Oblig 4

Start oppgave

Forfall 10. mars av 23.59 **Poeng** 1 **Må leveres** en filopplasting

Alle oppgavene skal være forsøkt løst etter beste evne (du *må* ikke løse bonusoppgavene, men det ligger en del lærdom i disse, så det oppfordres). Hvis du har en oppgave som du har forsøkt på, men ikke fått helt til, er det OK om denne kommenteres ut slik at vi kan se hvordan du har tenkt/hva du har forsøkt på. Resterende kode bør kunne kompileres og kjøres uten at det kræsjer. Oppgaver som er gjennomført og fungerer burde *ikke* kommenteres ut.

Les: Obligatoriske oppgaver - Informasjon (https://hiof.instructure.com/courses/7058/pages/obligatoriske-oppgaver-informasjon)

Obligen er strukturert opp i oppgaver, dere skal **levere hele IntelliJ prosjektet**.

Teorispørsmålene: Ikke .zip denne sammen med programmeringen, men levér som en egen fil. Leveres som docx/pdf/txt navngitt: Oblig4_JAVALIN_<DittNavn>.<filtype>
Eks: Oblig4_JAVALIN_LarsEmilKnudsen.pdf

Programeringsoppgavene: Leveres som en .zip fil av javaprosjektet navngitt:
Oblig4_JAVALIN_<DittNavn>.zip
Eks: Oblig4_JAVALIN_LarsEmilKnudsen.zip

Teori

Oppgave 1 - Ord og begreper

Lag deg en oversikt over hva følgende ord/begreper/teknologier betyr/er:

- Javalin
- Vue.js
- Anonym Klasse
- MVC (konseptet, og hver enkelt del)
  - Model
  - View
  - Controller

Oppgave 2 - Kodesammenligning

Gå sammen to og to (en du ikke har samarbeidet med). Ta for dere forrige oblig og forklar deres implementasjon.

Hva har dere gjort? Hvorfor har dere gjort det slik? Hva er forskjellig? Skriv et lite avsnitt om refleksjoner og funn.

Skriv hvem dere har gått sammen med, men skriv hver deres tekst.

Til de som *ikke* finner noen å gå sammen med, skriv opp navn og e-post i skjemaet her (https://hiof-my.sharepoint.com/:x:/g/personal/larseknu_hiof_no/EWL-EfRB3fNMsqLJKU-octoB0qHOJeDHbcRfwtEoJmwj5g?e=wKyHZm) . Den neste som skriver seg på, skriver seg opp i kolonne to, og tar kontakt for å avtale tid med den første.

Programmering

Vi skal fortsette med å utvide oppgaven vi lagde i Oblig 3. Du kan fortsette på din egen implementasjon, eller du kan starte fra løsningsforslaget som ligger her: Oblig3_ProposedSolution.zip (https://hiof.instructure.com/courses/7058/files/1285430?wrap=1) (https://hiof.instructure.com/courses/7058/files/1285430/download?download_frd=1)

Vi skal nå se gjøre noen mindre forbedringer av koden vi har, i form av å gjøre noe abstrakt, samt implementere interface for å kunne sortere dataene våre på en fornuftig måte.

Deretter skal vi begynne på implementasjon av et webgrensesnitt til applikasjonen vår.

Du står fritt til å ta bort det som er av kode i Main.java (du kan altså begynne med "blanke ark" hvis du ønsker). Vi skal uansett hovedsakelig bruke Application.java.

Oppgave 2.1 - Abstraksjon

Det er klasser vi har laget som det ikke er naturlig å lage konkrete objekter av. Hvilke er dette? Gjør så disse klassene abstrakte. (Det er i utgangspunktet bare én klasse, og det kan hende du allerede har gjort den abstrakt?)

Test at alt fortsatt fungerer etter dette.

Oppgave 2.2 - Sammenligning og sortering

Det er ofte naturlig å kunne sortere samlinger av objekter. Lag en naturlig sortering ved å implementere Comparable interfacet i klassene TvSerie og Episode. Du velger selv hva du ser på som en naturlig sortering.

Test at du kan sortere listen med episoder du har i en av tvseriene vi tidligere har opprettet.

Grensesnitt

Dere skal nå begynne å lage kode tilpasset et webgrensesnitt. Til dette skal dere benytte rammeverket Javalin. Dere vil få ferdig front-end i form av .vue filer, og skal tilpasse back-end koden til denne. Dere kan tilpasse/style front end'en hvis dere ønsker, men dette er ikke nødvendig.

Generell utfordring:

Siden dere i vil får en ferdig front-end, og ikke blir satt til å lage .vue filer selv, så vil det ikke nødvendigvis være en én-én match mellom navnene dere har på instansvariabler i modellklassene deres (teknisk sett get-

metodene), og navnene på dataene front-enden forventer at er der.

Det er lagt til en del feilhåndtering i front-end som vil gi en indikasjon på hva som er feil (slik at man bare ikke ender opp med en blank side når noe feiler). Det er ikke slik at dere får løst alle disse før mot slutten av alle oppgavene.

Hva skal dere da gjøre når noe er feil her?

Dere har to valg:

- Endre get-metodene deres til å matche dataene som front-enden leverer
- Kjøre en search-replace i vue-filene, slik at dataen frond-enden forventer, matcher de dere har i modell-filene deres
  - Disse er beskrevet i toppen av hver vue-fil, så det er bare å gjøre det systematisk

Oppgave 2.3 - Javalin og Maven - Nytt prosjekt

Først ønsker vi at vi skal kunne bygge prosjektet vårt ved hjelp av Maven. Den enkleste måten å gjøre dette på er å lage ett nytt prosjekt og velger "Maven" som "Build system". I Maven så setter vi opp de biblioteken vi ønsker å benytte i prosjektet vårt, så legg til disse i pom.xml. I dette tilfellet gjelder det javalin med tilhørende biblioteker.

Utdrag fra pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<!-- Litt annen informasjon er tatt bort for å gjøre det litt kortere -->

    <!-- Dependencies du skal legge til: -->
    <dependencies>
  <dependency>
    <groupId>io.javalin</groupId>
    <artifactId>javalin</artifactId>
    <version>5.3.2</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>2.0.6</version>
  </dependency>
  <dependency>
    <groupId>org.webjars.npm</groupId>
    <artifactId>vue</artifactId>
    <version>3.2.47</version>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.14.2</version>
  </dependency>
  <dependency>
```

```
      <groupId>com.fasterxml.jackson.datatype</groupId>
      <artifactId>jackson-datatype-jsr310</artifactId>
      <version>2.14.2</version>
    </dependency>
  </dependencies>
</project>
```

Lag en fil Application.java og lag en HelloWorld versjon ved hjelp av Javalin.

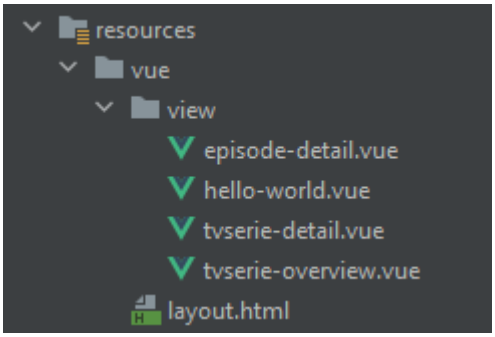
Bygg og kjør prosjektet.

Oppgave 2.4 - Forberedelser

a) Lag en pakke kalt "model", og kopier inn alle .java-filene alle fra det gamle prosjektet over i det nye maven-prosjektet (unntatt Main.java, pass på at pakkenavnene i filene som kopieres over blir korrekte).

b) Kopier inn mappen "vue"-mappen som ligger ved her, inn under mappen "resources". Dette er front-end koden du skal lage en back-end kode i Java til.

vue.zip (https://hiof.instructure.com/courses/7058/files/1284753?wrap=1) (https://hiof.instructure.com/courses/7058/files/1284753/download?download_frd=1)



Se at du fortsatt kan kompilere prosjektet.

Oppgave 2.5 - Repository

Når vi henter data i en back-end, er dette ofte fra en database. Slik det er nå, har vi ingen database vi benytter, men vi ønsker fortsatt et lite abstraksjonslag mellom businesslogikken i programmet vårt, og hentingen av data.

Lag et *interface* som heter **TvSerieRepository.java**, her vil de metodene som er aktuelle for uthenting av data defineres. Lag i tillegg en *klasse* som heter **TvSerieDataRepository.java**, som *implementerer* interfacet. Denne skal per nå opprette og holde på dataene vi benytte i applikasjonen vår. La denne filen ha en instansvariabel som er en ArrayList med TvSerier.

I konstruktøren oppretter du minst én TvSerie med noen episoder, og legger til i denne listen.

Definer get-metoder i interfacet og lag selve implementasjonen i klassen (vi skal utvide med flere metoder her senere):

- Hente alle tvserier

- Hente en spesifik tvserie

Oppgave 2.6 - TvSerier

I frontend har vi forskjellige views som kan vise data på forskjellige måter. Vi starter med de som omhandler tvserier, som er:

- /tvserie - Lister ut TvSerier
- /tvserie/{tvserie-id}/sesong/{sesong-nr} - Viser en spesifik tvserie (vi tar også med sesong her, fordi vi senere bare skal vise episoder fra én sesong av gangen)

a) Koble opp views

For å sette opp hvilke sider som skal vises når vi går til en spesiell URL-path, så må vi konfigurere dette ved hjelp av Javalin. Sett opp get-path'er i Application.java til å peke på disse Vue-komponentene.

b) Lage Controller

Koble opp view er ikke nok til å få data til front-enden, til dette må vi sette opp et API som gjør dataene vi har tilgjengelig i et .json format.

Front-end'en er nå satt opp til å spørre API'et via visse URL'er:

- **api**/tvserie - Gir en liste med TvSerier i .json format
- **api**/tvserie/{tvserie-id} - Gir en TvSerie .json format

For å få dette til å fungere må vi ha to elementer på plass - en definisjon av hvert av disse endepunktene, og en Controller som bestemmer hva som skal skje ved ett kall til dette (henter korrekt data og sender det til front-enden).

Lag en TvSerieController (gjerne i en passende pakke), med to metoder. En for å hente alle tvserier, og en for å hente en spesifikk. TvSerieController trenger også en referanse til TvSerieRepository.

c) Koble opp API og Controller

Sett opp get-path'er til å peke på disse API-punktene, og kalle korrekte metoder i TvSerieController.

Oppgave 2.7 - Episode

Gjør det samme som du gjorde i forrige oppgave for å også kunne vise episoder. Du må:

- Utvide TvSerieRepository med metoder for å hente alle episoder, og én enkelt.
- Lage en EpisodeController og metoder for å hente alle episoder, og én enkelt.
- Koble opp URL-punktet til *viewet*:
  - /tvserie/{tvserie-id}/sesong/{sesong-nr}/episode/{episode-nr} - Viser en spesifik episode
- Koble opp **API** URL-punktene til å kalle EpisodeController:
  - /**api**/tvserie/{tvserie-id}/sesdfong/{sesong-nr} - Henter alle episoder i en gitt sesong i .json format
  - /**api**/tvserie/{tvserie-id}/sesdfong/{sesong-nr}/episode/{episode-nr} - Henter én episode i .json format

Oppgave 2.8 - Bilder

Vi ønsker å kunne vise bilder for hver av episode og tvseriene vi viser i frontend'en. Utvid modellene Produksjon og TvSerie med instansvariabelen "bildeUrl". Gjør de nødvendige endringene i konstruktørene nedover i hierarkiet for å kunne sette denne verdien.

Legg til noen bildereferanser når du lager objektene i TvSerie*Data*Repository.

Oppgave 2.9 - Sortering

Vi ønsker å kunne sortere episodene. Legg til nødvendig funksjonalitet i EpisodeController for å kunne sortere når det hentes en liste med episoder.

Du kan få tak i sorteringsparameteren ved hjelp av:

context.queryParam("sortering");

Sorteringsparameterne som er tilgjengelig og skal kunne sorteres på er:

- ikke definert (blir vist i den rekkefølgen episodene ble laget i)
- *episodeNr*
- *tittel*
- *spilletid*

Bonusoppgaver

Bonusoppgave 3.1

Lek med interfacet og vue-filene. Hvordan endrer du hvilken data som vises? Kan du endre stilen på visningen?