Nerdeez.com

Introduction to webpack

BY: Yariv Katz

LinkedIn: https://www.linkedin.com/pub/yariv-katz/23/aa0/38a

email: yariv@nerdeez.com

What we will cover

- What is Webpack
- Installing webpack
- Basic configuration - entry, output
- Installing loaders - ts-loader, babel-loader

What is Webpack

- Our web application is made out of a lot of files
- Webpack takes all our application files and pack them to a distributable application
- Webpack can replace our Grunt or Gulp
- Webpack has a lot of plugins that adds a lot of extra features
- Things we can do with Webpack:
  - Transpile our ES6 or TypeScript code to Vanilla JS
  - Convert styling files: SCSS, SASS, Less to CSS
  - Minify our code
  - Upload our code to CDN
  - Deal with caching

Install Webpack

- You can Install Webpack with **npm**

  > npm init

  > npm install webpack --save-dev

Webpack basic config

- Webpack expects a configuration file called **webpack.config.js**
- In the config file, the entry tells webpack which files to process
- In the config file, the output tells webpack where to output the files

Webpack loaders

- The loaders can be used to tell webpack how to deal with certain files
- We will use **babel-loader** to deal with our **js** files and will transpile our **ES6** code to **Vanilla JS**
- We will use **ts-loader** to transpile our **TypeScript** files to **Vanilla JS**

Webpack Summary

- webpack is a most have tool for modern applications
- It can deal with different file types with loaders
- It can create our web bundle file
- you can add community plugins for common deploy actions

Babel & JSX

- It's very common and recommended to develop React apps using ES2015 and JSX
- JSX - stands for JavaScript as XML and allows us to add HTML like syntax in our JS files
- We will be using Babel to transform our JSX and ES2015 to supported JavaScript Code
- Another common tool recommended for React Development is webpack
- Webpack takes all our project files and outputs a single minified project file
- Webpack can also integrate Babel and JSX

Babel & JSX

JS code without JSX

```
React.createClass({
  render: function(){
    return React.createElement(
      'div', null, 'Hello DevGeekWeek!'
    )
  }
});
```

JS Code with JSX

```
React.createClass({
 render: function() {
  return (
    <div>
      Hello DevGeekWeek!
    </div>
  );
 }
});
```

JS Code with Babel & JSX

```
class HelloDevGeekWeek extends React.Component{
  render() {
    return (<div>Hello DevGeekWeek!</div>);
  }
};
```

Our first React component

- To Define a React component we simply need to extend React.Component
  - Example:

```
class HelloDevGeekWeek extends React.Component {
   ...
}
```

- React Components need to have a render function which outputs the DOM elements of the component (easier to implement with JSX)

```
class HelloDevGeekWeek extends React.Component {
   render(){
      Return (<div>Hello DevGeekWeek!</div>);
   }
}
```

Exercise 1

Let's build a component that prints 'Hello DevGeekWeek!'

Solution:

https://github.com/ywarezk/react-jb-helloworld

In branch 'master'

Properties

- We can use component properties to pass data to a react component
  - Example of passing data to React component:

```
ReactDOM.render(
<HelloDevGeekWeek greeting=”Hello DevGeekWeek” />,
document.getElementById('example')
);
```

- After we pass the property we can access it in the component code
  - Example:

```
class HelloDevGeekWeek extends React.Component {
render(){
  return (
    <div>{this.props.greeting}</div>
  );
}
}
```

Exercise 2

- Add a greeting property to our HelloDevGeekWeek components
- The component should print the text it gets in the greeting

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/greeting-prop

Events

- We can handle events in our component
- In your JSX component you can add events as attributes.
  - Example:

```
render(){
  return (
    <div>
      <button onClick={this.alertHello} >
        Alert
      </button>
      <div>{this.props.greeting}</div>
    </div>
  );
}
```

- We can now enter another method in our component class to deal with the event

Exercise 3

- In our component add a button and a click event to the button
- The click event should show an alert with Hello World text

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/react-events

State

- React Components need to know when they need to re-render
- The components are connected to a state
- When the state changes the component re-renders
- React Components have a function: *getInitialState* which they can override to set the initial state of the component
- With ES6 you need to set the initial state in the constructor
- React Components get a reference to its state by accessing: *this.state*
- A Component can change its state by calling: *this.setState*
- Components can’t change the state in the render function!

Exercise 4

- Our HelloDevGeekWeek button should toggle the visibility of the greeting
  - Set the initial state of the component to have a boolean *isGreetingShown* marked true when the greeting is visible
  - When the user clicks the button the *this.setState* should toggle the *isGreetingShown* state variable
  - When the *isGreetingShown* is true we display the greeting, otherwise we hide it
  - The text on the button should also change from Show Greeting to Hide Greeting according to the isGreetingShown state.

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/states

How can I build a full application with React?

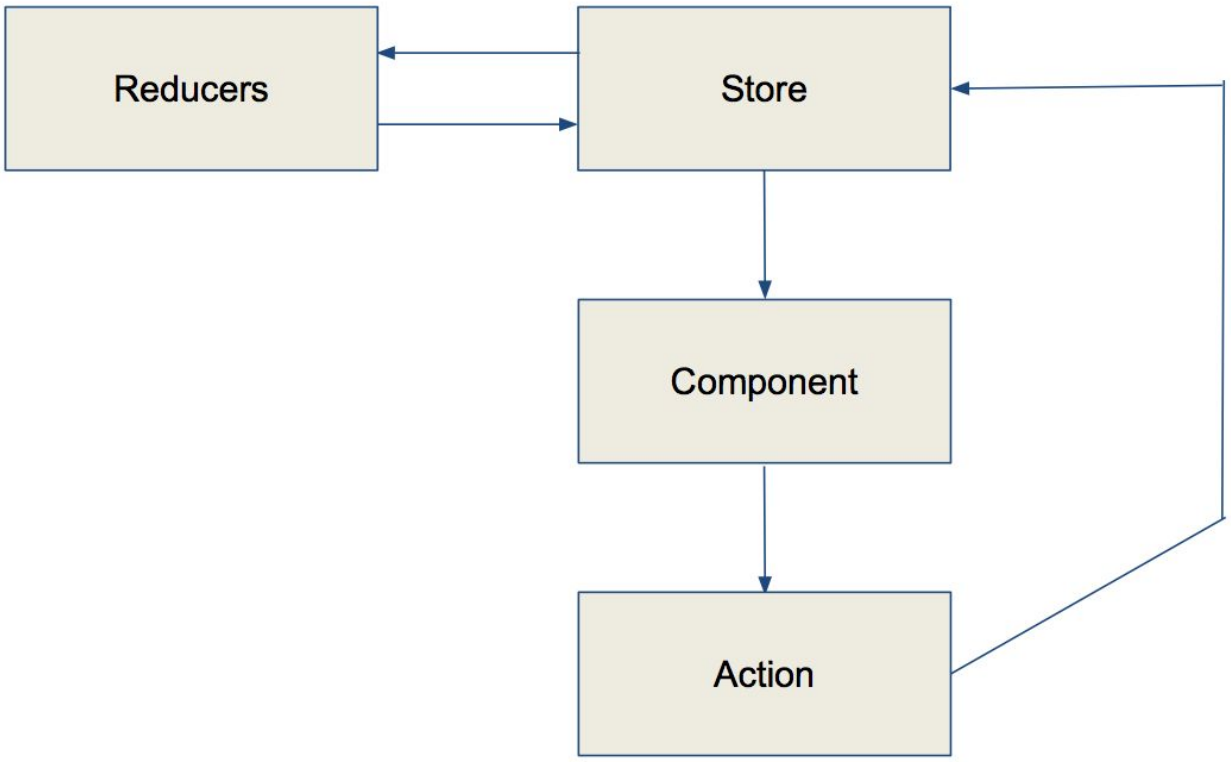
If React is just used for building UI Component, How can I use it to build a large scale Web Application

Redux

- Redux is an application architecture commonly used for building client side web applications
- Similar to Flux it keeps a Unidirectional Data Flow (better explained in the upcoming diagram)
- Redux can also work with other frameworks like AngularJS
- Redux has 3 principles:
  1. Single source of truth: the state of the app is located in a single store
  2. State is read-only and can only be changed via actions
  3. Changes to the state are executed by pure functions called Reducers

Nerdeez.com

UniDirectional Data Flow with Redux



React Redux Movie App

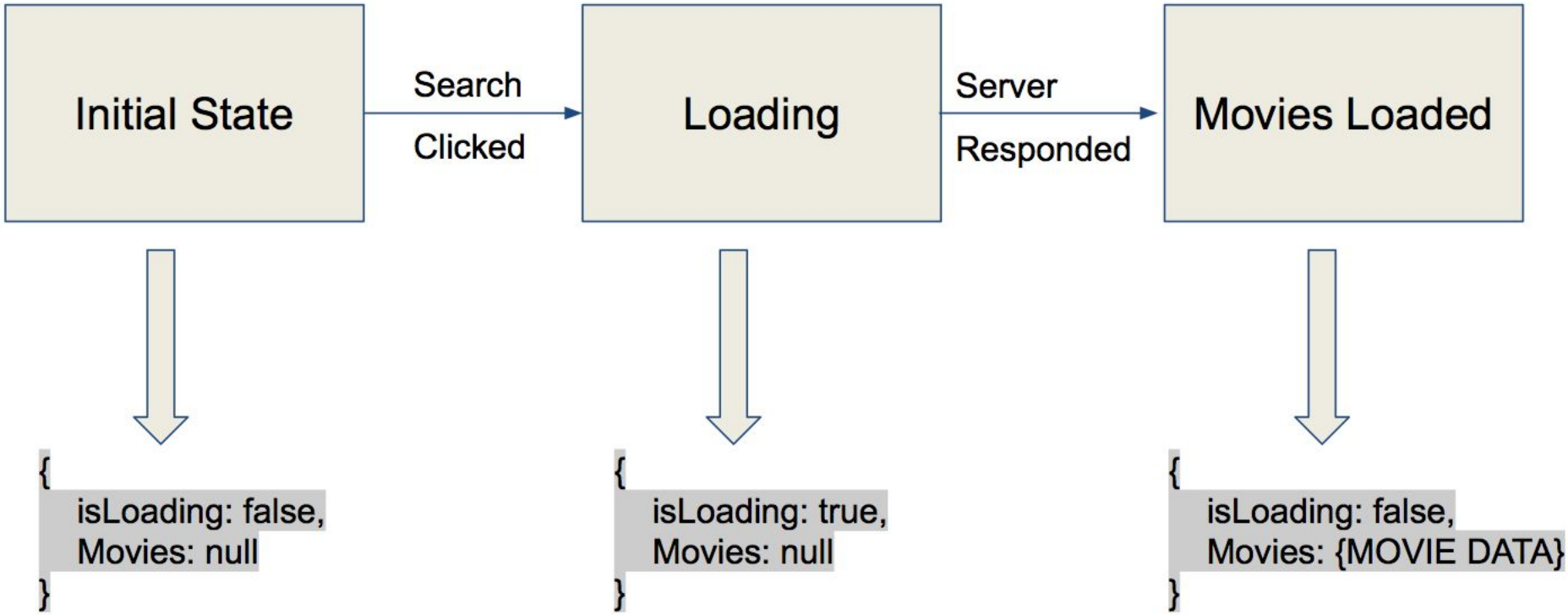
- Using React & Redux we will create our first app
- The app will display information about movies
- The app will contain a search box for searching movies
- When the user clicks on ‘search’ it will query the Open Movie Database API and display the information
- To query a search term we need to send a *GET* request to the following url:
  - http://www.omdbapi.com/?t=<SEARCH_TERM>&y=&plot=short&r=json

state machine - Exercise 5

- When you design your app with Redux, you have to start perceiving your app as a State Machine.
- In Redux we have a single store - single source of truth which holds our entire app state, as a JavaScript Object
- In our movie app, what will our State Machine look like?

Nerdeez.com

Movie App State Machine



Redux - Actions/Action Creators

- Component can change the state via actions
- Actions are plain JavaScript objects
- Action objects must contain a *type* attribute that indicate the type of action performed
- Action Creators are functions that return an action object
- A State change is performed by calling

  store.dispatch(<ACTION_CREATOR>);

Actions/Action Creators - Async Actions

- Actions Creators can also change the state asynchronously.
- To change state asynchronously the action creator needs to return a function
- The function receives the store dispatch function as a parameter
- We can change the state asynchronously by calling the dispatch method

Example:

```
function changeStateAsyncSample(){
   return function(dispatch){
       setTimeout(function(){
           dispatch(<ANOTHER ACTION CREATOR>);
       }, 3000);
   }
}
```

Action/Action Creators - Exercise 6

- In our Movie App we have 3 actions:
  - An action that changes the loading variable in the state (called when the user hits the search button)
  - An action that changes the movies object in the state (called when we receive the response from the movie API)
  - An action that requests data from the movie API server
- In our hello world project add a file called actions.js with the 3 actions described above
- Don’t forget to add that file to the index.html

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/actions

Redux - Reducers

- Actions represent something that happened in the app, and return a payload of data
- Actions do not describe how the state of the app will change
- For this we have reducers
- Reducers hold a reference to the state of the app, receive the action with the action payload, and their job is to determine how the next state will look like
- The reducer is usually a function with a switch case statement for every action
- Reducer is where we also define our initial state
- Redux will call our reducer with an undefined state when it needs our initial state

Redux - Reducers Exercise 7

- Add a file in our movie app called: *reducer.js*
- The file will contain our initial state and our reducer

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/reducer

Redux - Store

- Holds the entire state of the application - our single source of truth
- You can get the state from the store via *getState*
- Allows the state to be update via *dispatch(action)*
- In Redux library there is a *createStore* method to create our store
  - The create store receives our combined reducers as first argument
  - The second argument is the initial state of our app, which is used for server side rendering.
  - Our Redux store needs to support asynchronous actions, for that we need to apply middleware called redux-thunk

Redux - Store - Exercise 8

- Create a file named *store.js*
- In the file call *createStore* and pass it the reducer we created in exercise 7
- Apply the redux-thunk middleware to enable asynchronous actions

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/store

Redux - Movie App Component

- Let’s try and connect our store, reducer, and actions to our movie app flow with our components
- Our components will respond to app actions and start the entire flow.

Redux - Component - Exercise 9

- Create a file named components.js
- In that file create our form in the render function with a search box and a submit button
- Attach an event for the form submission which will print an alert with the text in the search field
- You can attach **ref** and use: **this.refs** to refer to the text field and grab the value

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/search-form

Redux - Provider - Exercise 10

- We created a redux store previously, now we need to make that store available to our components
- For that we use react-redux Provider, which makes our store available to our components
- In our render function wrap our components with the provider components
- The provider component needs to receive the store we created earlier

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/wrap-provider

Redux - Passing State & Actions

- We want our search component to be connected to the state and the actions
- We can use ReactRedux *connect* method for that
- The *connect* method receives two arguments:
  - *mapStateToProps* - a function that receives the state and returns an object with the part from the state that will be transferred to the component properties
  - *mapDispatchToProps* - function that receives the store dispatch function and returns an object with the actions connected to the store. This will also be transferred to the component properties, to help us connect action creators to disptach we can use bindActionCreators
- Our component will re-render when the state that the component is connected to changes

Exercise 11 - Connecting component

- We will now connect our *SearchForm* component to our state and to our action
- The *SearchForm* component will be connected to the *isLoading* and *movie* state and to the *requestMovieFromApi* action
- When isLoading is true we will display a loading text
- The submit button will call the *requestMovieFromApi*
- We will display the movies we got from the API

Solution:

https://github.com/ywarezk/react-jb-helloworld

On branch: feature/final

Final Notes

- Recommended Starter Kits:
  - https://github.com/davezuko/react-redux-starter-kit
- This lecture is published in our facebook page
  - Just search nerdeez

Thanks for listening!