Software Engineering I (IB)

Blatt 3

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 22.11.2018 12:15

Die beiden Aufgaben auf diesem Blatt müssen Sie für den Schein abgeben. Achten Sie insbesondere darauf, dass die beiden Aufgaben **unterschiedliche** Abgabetermine haben!

Aufgabe 1 — UML (Abgabe bis 14.11.2018, 12:00 Uhr)

Das Repository für diese Aufgabe bekommen Sie unter https://classroom.github.com/a/Q2FOIe5k.

UML-Diagramme können mit verschiedenen Mitteln erzeugt werden. Sie könnten ein einfaches Zeichenprogramm nehmen, Sie könnten per Hand malen und einscannen, Sie könnten verschiedene spezielle UML-Editoren nutzen.

Wir machen etwas anderes. Wir erzeugen die UML-Diagramme mit Hilfe von PlantUML aus einer textuellen Repräsentation. Das hat einige Vorteile:

- Sie müssen sich nicht um das Layout kümmern. Das macht PlantUML automatisch. Zwar nicht immer besonders schön, aber ausreichend für unsere Zwecke.
- Sie können Änderungen in der Commit-Historie sehen.
- Sie können die Diagramme in verschiedenen Formaten erzeugen (PNG, SVG, PDF, …)

Für WebStorm gibt es ein PlantUML-Plugin mit dem es sich sehr komfortabel arbeiten lässt. Während Sie die PlantUML-Datei editieren, wird es daneben gleich gerendert und zeigt, ganz leicht zeitversetzt, den aktuellen Stand als Bild.

Für die Abgaben der UML-Aufgaben gehen wir etwas anders vor als sonst. Sie bekommen ein GitHub-Repository in dem Sie, ohne weitere Unterverzeichnisse, Ihre PlantUML-Dateien direkt in den `master`-Branch committen. In dem Repository befindet sich neben einer `.gitignore`-Datei auch eine Datei für den Travis mit folgendem Inhalt:

```
os: osx

before_install:
  - brew update > /dev/null 2>&1
  - brew install plantuml
  - mkdir output

script:
  - plantuml -ooutput -tsvg *.puml

before_deploy:
  - git tag "$(TZ=Europe/Berlin date +'%Y-%m-%d,%H.%M.%S')-$(git log --format=%h -1)"

deploy:
  provider: releases
  api_key: $GITHUB_TOKEN
  file_glob: true
  file: output/*
  skip_cleanup: true
```

Diese `.travis.yml` installiert PlantUML und erzeugt automatisch SVG-Images zu allen Dateien mit der Endung `.puml` und legt diese in einem Unterverzeichnis `output` ab. Diese Dateien werden dann automatisch zu einem GitHub Release deployed.

Achtung: Damit das funktioniert, müssen Sie auf Travis-CI ein GitHub-Token hinterlegen.

Erzeugen Sie ein GitHub-Token unter https://github.com/settings/tokens. Wählen Sie dabei unter `Select scopes` den Oberpunkt `repo` aus. Gehen Sie anschließend auf die Travis CI Plattform um den GitHub-Token dort zu hinterlegen. Ersetzen Sie in der GitHub-URL Ihres Repositorys einfach `github` durch `travis-ci`, z.B.

`https://travis-ci.com/ob-swengiib-ws18/blatt-2-aufgabe-1-obcode`

Loggen Sie sich mit Ihrem GitHub-Account ein. Rechts unter `More Options` finden Sie den Punkt `Settings`. Tragen Sie im Abschnitt `Environment Variables` eine neue Variable ein mit dem Namen `GITHUB_TOKEN` und dem Wert Ihres GitHub-Tokens. `Display value in build log` muss dabei ausgeschaltet bleiben.

Wenn Sie ab jetzt in ihr Repository pushen, sollte der Travis automatisch die SVGs erzeugen und in Ihrem Repository unter `Releases` → `Assets` verfügbar machen. Der Tag,

den Travis dazu automatisch erzeugt hat, hat als Namen das Datum und die Uhrzeit des Commits. Als Abgabe gilt dann das letzte Release **vor** Ablauf der Abgabefrist. Erzeugen Sie zur Abgabe außerdem ein Issue mit dem Titel **Abgabe** und weisen Sie es Ihrem Tutor und mir zu!

Dieses Vorgehen gilt auch für alle folgenden UML-Abgaben.

Nun zur eigentlichen Aufgabe:

Unser Kunde will eine Verwaltung (Website) für ein Multiplex-Kino von uns implementieren lassen. Dazu hat er folgende Vorstellungen in einem ersten Gespräch geäußert. Da Sie nicht selbst dabei sein konnten, hat ihr Kollege in einer Strichliste für Sie alles mit notiert:

- Mehrere Kinos.
- Es laufen Vorstellungen an einem bestimmten Tag zu einer bestimmten Uhrzeit.
- Jeder Film kann auch mehrfach gezeigt werden.
- Ein Kunde soll auf der Website verschiedene Sichten haben.
- Einmal soll er nur alle verschiedenen Filme sehen.
- Dann soll er angezeigt bekommen, was an einem Tag alles läuft.
- Dann für einen Film, wann der läuft.
- Es gibt ein paar Kunden, die wollen nur wissen was in einem bestimmten Kino läuft, z.B. weil es eine sehr große Leinwand hat.
- Für eine bestimmte Vorstellung soll der Kunde dann eine Übersicht über alle Sitze bekommen. Dabei sieht er, welche Sitzplätze frei sind und welche reserviert.
- Dort kann er dann auch Sitzplätze auswählen, aber auch wieder abwählen.
- Die Mitarbeiter sollen das auch alles machen können, weil die kommen ja auch als Kunden in Frage ;-)
- Außerdem soll ein Mitarbeiter neue Filme und neue Vorstellungen einpflegen können.
- Manchmal müssen sie eine Vorstellung wieder herausstreichen. Da gehen die Reservierungen natürlich verloren.

Erstellen Sie aus diesen Informationen folgende Diagramme:

- Aktivitätsdiagramm(e)
- Anwendungsfalldiagramm(e)
- (Fachliches) Klassenmodell

Überlegen Sie sich genau, was sinnvollerweise in den Diagrammen stehen muss und wie. Eventuell gibt es noch Informationen die nicht genannt, aber klar sind. Oder es wurden Dinge genannt, die für die Diagramme gar nicht relevant sind. Versuchen Sie auch alles was Sie jetzt wissen oder was klar ist, so genau wie möglich in den Diagrammen darzustellen, z.B. durch Kardinalitäten, Erweiterungen, Beziehungen, … Aber “verkünsteln” Sie sich nicht zu sehr. Wichtig ist natürlich auch keine allzugroße Komplexität zu kreieren.

Aufgabe 2 — Angular App (Abgabe bis 05.12.2018, 12:00 Uhr)

> Im Rahmen der Praktika am 06.12.2018 müssen Sie Ihre Angular-App auf dem Beamer kurz (max. 5 Minuten) präsentieren.

Diese Aufgabe können Sie alleine oder zu zweit bearbeiten und abgeben. Über den GitHub-Classroom-Link https://classroom.github.com/g/FFIcj24L müssen Sie als erstes ein neues Team erstellen (als der/die Erste von beiden oder wenn Sie alleine arbeiten). Den Namen können Sie sich selbst aussuchen. Wenn Sie als ZweiteR einer Gruppe auf den Link gehen, müssen Sie dann dem bereits erzeugten Team beitreten. Der/die Erste hat bereits ein Repository bekommen, dem Sie dann beitreten werden.

Wenn Sie alleine arbeiten, dann gehen Sie jetzt nach dem Git Flow oder nach dem GitHub-Flow vor, zu zweit nach dem Git-Flow. Die Abgabe erfolgt in beiden Fällen als Pull-Request. In beiden Fällen sollten Sie, nachdem Sie Ihr Angular Projekt aufgesetzt haben, den Travis vom `develop`-Branch aus deployen lassen. Arbeiten Sie ansonsten analog zur Tour of Heroes. Deployen Sie Ihre AngularApp auch wieder mit Hilfe des Travis CI als GitHub Page in Ihr Repository und fügen Sie den Link auf der GitHub-Website Ihres Repositorys ein.

Die Angular-App die Sie implementieren sollen, dient zur Verwaltung eines Multiplex-Kinos. Es werden dabei Teile aus Aufgabe 1 umgesetzt.

Die Daten sollen von einem `InMemoryDbService` geliefert werden. Er liefert ein Array von `Screenings`. Ein Objekt vom Typ `Screening` hat eine **eindeutige** `id: number` und enthält einen `movie: Movie`, ein `date: Date` und ein `cinema: Cinema`. `Date` ist ein bereits existierender TypeScript-Datentyp. Ein Objekt vom Typ `Movie` besteht nur aus einem `title: string`. Für diese App können Sie davon ausgehen, dass der Filmtitel einen Film **eindeutig** identifiziert. Ein Objekt vom Typ `Cinema` enthält eine eindeutige `id: number` sowie ein zweidimensionales Array `seats: boolean[][]`. Die erste Dimension sind die Reihen, wobei die erste Reihe, mit dem Index `0`, direkt vor der Leinwand ist. Die zweite Dimension steht dann für den Sitz innerhalb der Reihe. Der boolsche Wert zeigt dann an, ob der Platz noch frei (`false`) oder bereits besetzt/reserviert (`true`) ist.

Definieren Sie in Ihrem `InMemoryDbService` mindestens 2 Kinos in denen mindestens 4 Filme über mehrere Tage verteilt mehrfach laufen. Geben Sie die Screenings bereits nach Datum/Zeit sortiert zurück. Sollten zwei Filme gleichzeitig laufen, sortieren Sie diese aufsteigend nach der Kino-Id.

Aus den Angular-Komponenten bekommen Sie die Daten über einen `ScreeningService`, der folgende Funktionen anbietet:

```
getScreening(id: number): Observable<Screening>
getScreenings(): Observable<Screening[]>
getScreeningsForCinema(id: number): Observable<Screening[]>
```

```
getScreeningsForDate(year: number, month: number, day: number): Observable<Screening[
getScreeningsWithTitle(title: string): Observable<Screening[]>
```

Mit dem `InMemoryDbService` und dem `date`-Feld im `Screening`-Objekt, gibt es eine Besonderheit. Statt einem `Date`-Objekt mit dem Sie gleich weiter arbeiten können wird eine `string`-Repräsentation davon übertragen. Mit folgendem Workaround im `ScreeningService` können Sie das `Date`-Objekt wieder bekommen:

```
getScreenings(): Observable<Screening[]> {
  return this.http.get<Screening[]>(this.screeningsUrl).pipe(
    map(screenings =>
      screenings.map(s => {
        s.date = new Date(s.date);
        return s;
      })));
}
```

Versuchen Sie den Code zu verstehen (oder lassen Sie ihn sich von den Tutoren oder mir erklären).

Die CinemaApp hat ein Routing-Modul mit folgenden Routen:

```
const routes: Routes = [
  {path: '', redirectTo: '/allMovies', pathMatch: 'full'},
  {path: 'allMovies', component: DashboardComponent},
  {path: 'allScreenings', component: ScreeningsComponent},
  {path: 'cinema/:id', component: CinemaComponent},
  {path: 'day/:year/:month/:day', component: DayComponent},
  {path: 'movie/:title', component: MovieComponent},
  {path: 'buy/:id', component: BuyComponent}
];
```

Die App hat als Navigation die beiden Links `/allMovies` und `/allScreenings`. Im Folgenden werden die verschiedenen Komponenten kurz beschrieben. Die Gestaltung ist Ihnen freigestellt.

`DashboardComponent`

Anzeige der Film-Titel der **verschiedenen** Filme in alphabetischer Reihenfolge. Klick auf den Film-Titel führt zur Komponente `MovieComponent`.

`ScreeningsComponent`

Anzeige aller `Screenings` in sortierter (zuerst nach Datum, dann Uhrzeit, dann Kino-Nummer) Reihenfolge. Zu jedem Screening werden angezeigt:

- Datum und Uhrzeit in der Form `11.12.2018, 20:00 Uhr`. Ein Klick darauf führt zur entsprechenden `DayComponent`.
- die Kino-Nummer in der Form `Kino 1`. Ein Klick darauf führt zur entsprechenden `CinemaComponent`.
- der Film-Titel. Ein Klick darauf führt zur entsprechenden `MovieComponent`.
- ein Link `Karten kaufen`, der zur entsprechenden `BuyComponent` führt.

`CinemaComponent`

Wenn es in dem Kino mit der `id` keine Filme gibt (weil es das Kino beispielsweise gar nicht gibt), soll beispielsweise angezeigt werden:

```
In Kino 9 laufen keine Filme bzw. es existiert nicht.
```

Anderenfalls sollen alle Filme in zeitlich sortierter Reihenfolge angezeigt werden, die in dem entsprechenden Kino laufen. Es sollen die selben Infos wie bei der `ScreeningsComponent`, bis auf das Kino, angezeigt werden und mit den selben Links hinterlegt werden. Die Kinonummer sollte nur in der Überschrift auftauchen.

`DayComponent`

Wenn an dem angegebenen Tag keine Filme laufen, soll z.B. folgendes angezeigt werden:

```
Das Kino ist am 23.12.2018 geschlossen.
```

Anderenfalls wie bei `ScreeningsComponent` allerdings nur mit der Uhrzeit, die nicht anklickbar ist. Schreiben Sie das Datum in die Überschrift.

`MovieComponent`

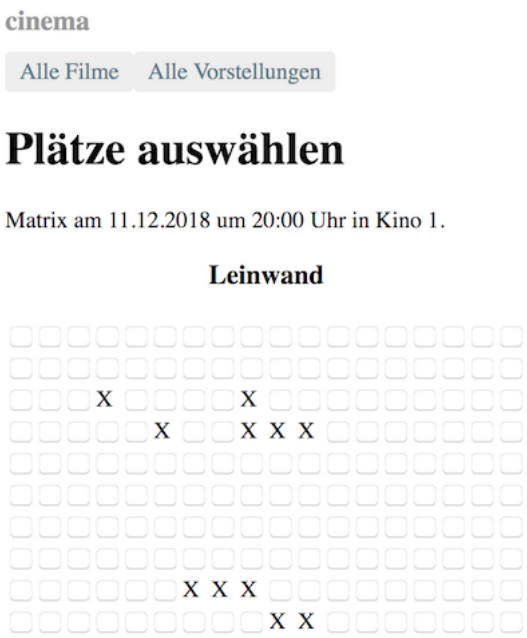
Werden zu dem Filmtitel keine Filme gefunden, soll z.B. ausgegeben werden:

```
Matrix 2 läuft zur Zeit nicht.
```

Anderenfalls die entsprechenden, anklickbaren Infos.

`BuyComponent`

Die `id` im Pfad der `BuyComponent` bezeichnet die eindeutige `id` eines Screenings. Zu dieser Vorstellung sollen alle üblichen Infos dargestellt werden. Außerdem sollen die Sitzplätze reihenweise angezeigt werden, z.B. wie auf dem folgenden Screenshot zu sehen ist.



Dabei soll unterschieden werden ob der Sitzplatz frei oder besetzt ist. Ein freier Sitzplatz kann z.B. durch einen Button, ein besetzter durch ein `X` dargestellt werden. Das können Sie aber auch wieder anders lösen.

In beiden Fällen ist jeder Platz anklickbar und wechselt über eine Methode von frei zu besetzt bzw. durch eine weitere Methode von besetzt zu frei. Implementieren Sie das so, dass sich die Ansicht dann gleich entsprechend ändert.

Außerdem soll unter den Sitzen die Liste der derzeit reservierten Sitze angezeigt werden. Sind keine Sitze reserviert, wird gar nichts angezeigt. Sobald ein Sitz ausgewählt wird, wird dieser angezeigt. Wird er wieder als frei markiert, verschwindet er wieder aus der Anzeige. Die angezeigten Sitze müssen nicht sortiert werden. Hängen Sie den zuletzt reservierten einfach hinten an.