ReactJS

Goal

The goal of this lab is to get familiar with building a basic application using React.

The Mission

In this lab you will be building a movie editor using React. The application will let you browse through the list of known movies and select one to edit. Once you have made changes to a movie you have the option of saving your changes or canceling them.

There is no Begin application, you will use CREATE-REACT-APPLICATION to create the basic application skeleton. You will also need to add an http server containing a REST service at the endpoint http://localhost:3001/api/movies.

Please note that the first time you start this will take some time as number of NPM packages will download. This requires an internet connection where the NPM registry is not blocked by a firewall.

Note: The FINISHED folder contains a finished version of this lab as a reference.

Type it out by hand?

Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now.

Assignment 1

Create a new application using CREATE-REACT-APP.

First make sure you have NodeJS and NPM installed and then make sure you have the NPM create-react-app package installed as a global module using NPM INSTALL CREATE-REACT-APP --GLOBAL.

Create the new application using CREATE-REACT-APP BEGIN. This will create the new application in the BEGIN folder. Once this is done you can make BEGIN the default folder.

In order to add the API server you need to do a few additional steps.

1. Copy the DB folder into the BEGIN folder.

2. Install the extra required NPM packages using:
   npm install json-server concurrently
3. Update the start script in the package.json to both run the API server and the React application using:
   "start": " concurrently \"npm run start:db\" \"react-scripts start\" "
4. Add the following script to the scripts section:
   "start:db": "json-server ./db/movies.json --routes ./db/routes.json --port 3001"
5. Add the following line to the package.json to forward all API requests:
   "proxy": "http://localhost:3001"

You can now start the application using *npm start* or *yarn start*. This should open the browser at http://localhost:3000/ displaying the new React application.

Open **APP.JS** in the **BEGIN/SRC** folder and update the *React* component named App. In its *render()* function you can return a *div* element with a containing *h1* header element and a page title.

Open **INDEX.JS** and observe that *ReactDOM* is used to render the *App* component into the div element with id *root*. This element is already located in the **PUBLIC/INDEX.HTML**.

Make sure your application renders without errors in the browser. You should see the *title* element.

Assignment 2

Create a file **MOVIELIST.JS** and create a new *React* component named *MovieList*. Add some code in the *componentDidMount()* lifecycle function to do an AJAX request. This request will load the list of movies. You can do this using the *fetch()* function and the **/API/MOVIES** REST endpoint. Save the loaded movies in the components state.

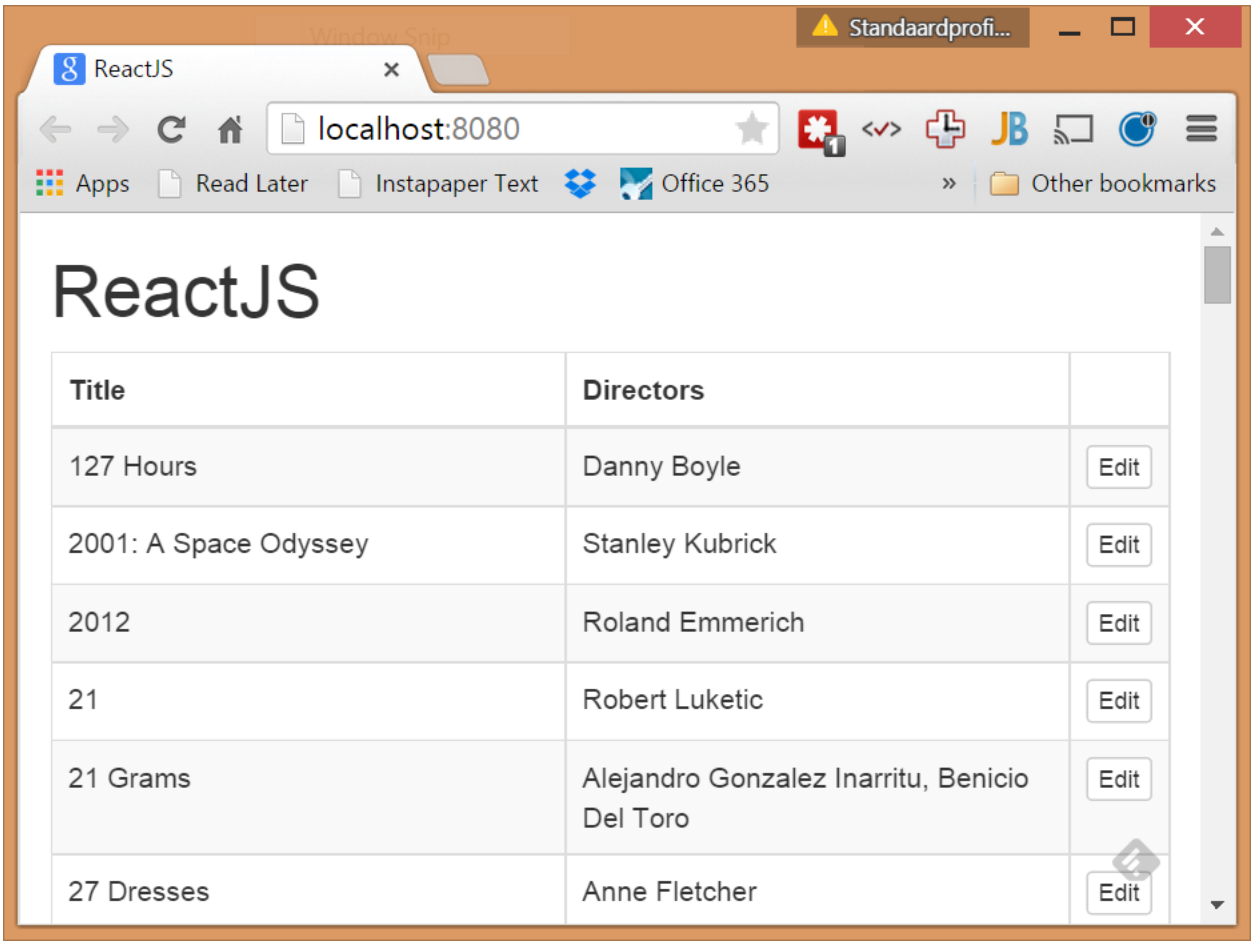
Install the Bootstrap 4 npm package using: **NPM INSTALL BOOTSTRAP** and import the main **BOOTSTRAP.CSS** into the App.js. This file can be located in **./NODE_MODULES/BOOTSTRAP/DIST/CSS/**. Replace the CSS class *App* with *container* in **APP.JS**

Code the component *render()* function to show a table with all the movies loaded. Add the *table table-bordered table-striped* classes to use the Bootstrap styling.

Create a new **MOVIEROW** component and use it to display the movie title and the directors in a column each as shown below. Add a third column with an edit button that will let the user edit the movie in that row. For now the button doesn't need to do anything yet.

Update the *App* component to render the *MovieList* component.

Make sure your application renders without errors in the browser and you can see the list of moves.



Assignment 3

Create a new **MOVIEEDIT.JS** file and add a basic *MovieEdit* component. Include a *render()* function that just displays a placeholder message. Also add a cancel button to navigate back to the list view. Add an *editMode* and *id* property to the *state* off the *App* component. Add the following logic to its *render()* function. Display the *MovieList* component when *editMode* is false. Render the *MovieEdit* component when *editMode* is true.

Add a *click handler* to the button in each table row in the *MovieList* component. When the button is clicked it should change the state in the *App* component to reflect that *editMode* is true. Also set the *state.id* to the *id* of the movie selected. When this happens React should render the *MovieEdit* component.

Add a *click handler* to the button on the *MovieEdit* component. In the function change the *editMode* property in the *state* in the *App* component back to false. When this happens React should render the *MovieList* component again.

Make sure your application renders without errors in the browser. You can now switch between the movie list and edit form by clicking on one of the edit buttons.

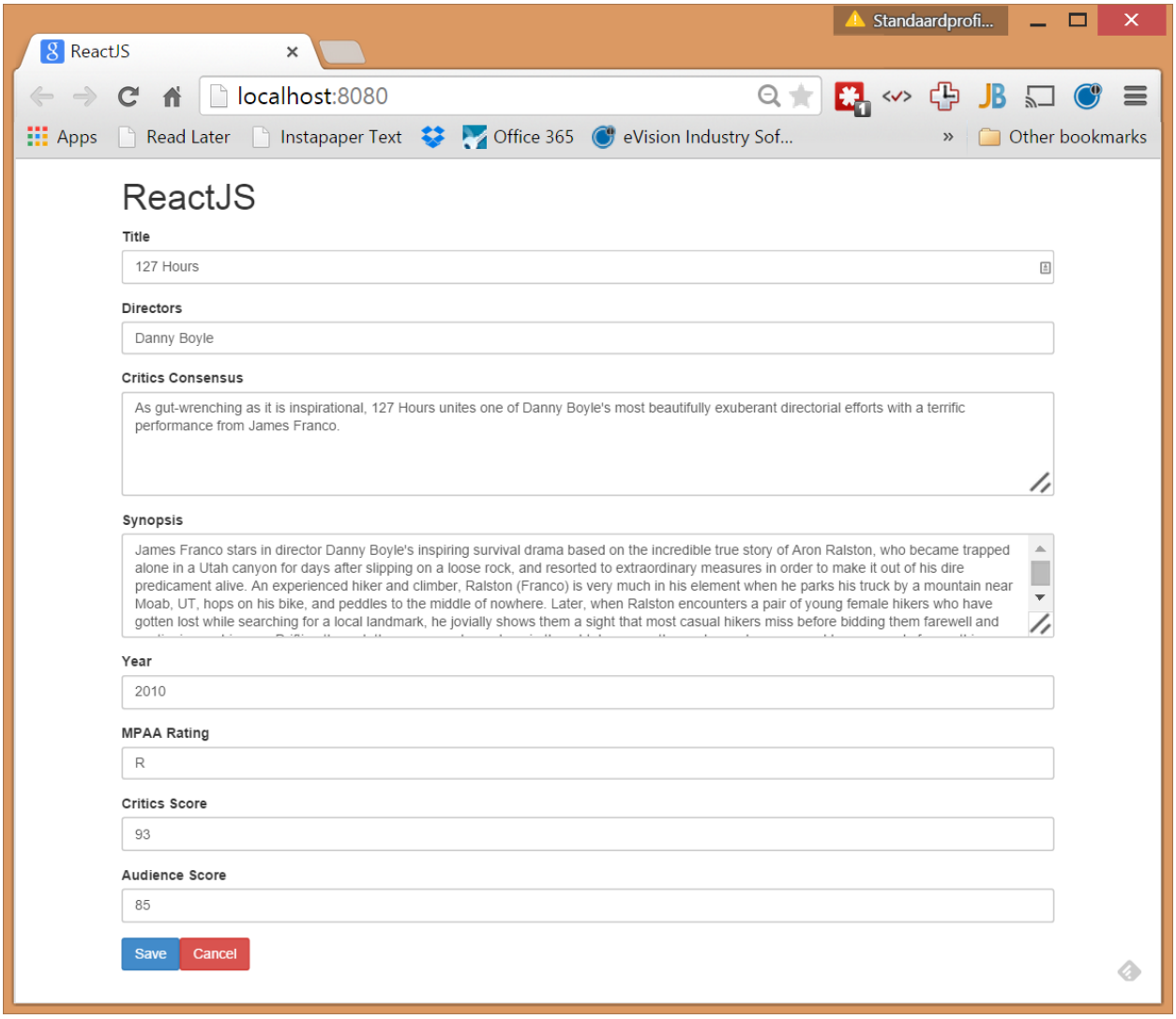
Assignment 4

Add a new **MOVIEEDIT.JS** file. Replace create a new *MovieEdit* component with code to display the complete movie details in an edit form. There are some input type text and text area components on the page. Create two components *InputText* and *TextArea* to reuse in the editor.

Add some code in the *componentDidMount()* lifecycle function to do an AJAX request to load the movie specified. You can do this using the *fetch()* function and the **/API/MOVIES/<ID>** REST endpoint. Save the loaded movie in the components state.

Add both a *Save* and a *Cancel* button to commit or cancel changes as required. When the user clicks the save button the movie should be send to the server using an AJAX put action and only when successful switch to the movie table. Clicking the cancel button should just discard all changes and switch the view back to the movie table.

Make sure your application renders without errors in the browser and you can edit movies and save or discard the changes as required.


Standaardprofi...
ReactJS
localhost:8080
Apps
Read Later
Instapaper Text
Office 365
eVision Industry Sof...
Other bookmarks
ReactJS
Title
127 Hours
Directors
Danny Boyle
Critics Consensus
As gut-wrenching as it is inspirational, 127 Hours unites one of Danny Boyle's most beautifully exuberant directorial efforts with a terrific performance from James Franco.
Synopsis
James Franco stars in director Danny Boyle's inspiring survival drama based on the incredible true story of Aron Ralston, who became trapped alone in a Utah canyon for days after slipping on a loose rock, and resorted to extraordinary measures in order to make it out of his dire predicament alive. An experienced hiker and climber, Ralston (Franco) is very much in his element when he parks his truck by a mountain near Moab, UT, hops on his bike, and peddles to the middle of nowhere. Later, when Ralston encounters a pair of young female hikers who have gotten lost while searching for a local landmark, he jovially shows them a sight that most casual hikers miss before bidding them farewell and
Year
2010
MPAA Rating
R
Critics Score
93
Audience Score
85
Save
Cancel