IT2810 – Assignment 3 – Group 10

A description of our application can be found under “Overview of the website”. We have chosen to focus on Angular 2.0, and reasoning for this can be found in “The choice between Angular 2.0 and React”. Therefore the Angular 2.0 code is in the master branch of the project. In the master branch the newest, running code is located. The code we delivered in assignment 3.2 is located in the React and Angular branches. The other branches are used while we code different parts of the application, and there might be outdated code in these. The website can be found on http://it2810-10.idi.ntnu.no.

Run project: How to pack and run with webpack

1. Clone the project >

```
git clone https://bitbucket.org/trondaal/it2810-10-oppgave-3.git
```

2. Change to the project folder >

```
cd it2810-10-oppgave-3
```

3. Run `npm install` in the root directory. This will install the necessary dependencies required for webpack

4. Run `npm start` to build the production pack for the project. This will build the dist/ folder and run the project from there. This will run the server with forever js, which runs the node server “forever”.

5. Access the website through http://localhost:80/

6. Because forever.js causes the server to run “forever”, it is required to kill the processes to stop it. This can be done as follows:

    Windows: From the task manager by stopping all node processes.

    Linux: From the terminal run “sudo ps -aux | grep node” to find the id of the node processes and then “sudo kill -9 <id>” to kill it.

Note about the facebook login:

Because we now have delivered the project, and launched the page to it2810-10.idi.ntnu.no, we also had to turn off developer mode in the facebook application. This also meant that we had to change the home page url of the app to it2810-10.idi.ntnu.no. Because of this, the login won’t work on localhost. A fix for this could be to create your own facebook app, and

just replace our app id with yours, and set the homepage to localhost. The login functionality can be seen to work on the server: it2810-10.idi.ntnu.no.

Note about running the project:
For some reason when building the app with webpack, it won't allow us to have moduleId: module.id, but needs us to have "module.id" (a string instead). The string resulted in a crash when we tried it locally, so the first option wouldn't work, but the webpack option worked. Because of the extreme performance boost of webpack we chose to stick with that version.

Supported browsers:

- Chrome
- Firefox

Angular 2.0 default project files:
The files "package.json", "tsconfig.json", "typings.json" og "systemjs.config.js" describes how the application is supposed to be configured, for example how it should look for files and how the files are supposed to be compiled.

The "index.html" file will be the first file the browser will look for when the application is run. This file will then run the "app.module.ts"-file that is the main module to the application. By giving access to the next classes in the file, the "app.module.ts"-file is the file that will get the rest of the application up and running.

Naming conventions in Angular 2.0:
The naming conventions in Angular 2.0 makes it easy to find what a file is supposed to be used for, and we have chosen to follow these.

.component:
The files that includes ".component." is components or files that supports a component, that we would see or use in the finished application.

.module:
The files that includes ".module." is the ones that are responsible for defining which modules we include from the Angular framework and how we use them.

The file “app-routing.module.ts” lets us describe how to use Routing from the Angular framework, while the file “app.module.ts” includes an overview of the components and other modules the application will need to run as expected.

Folder structure:

To be able to differentiate between the client side and the server side of the application, all client side files are placed in the client folder. The server side files are outside the client folder.

```
| client/
|   | app/
|   |   | common/
|   |   |   | …
|   |   | home/
|   |   |   | …
|   |   | movie-review/
|   |   |   | …
|   |   | movie-search/
|   |   |   | …
|   |   | navbar/
|   |   |   | …
|   |   | profile/
|   |   |   | …
| config/
|   | helpers.js
|   | webpack.common.js
|   | webpack.prod.js
|   | webpack.test.js
| dist/ (after build)
|   | app.[hash].css
|   | app.[hash].css.map
|   | app.[hash].js
|   | app.[hash].js.map
|   | index.html
|   | polyfills.[hash].js
```

| | polyfills.[hash].js.map
| | vendor.[hash].js
| | vendor.[hash].js.map
| node_modules/
| app.js
| database.db
| dbHandler.js
| dbInit.js
| package.json
| README.md
| routes.js
| systemjs.config.js
| tsconfig.json
| typings.json
| webpack.config.js

client folder:
Inside the client folder you find an “app” folder, which contains the different components, services and providers that are needed for the application. The different component groups have their own folders. Therefore we have the common file auth.guard.ts inside the common folder, and everything that has to do with the home page in the home folder, movie-reviews in the movie-review folder, movie search in movie-search folder, navigation bar at the top of the page in the navbar folder and profile page in the profile folder. These are the files that are served to the user (client).

config folder:
The config folder has 4 config files for webpack. These are files required to pack the project with webpack. One could have made it into one file, but we felt it was cleaner with separate files. The way it is now makes it very easy to build a webpack.dev.js file, which will use the webpack.common.js file and build further on it. The webpack.prod.js file is the main file used for packing the application for production, and it uses the common and the helper file to do so.

dist folder:
The dist folder has the distribution files, which are the packed files from webpack. They consist of the app files, index.html, vendor files, and polyfill files. The polyfills file imports

core-js functionality required, while the vendor file is for importing other frameworks like bootstrap, jquery and rxjs.
This is also the folder which the app.js file gets the app files to serve (through the new index.html file).

Database files

The database files are separated into two different files: “db.init.js” and “db.handler.js”. The first file is responsible for initializing the database: create tables with attributes and add some dummy data. The handler file is used to handle different database transactions like adding a user to the database or getting reviews from it.

Facebook API

fb.component.ts inside the “\app\navbar” folder is a component that handles call to the Facebook API. The button has a (click)-functionality that calls login and logout, and gets userdata from the API. At the moment we get name, email and image url.

Choosing SQLite:

SQLite is a fast open source database. SQLite was chosen because it offers very easy setup and connection and since it’s an internal database it should perform faster than other alternatives. Almost all major frameworks/languages have native support for this database, and we felt that it would be too tedious to use time figuring out how to configure i.e MySQL or Oracle. Since our application will not scale to support millions of requests a full blown RDBMS is not necessary. Our group also had previous experience using SQLite which was a plus.

Choosing Webpack

The “packing tool” most talked about was webpack, and it also had most documentation, which is why we chose it to pack our angular files. We needed webpack to pack the files, or we would’ve had 30 seconds to 2 minutes to serve the web page to the user. And these loading times would then happen again if we were to refresh the web page. With webpack we got it down to “normal” times of mere seconds to serve the web page initially.

The choice between Angular 2.0 and React

To be able to choose between Angular 2.0 and React, we first listed all the positive and negative information we had about each technology.

Positives about Angular 2.0:

- It is seems to be straightforward to use, especially when you look at routing because this is described well in the toturial
- It looks complete, and does not look like we would need to look for extra libraries to be able to complete our task.
- As a group with limited time to finish the task, to choose Angular 2.0 instead of React might be a smart move, because in React we would need discussions about which extra libraries we would want to use.
- It seems precise, because it is only one way to declare a component if we continue to use TypeScript
- Good documented
- Good toturial

Negatives about Angular 2.0:

- There are many files and imports we would need to keep an overview over
- The whole group need to learn TypeScript and Angular specific syntax if we choose to continue to use this
- There has been cases where group members have experienced that the application won't start because of a syntax error in the code, and where Angular have not made an alert for it
- It could be difficult to find answers to new problems because the framework is new. It is possible no one else have had the same problem before.

Positives about React:

- Freedom of choice: we can choose what libraries we want to include. They are also pretty easy to find if you already know what kind of library you need.
- The application will only need the libraries we want to use
- Good documentation
- JSX makes the code easy to read for those who have used html and javascript earlier - something everyone in this subject should know
- Usually provides good and accurate error messages. You do not need to look for the error yourself.
- Easy to find someone else on the internet with the same problem, and with an answer that we can use.

Negatives about React:

- We would need to find the libraries we would need to complete the application we want to make - this will add time to discuss these choices in the group

- We would need to discuss how we want to write the code, or the code will become messy. - It exists more than one way to declare a component.
- More confusing tutorial than the one Angular have.
- If the application is going to be used later, we would need to be careful to install updates from the libraries we use. We would also need to check if they will work in the same ways, so that they do not suddenly do something different than what we expect.
- It could be difficult to find answers to new problems in the newest libraries we might use. It is possible no one else have had the same problem before.

Conclusion:
We choose Angular 2.0, because we think that might be a waste of time to discuss which libraries we want to add to React, even though this might gives us more freedom of choice. Also, our experience at this time is that it is simpler to implement the facebook-API in Angular 2.0 than in React.

We have also discussed the tutorials, and find the one in Angular 2.0 to be easier to understand. This might make it easy to learn Angular, even though it won't be as easy as to read JSX (for those who already know HTML and JavaScript already). Another thing we have discussed is that we think that Angular gives enough feedback on errors that this won't be a problem, even though we think React is a bit better and faster to give the feedback.

Implementation of the requirements in this task:

- We have implemented an application of the type "catalog" by storing movies.
- We use node.js on the server side, and the webpage is now up and running on http://it2810-10.idi.ntnu.no/
- Our application uses Angular 2.0.
- We have chosen to use SQLite as a database. And we have demonstrated both writing and reading to the database from the web application by inserting users, and reading movie and review information.
- A search in the database is done by letting the user search for movies he/she wants to read about.

- The user interface can show elements in a list based way, and include a way to see more details of each element in the list. This is implemented by just showing the pictures and titles of the movies in a search result, and letting you click on the picture to get to a page where the movies is shown in more detail. An expand button to show more details of a list element is also included for the reviews for a movie.
- The list based view can be sorted on different properties. This is implemented for reviews of a movie. When you enter the detailed information of a movie, you can see the reviews, and choose to sort the list by name or rating.
- The list based view can be filtered on different properties. This is implemented for reviews of a movie. When you enter the detailed information of a movie, you can see the reviews, and choose to filter the views by rating or name.
- The list based view is dynamically loaded. This is implemented in reviews of a movie. When the user scrolls down to the bottom a number of reviews are loaded underneath the previous ones.
- The application has a “my page” functionality. This is implemented by letting the user log into the application, and then on the profile page let the user see which reviews he/she have written to different movies.
- The application has session-handling. This is implemented in Angular 2.0, by checking if the user is logged in before letting the user navigate to a page where you have to be logged on to get to. If the user is not logged into the application, the user is redirected to the home page.
- Session is also implemented to handle recently viewed movies.
- The application has a “fancy” alternative to a view of the list. This is implemented by showing the rating of a given movie in a graph.

Things we would have added or fixed if we had more time

- If a user is logged into facebook before the database is created, the user won’t be added to the database if the user has already approved the application.
- Fix so reviews that are written to a movie are added to the list of reviews, without needing to reload the page.
- More user / profile page functionality:
    - Add followers to users
    - Show user pages of other users
    - Add description on the profile page of a user
- Add possibility to edit reviews

- Fix so a review only can be added with the user and movie existing

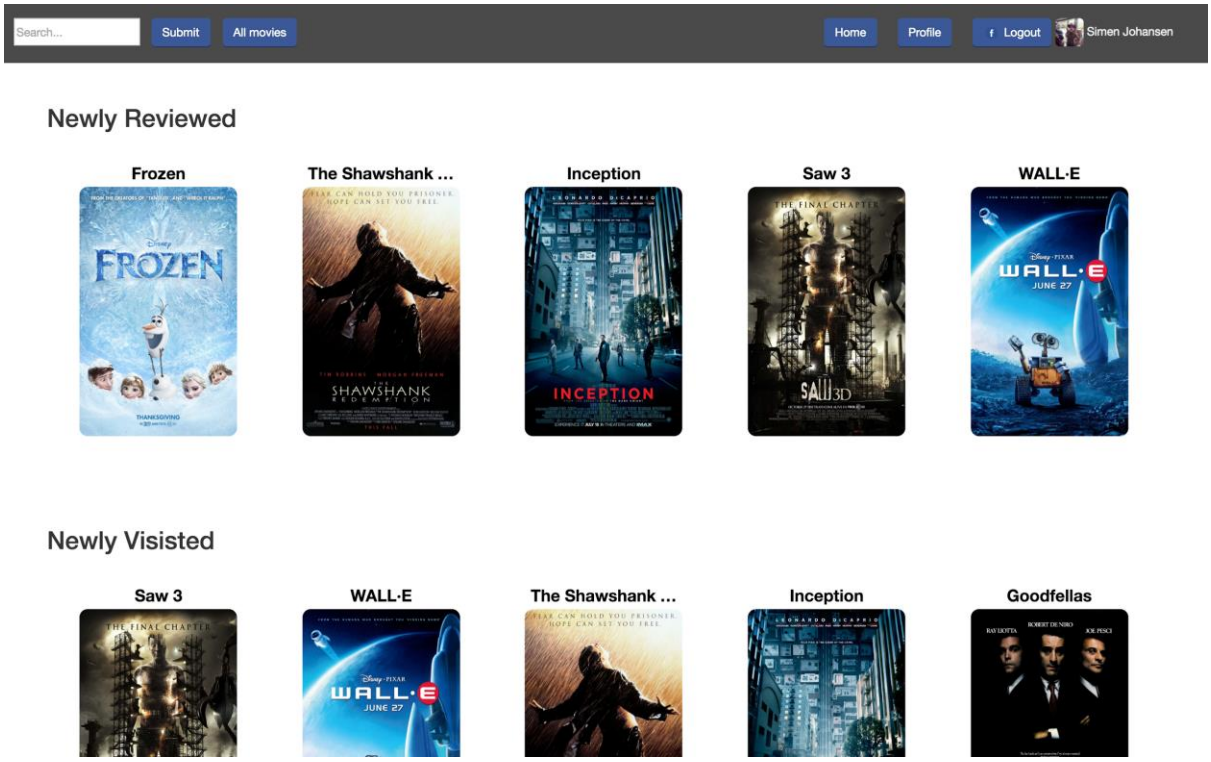
For some reason when running the app initially, when it has to initialize and create the database, it will get the SQLITE_LOCKED error and will crash. It only happens sometimes, so it wasn't a high priority for the project. We know it is caused by having two accesses to the database in the init phase, and sometimes the first transaction has not completed before the second tries to connect.

-

Overview of the website

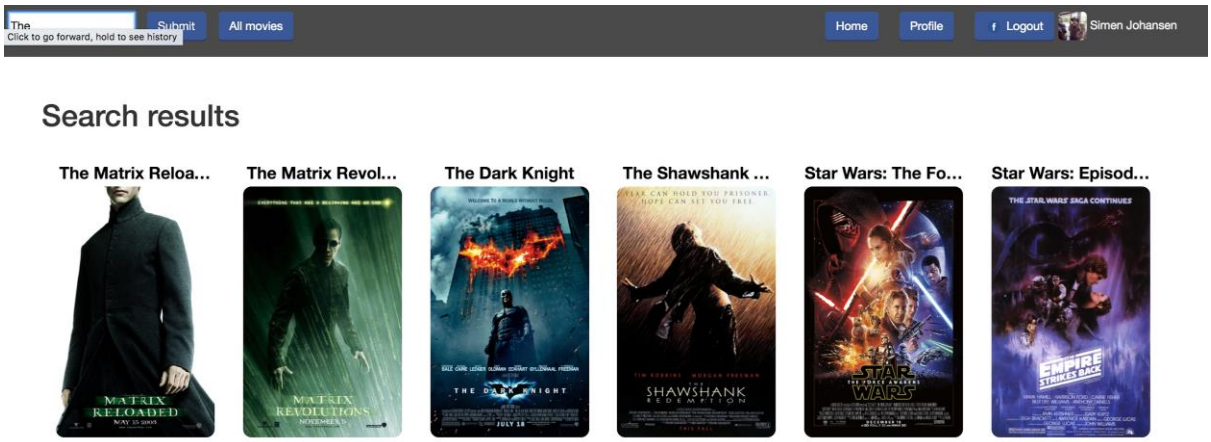
1. Home-page

When you go to <insert url from server>/home, you will get a list of newly reviewed movies as well as recent movies you have visited. If it's the first time you access the page, newly visited movies will be empty.



2. Search

It is possible to search for a movie by entering a search term in the search field, followed by either pressing the enter-key or submit-button.



This will bring you to the search results page where you can select the movie you wanted to get more details about.

3. Movie details

Once you have entered a specific movie, you will be presented with more information, the option to write a review (if you're logged in) and see reviews written by other users.

The Matrix Revolutions

Year: 2003

Description: The human city of Zion defends itself against the massive invasion of the machines as Neo fights to end the war at another front while also opposing the rogue Agent Smith.

Review form

Review title

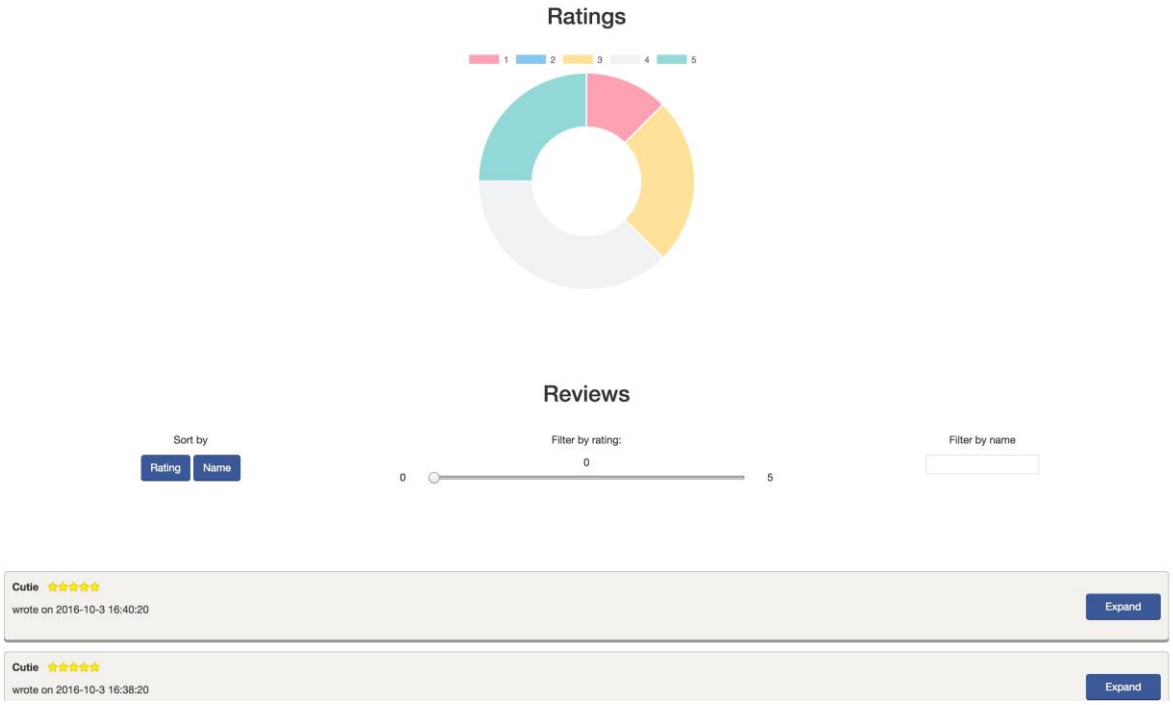
Rating

0

Review text

Submit | New Review

If you scroll further down you will see the following:

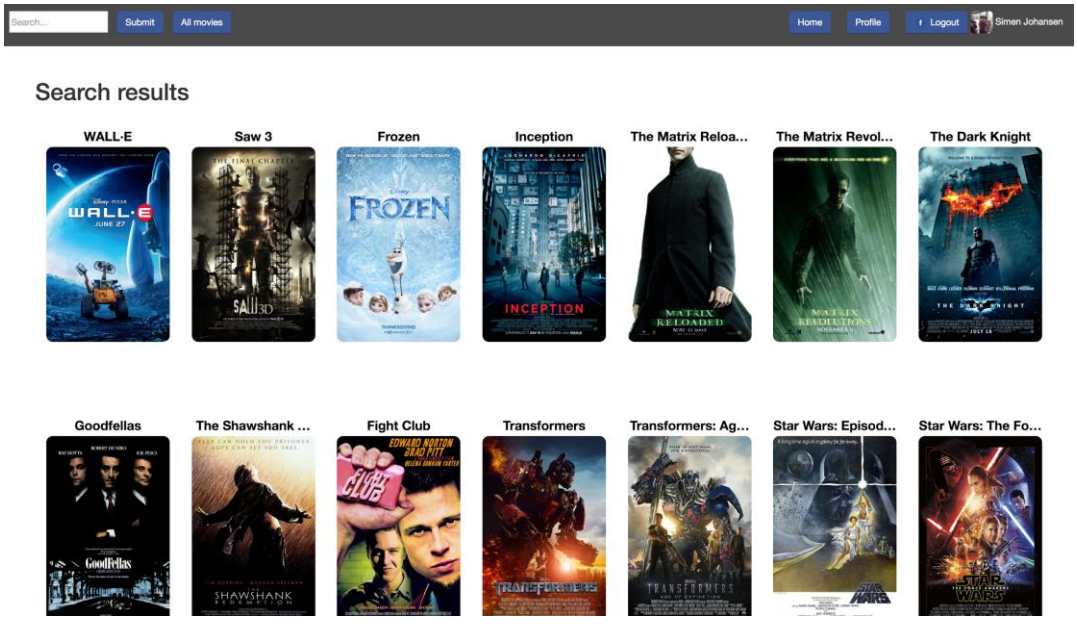


Here we have a diagram indicating how many ratings of different values are submitted. You will also see user reviews which you can sort by rating or name. If the slider is adjusted the visible reviews will be filtered by rating score. The page initially loads only two reviews but is expanded as you scroll further down (if there are more available reviews), loading two reviews each time. When the user reaches the bottom of the page a call to the database is

fired to load the next reviews (i.e all reviews are not fetched from the database at the same time).
A review contains a title, rating, date, and the review text itself. If you press the expand button, all registered information about that review will be displayed.

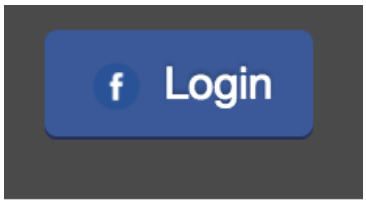
4. All movies

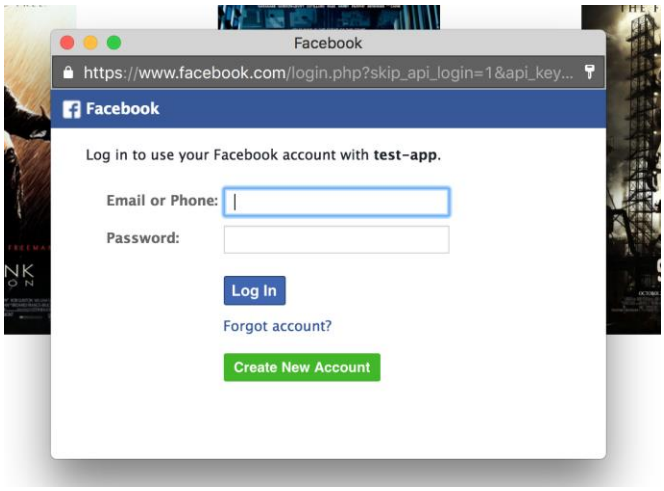
It is possible to get a list of all registered movies by clicking “All movies”.



5. Register / Login

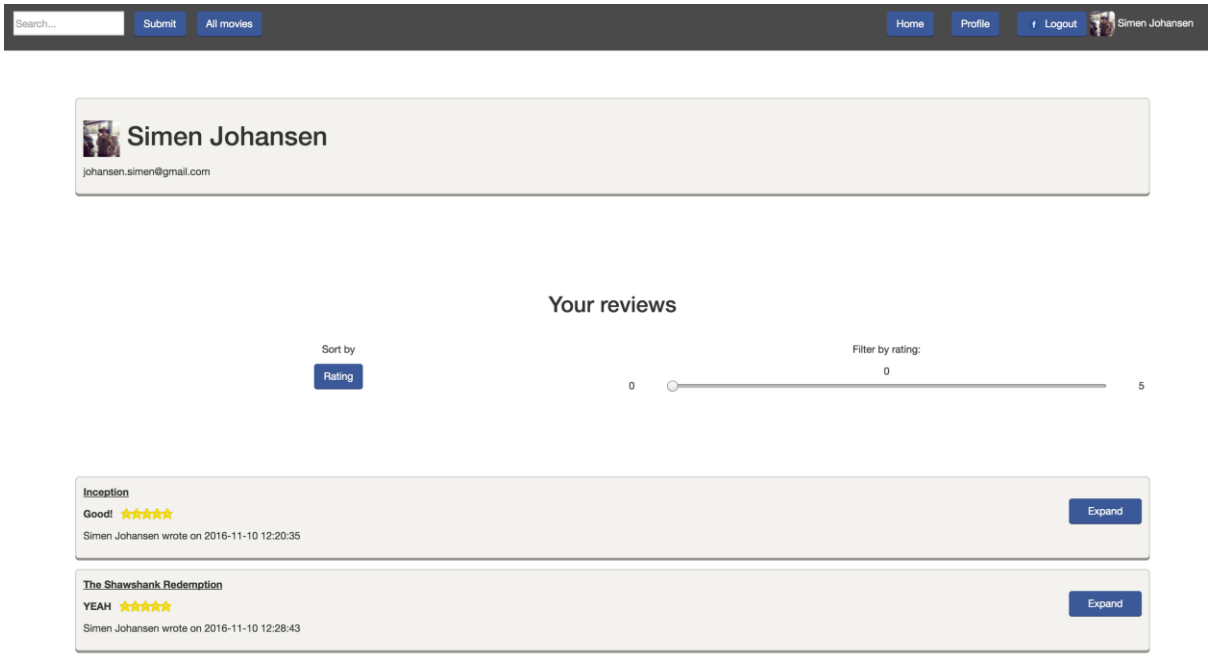
To be able to access the profile page and write reveiws, the user must authenticate / login through facebook. A dialog will open if you press “Login”, redirecting you to facebook’s login-window.





6. Profile page

Once logged in you will be redirected to your profile page. Here you get an overview of the information we have stored about the user as well as committed reviews (if any).


Search...
Submit
All movies
Home
Profile
Logout
Simen Johansen
Simen Johansen
johansen.simen@gmail.com
Your reviews
Sort by
Rating
Filter by rating:
0
0
5
Inception
Good! ★★★★★
Simen Johansen wrote on 2016-11-10 12:20:35
Expand
The Shawshank Redemption
YEAH ★★★★★
Simen Johansen wrote on 2016-11-10 12:28:43
Expand