React

Wat is react?

Is een User Interface Library gemaakt door facebook. Het is een template taal die HTML rendered. React beperkt zich tot user interface en interactie allen. Dit maakt dat componenten herbruikbaar zijn. React maakt het mogelijk om zowel via de client als via de server te runnen. React update en rendered alleen wanneer de view veranderd.

JSX : React maakt gebruik van JSX een subset van het vertrouwde javascript.

Redux or Flux : React heeft enkel een one-way data flow. Flux of Redux (de populairste) helpen je om de data unidirectioneel te maken.

Components : In React is alles gebaseerd op components. Alles is een comonent zodat je applicatie meer schaalbaar en onderhoudbaar wordt.

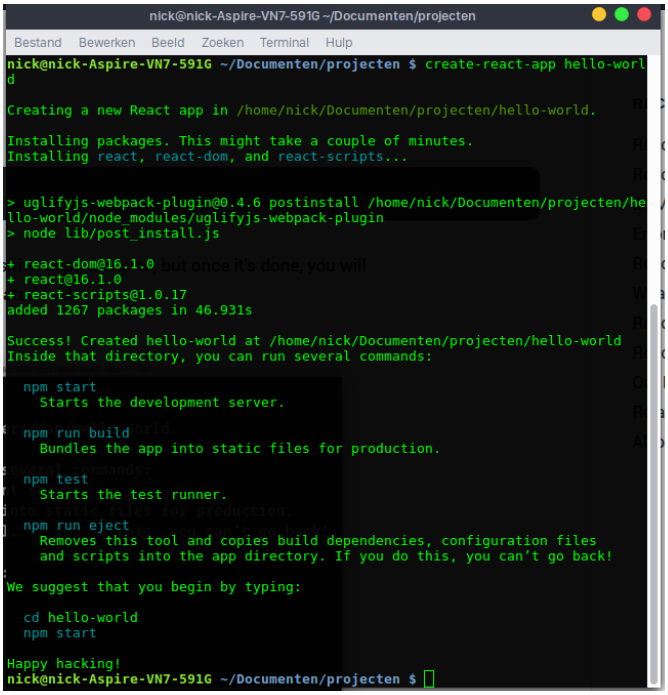
Hello world! (07/10)

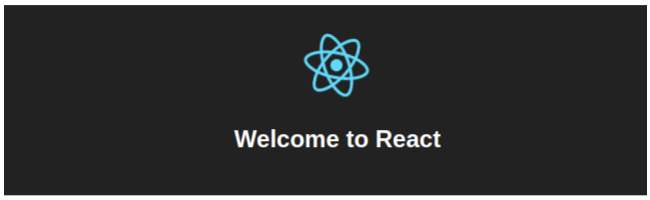
Het creëren van een applicatie is niet altijd evident als je wilt gebruik maken van de nieuwste technologieën e.g. es6, babel transpiler etc… Maar dit is verleden tijd met de react-cli die dit voor u doet.

Installing Create React App → *npm install -g create-react-app*

Creating react app → *create-react-app hello-world*

Running the application → *npm start*





To get started, edit `src/App.js` and save to reload.

My first component (07/10)

Variabelen en functies gebruiken in react :

```
class App extends Component {
  getVal(){
    return 'World'
  }
 render() {
  return (
   <div className="App">
    <header className="App-header">
     <img src={logo} className="App-logo" alt="logo" />
     <h1 className="App-title">Hello {this.getVal()}</h1>
    </header>
    <p className="App-intro">
     To get started, edit <code>src/App.js</code> and save to
reload.
    </p>
   </div>
  );
 }
}
export default App;
```

Constructor :

```
class App extends Component {
 constructor() {
   super();
   this.val = 'Nick'
 }
  render() {
   return (
    <div className="App">
     <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <h1 className="App-title">Hello {this.val}</h1>
     </header>
     <p className="App-intro">
      To get started, edit <code>src/App.js</code> and save to reload.
     </p>
    </div>
   );
  }
}
export default App;
```

Components gebruiken:

```
<Header/>
```

Component kunnen ook in een lijst gestoken worden:

```
  const list = [
    <Header/>,
    <Header/>,
    <Header/>
  ];
```

```
  {list}
```

Data Management in React (14/10)

Twee manieren waarop data wordt verwerkt in react :

State : Verkrijgbaar door ‘this.state’ default is null.

Wanneer state veranderd op een component zal de comonent re-renderen en de DOM updaten als er verandering zijn. Als er geen veranderingen zijn veranderd de DOM niet. Hoe dit werkt is : React houdt een virtuele DOM tree bij die deze vergelijkt met de eigenlijke DOM en zo veranderingen kan detecteren. Best practice : Als een component moet veranderen van data gebruik state anders gebruik props.

Hieronder een simulatie van een state change door time out. Na 1 seconde veranderd de user input en zal react deze component updaten.

```
import React, { Component } from 'react';
import './Body.css';
class Body extends Component {
  constructor() {
    super();
    this.state = {
      name: 'Nick'
    }
  }
  render() {
    setTimeout(()=>{
      this.setState({name:  'Iemand anders'});
    }, 1000);
    return (
      <p className="App-intro">
        {this.state.name}
      </p>
    );
  }
}
export default Body;
```

Props : verkijgbaar door this.props default null.

Zijn als het ware inputs die meegegeven worden door de parent component.

App.js :

```
import React, {Component} from 'react';
import './App.css';
import './Header/Header';
import Header from "./Header/Header";
import Footer from "./Footer/Footer";
import Body from "./Body/Body";
class App extends Component {
  render() {
    return (
      <div className="App">
        <Header />
        <Body text={'hallo iedereen'}/>
        <Body text={'ik ben Nick'} />
        <Footer/>
      </div>
    );
  }
}
export default App;
```

Body.js :

```
import React, { Component } from 'react';
import './Body.css';
class Body extends Component {
  render() {
    return (
      <p className="App-intro">
        {this.props.text}
      </p>
    );
  }
}
export default Body;
```

States en props zijn combineerbaar.

Meer info over props en states : https://github.com/uberVU/react-guide/blob/master/props-vs-state.md

Events en Data veranderingen (28/10)

Waneer we het inputveld invullen veranderen we de state van de app component en geven we dit door aan de props van de Header.

App.js

```
class App extends Component {
constructor(){
    super();
    this.state = {title: "User"}
  }
  changeTitle(text){
    this.setState({title: text});
  }
  render() {
    return (
      <div className="App">
        <Header  title={this.state.title}/>
        <Body changeTitle={this.changeTitle.bind(this)} />
        <Footer/>
      </div>
    );
  }
}
```

Body.js

```
class Body extends Component {
  handleChange(event) {
    const title = event.target.value;
    this.props.changeTitle(title);
  }
  render() {
    return (
      <p className="App-intro">
        <input onChange={this.handleChange.bind(this)} />
      </p>
    );
  }
}
export default Body;
```

Header.js

```
class Header extends Component {
  render() {
    return (
        <header className="head">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Hello {this.props.title}</h1>
        </header>
    );
  }
}
export default Header;
```

Project folder : hello-world

Single Page Application : react router (04/11)

Install react router : *npm install –-save react-router-dom*

Setting up routes (created components Featured, Archived, Settings

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './Pages/Layout';
import {BrowserRouter as Router, Route} from 'react-router-dom';
import registerServiceWorker from './registerServiceWorker';
ReactDOM.render(
  <Router>
    <div className="container-fluid">
      <Route path="/" component={App}></Route>
    </div>
  </Router>
  , document.getElementById('root'));
registerServiceWorker();
```

Pages/Layout.js

```
import React, { Component } from 'react';
import '../index.css';
import Header from "./Header";
import {Route, Switch} from 'react-router-dom';
import Featured from "./Featured";
import Archived from "./Archived";
import Settings from "./Settings";
class App extends Component {
  render() {
    return (
      <div>
        <Header/>
        <div>
          <Switch>
            <Route path='/archives/:article' component={Archived} />
            <Route path='/archives' component={Archived} />
          </Switch>
          <Route path='/settings' component={Settings} />
          <Route path='/featured' component={Featured} />
          <Route exact path='/' component={Featured} />
        </div>
      </div>
    );
  }
}
export default App;
```

Demo folder : single-page-basic-router

Components/Header.js

Note : activeClassName (when is active assign class) werkt alleen maar met NavLink ipv Link)

```
import React, { Component } from 'react';
import '../index.css';
import {Link, withRouter} from 'react-router-dom';
class Header extends Component {
  constructor() {
    super();
  }
  navigate(){
    this.props.history.push('/featured');
  }
  render() {
    return (
      <header>
        <nav className="navbar navbar-default">
            <div className="navbar-header">
              <a className="navbar-brand" href="#">Single Page App</a>
            </div>
            <ul className="nav navbar-nav">
              <li className="active"><Link to="/">Home</Link></li>
              <li><Link to="/archives">Archives</Link></li>
              <li><Link to="/featured">Featured</Link></li>
              <li><a onClick={this.navigate.bind(this)}>Settings</a></li>
            </ul>
        </nav>
      </header>
    );
  }
}
export default withRouter(Header);
```

Export default withRouter(Header) om de props van de router te verkrijgen en zo via code te navigeren naar /featured.

Query parameters

Setting up query parameters:

```
<Switch>
  <Route path='/archives/:article' component={Archived} />
  <Route path='/archives' component={Archived} />
</Switch>
```

Getting query parameters (URSearchParams for ?date=today&filter=bla) :

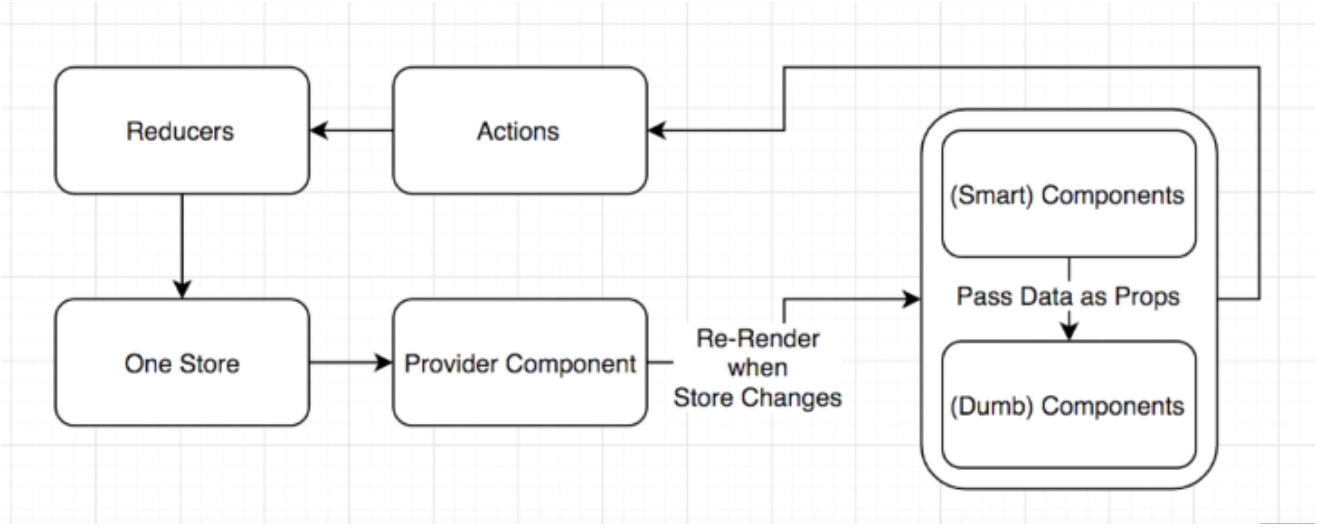
```
render() {
  const  article  = this.props.match.params.article;
  const query = new URLSearchParams( this.props.location.search);
  return (
    <div>
      <h1>Archived ({article})</h1>
      date : {query.get('date')}, filter: {query.get('filter')}
    </div>
  );
}
```

Demo folder : single-page-app

Redux (04/11)

React heeft geen mogelijkheid om alle data te beheren, components kunnen alleen maar props en states beheren. Daarom is React alleen maar een view-laag om echt tot een framework op te tillen moeten we gebruik maken van Redux (een implementatie van het Flux pattern). De verschillen tussen Flux en Redux kan je hier lezen : https://edgecoders.com/the-difference-between-flux-and-redux-71d31b118c1

Redux en Flux pattern begrijpen: http://www.youhavetolearncomputers.com/blog/2015/9/15/a-conceptual-overview-of-redux-or-how-i-fell-in-love-with-a-javascript-state-container



Provider : encapsuleert de app en injecteert de store.
Store : bevat de hele state van de applicatie.
Reducers : Luisteren naar acties en maken de veranderingen aan de store.
Actions : Geeft aan welke actie moet gebeuren.
Components : React components kunnen geïnjecteerd worden met verschillende soorten data. React components triggeren ook Redux acties zo kan alles verbonden worden.

My first Redux store!

Install Redux : *npm install redux –-save*

Om te beginnen alles in console omdat Redux niet gebonden is aan React. Redux kan met alles gebruikt worden maar werkt zeer goed met functional programming. Functional programming read trough : https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0

import redux :

```
import { createStore } from 'redux';
```

reducer :

```
const reducer = (state, action) => {
  if(action.type === 'PLUS'){
    return state + action.payload;
  }
  return state;
}
```

reducer maakt de changes actueel wanneer een store veranderd op basis van de action. Dus hier ontvangt die een action die de state van de applicatie plus de payload van de action.

create store :

```
const store = createStore(reducer, 0);
```

createStore met de reducer en een initial state. Dit is meestal een object = {…}
Voor dit voorbeeld is nul (integer) omdat het een simpel optel som wordt.

Naar store changes luisteren :

```
store.subscribe(
  () => {
   console.log('store changed', store.getState());
  }
);
```

Actions :

```
store.dispatch({type: 'PLUS', payload: 1});
store.dispatch({type: 'PLUS', payload: 10});
store.dispatch({type: 'PLUS', payload: 5});
store.dispatch({type: 'PLUS', payload: 3});
```

output :

index.js:14 store changed 1
index.js:14 store changed 11
index.js:14 store changed 16
index.js:14 store changed 19

Demo folder : redux-intro

Meerdere Reducers

```
import {combineReducers, createStore} from 'redux';

//in een echt project komen elke reducer in een aparte file.
//state = {} -> default value niet meer op de store maar op de parameter
const userReducer = (state = {}, action) => {
  switch (action.type) {
    case "CHANGE_NAME": {
      state = {...state, name: action.payload};
      break;
    }
    case "CHANGE_AGE": {
      state = {...state, age: action.payload};
      break;
    }
  }
  return state;
};
const tweetsReducer = (state = [], action) => {
  return state;
};
const reducers = combineReducers({
  //welk stuk data willen we bewerken? : en welke reducerfunction gaat dit
afhandelen?
  user: userReducer,
  tweets: tweetsReducer
});
const store = createStore(reducers);
store.subscribe(
  () => {
    console.log("store changed", store.getState());
  }
);
store.dispatch({type: 'CHANGE_NAME', payload: "Fred"});
store.dispatch({type: 'CHANGE_AGE', payload: 35});
```

Demo folder : redux-multiple-reducers

Middleware

```
import {applyMiddleware, combineReducers, createStore} from 'redux';
//in een echt project komen elke reducer in een aparte file.
//state = {} -> default value niet meer op de store maar op de parameter
const userReducer = (state = {}, action) => {
  switch (action.type) {
    case "CHANGE_NAME": {
      state = {...state, name: action.payload};
      break;
    }
    case "CHANGE_AGE": {
      state = {...state, age: action.payload};
      break;
    }
  }
  return state;
};
const tweetsReducer = (state = [], action) => {
  return state;
};
//own logger middleware
const logger = (store) => (next) => (action) => {
  console.log('action fired');
  next(action);
}
//imported middleware can be comma seperated in here.
const middleware = applyMiddleware(logger);
const reducers = combineReducers({
  //welk stuk data willen we bewerken? : en welke reducerfunction gaat dit
afhandelen?
  user: userReducer,
  tweets: tweetsReducer
});
const store = createStore(reducers, middleware);
store.subscribe(
  () => {
    console.log("store changed", store.getState());
  }
);
store.dispatch({type: 'CHANGE_NAME', payload: "Fred"});
store.dispatch({type: 'CHANGE_AGE', payload: 35});
```

Demo folder : redux-middleware

Async actions

install redux logger : *npm install –-save redux-logger*
install thunk : *npm install –-save redux*-thunk
install axios : npm install –-save axios

Axios is een xhr client.
Thunk laat ons toe om async actions te doen. Logger is een logger :p

Basic idea :

```
import {applyMiddleware, createStore} from 'redux';
import {createLogger}  from 'redux-logger';
import thunk from 'redux-thunk';
const reducer = (state = {}, action) => {
  return state;
}
const middleware = applyMiddleware(createLogger(), thunk);
const store = createStore(reducer, middleware);
store.dispatch(
  (dispatch) => {
    dispatch({type: 'FOO'});
    //do someting async
    dispatch({type: 'BAR'});
});
```

Basic api calling :

```
import  applyMiddleware, createStore} from 'redux';
import {createLogger}  from 'redux-logger';
import axios from 'axios';
import thunk from 'redux-thunk';
const initialState = {
  fetching: false,
  fetched: false,
  users: [],
  error: null
}
const reducer = (state = initialState, action) => {
  switch (action.type) {
    case 'FETCH_USERS_START': {
      return {...state,
        fetching: true};
    }
    case 'FETCH_USERS_ERROR': {
      return {...state,
        fetching: false,
        error: action.payload};
    }
    case 'RECEIVE_USERS': {
      return {...state,
        fetching:false,
        fetched: true,
        users: action.payload}
    }
    default: {
      return state;
    }
  }
}
const middleware = applyMiddleware(createLogger(), thunk);
```

```
const store = createStore(reducer, middleware);
store.dispatch(
  (dispatch) => {
    dispatch({type: 'FETCH_USERS_START'});
    axios.get('http://rest.learncode.academy/api/john/users')
      .then((response) => {
        dispatch({type: 'RECEIVE_USERS', payload: response.data});
      })
      .catch((error) => {
        dispatch({type: 'FETCH_USERS_ERROR', payload: error});
      })
});
```

Met redux-promise middleware wordt de code wat opgeschoont en worden de errors voor jou opgevangen.
Uitleg redux promise : Redux promise middleware enables robust handling of async code in Redux. The middleware enables optimistic updates and dispatches pending, fulfilled and rejected actions. It can be combined with redux-thunk to chain async actions.

Install redux-promise-middleware : npm install redux-promise-middleware –-save

with Promise Middleware :

```
import {applyMiddleware, createStore} from 'redux';
import {createLogger}  from 'redux-logger';
import axios from 'axios';
import thunk from 'redux-thunk';
import promise from 'redux-promise-middleware';
const initialState = {
  fetching: false,
  fetched: false,
  users: [],
  error: null
}
const reducer = (state = initialState, action) => {
  switch (action.type) {
    case 'FETCH_USERS_PENDING': {
      return {...state,
        fetching: true};
    }
    case 'FETCH_USERS_REJECTED': {
      return {...state,
        fetching: false,
        error: action.payload};
    }
    case 'FETCH_USERS_FULFILLED': {
      return {...state,
        fetching:false,
        fetched: true,
        users: action.payload}
    }
    default: {
      return state;}
  }
}
const middleware = applyMiddleware(promise(), thunk, createLogger());
const store = createStore(reducer, middleware);
store.dispatch({
  type: 'FETCH_USERS',
  payload : axios.get('http://rest.learncode.academy/api/john/users')});
```

Demo folder : redux-async-actions

React en Redux verbinden (11/11)

Breakdown

app command : create-react-app connect-react-redux
npm installs :

npm install –-save redux
npm install –-save redux-logger
npm install –-save redux-thunk
npm install –-save redux-promise-middleware
npm install –-save axios

Map structuur in src :
+ **actions**
  – *TweetsActions.js*
  – *UserActions.js*
+ **components**
  – Layout.js
+ **reducers**
  – *Index.js*
  – *TweetsReducers.js*
  – *UserReducer.js*
– *index.css*
– *index.js*
– *store.js*

Connecting

install react-redux : *npm install –-save react-redux*

2 stappen om te connecteren met de store :

1) encapsuleer het toplevel component met de react-redux provider en store meegeven

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import { Provider } from 'react-redux';
import Layout from './components/Layout';
import store from './store';
import registerServiceWorker from './registerServiceWorker';
ReactDOM.render(<Provider store={store}><Layout /></Provider>,
document.getElementById('root'));
registerServiceWorker();
```

2) Import connect op de react components

```
import React, { Component } from 'react';
import logo from '../logo.svg';
import './Layout.css';
import { connect } from 'react-redux';
import { fetchUser } from "../actions/Useractions"
import {fetchTweets} from "../actions/TweetsActions";
class App extends Component {
  componentWillMount(){
    this.props.dispatch(fetchUser());
```

```
  }
  fetchTweets(event){
    this.props.dispatch(fetchTweets());
  }
  render() {
    const {user, tweets } = this.props;
    if(!tweets.length){
      return <button onClick={this.fetchTweets.bind(this)} >Load tweets</button>
    }
    const mappedTweets = tweets.map(tweet => <li>{tweet.text}</li>);
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome {user.name}</h1>
        </header>
        <p className="App-intro">
          <ul>
            {mappedTweets}
          </ul>
        </p>
      </div>
    );
  }
}
export default connect((store) => {
  return {
    user: store.user.user,
    userFetched: store.user.fetched,
    tweets: store.tweets.tweets
  }
})(App);
```

Volledig project todo-lijstje

zie folder react-todo