Beamer vor.