Requirements documentation

- **The web-application**
  The project runs on the group's virtual machine and uses node.js on the server-side. We started developing the web-application in Angular version 4, but we jumped over to version 5, when it came out. We also used the user interface library UIkit for developing our project.
- **Backend and database**
  Out nodejs/express server is made to be highly modular and expandable with functionality mostly diveden in its own files. We chose PostgreSQL as our database of choice, with Sequelize as our ORM (Object Relational Mapper). This is because it makes relational queries very easy, and lets us do things fast. We started with using MongoDB with Mongoose, but found out our needs were better suited with a SQL database.
- **Writing and reading to the database and search**
  Our web-application demonstrate both writing and reading to the database. We have chosen to use TMDB as our API. By extracting data from here, we have implemented dynamic user-defined selection by letting users search through the data. The backend caches all used movie and series objects to the database to not tax the TMDb api too much.
- **List based view**
  Our list based view is implemented both in “library and “watchlist” where the user can see the movies and series they have added in different ways. The user will have the option to choose between a less detailed list view and a more detailed “card” view. So the user can search in the less details list view and see all their added movies and series. They can also get more details by clicking one of the items in the list, and the list will expand. If the user clicks on a card, they will also be able to see the item in a separate window with more details.
    - **Sorting**
      The list based view can also be sortes on “name”, “rating” and “date”. By clicking one of the buttons the user can choose between descending and ascending order for the items to be shown.
    - **Filtering**
      The list can also be filtered on movies and series. If a user search for something in the list, they can choose between searching for movies and series. The user also have the option of filtering by popularity.
    - **Dynamic data loading**
      Dynamic data loading is handled with pages. This is a feature supported by our database, and works really well with how PostgreSQL queries are done with Sequelize. Our services store page numbers and loads dynamically from the API seemlessly. We mostly use pages of size 10, and if your library or watchlist has more than 10 elements a 'load more' button will appear.
- **"My page" functionality**
  Our “my page” is the profile-page for the users when they are logged in. A user can log in and will then automatically end up here, where the user can see their last visits and more. This is described in detail over.
- **"Session" handling**
  Our sessions are handled by express, passport and cookies. We use hte Passport library to handle Oauth 2 login with Google.

- **Alternative view**
  Our alternative “fancy” view is a word cloud of genres of the movies and series that you have watched the most (library), and genres of the movies and series you want to watch (watchlist). We have also implemented the word cloud view on the “not found” page.
- **Testing**
  For testing we are using Karma with Jasmine. All tests for our components are running.