Proof of concept

De voordelen van React koppelen met Angular

Zoals besproken in de voorbije studies hebben zowel React als Angular hun voor en hun nadelen. Maar wat als we die voor en nadelen kunnen bundelen tot een ultieme oplossing? Een framework met conventies, een cli en de meest gebruikte libraries voor developers. Dit is wat ik ga proberen maken met deze proof of concept. Het enige nadeel wat je nooit zelf kan voorkomen is dat libraries verouderd worden, aangezien Angular onderhouden wordt door Google en libraries door andere developers kan je nooit 100% zeker zijn dat een library in de toekomst zal verder leven en sta ik voor onderhoud er volledig zelf voor.

Welke libraries?

Er zijn heel wat libraries en een keuze maken is geen gemakkelijke taak. Ik heb deze gekozen omdat deze de meest onderhouden zijn (momenteel), en ik denk dat dit al heel wat functionaliteit biedt out-of-the-box.

– LESS,SASS

– MobX

– Routing

– Lazy loading componenten

– Webpack

– Typescript (???)

Ik heb de overstap gemaakt naar mobX ipv Redux omdat (ik persoonlijk vind) dit een makkelijkere versie is van Redux en het coderen gaat veel sneller met mobX dan met Redux. Ook zijn mobX objecten observable en gaan deze automatisch reageren op changes (meer gelijkend op Angular). Een volledig vergelijk vind je hier : https://www.robinwieruch.de/redux-mobx-confusion/

Create a boilerplate

De eerst stap dat ik ga doen is een boilerplate maken met een structuur in. Ik heb hiervoor eerst onderzoek gedaan hoe de meeste developers hun react-project structureren en daarvan de best practices uithalen.

Stappen :

npm install van mobx, react-router, lazy-router, sass (npm run all + package.json aanpassen), axios

De basis boilerplate heeft een aantal functionaliteiten dat ik erin wou zetten namelijk :

– Een simpele Api call in de mobx store en het weergeven van de resultaten op een pagina.

– Async componenten laden

– Opsplitsing van componenten.

– Een duidelijke structuur waarbij component in een folder zitten met hun bijgevoegde scss file.

– Een beveiligde route waar niet naartoe kan gegaan worden zonder ingelogd te zijn → authenticatie faken

Project folder : **nick-react-boilerplate**

Een command-line interface maken

Om te beginnen met de command-line interface heb ik eerst wat onderzoek moeten doen over hoe ik dit juist aanpak ik ben dan tot volgende conclusie gekomen :

Een npm package maken in **node** met de packages :

– shellJs → command line commando's uitvoeren in node ('*shell.exec*')

– colors → command line kleuren bij bepaalde acties weergeven ('*console.log("error".red)*')

– Meerdere zullen waarschijlijk toegevoegd worden naarmate ik meer features zal toevoegen.

Create-react-app uitbreiden

Run the create-react-app command :

```
#!/usr/bin/env node
let shell = require('shelljs')
let colors = require('colors')
let fs = require('fs')
let appName = process.argv[2]
let appDirectory = `${process.cwd()}/${appName}`
const run = async () => {
  let success = await createReactApp()
  if (!success) {
    console.log('Something went wrong while trying to create a new React app using
create-react-app'.red)
    return false;
  }
  await cdIntoNewApp()
  console.log("All done")
}
const createReactApp = () => {
  return new Promise(resolve => {
    if (appName) {
      shell.exec(`create-react-app ${appName}`, (stdout) => {
        console.log('stdout: ' + stdout);
        console.log("Created react app".green)
        resolve(true)
      })
    } else {
      console.log("\nNo app name was provided.".red)
      console.log("\nProvide an app name in the following format: ")
      console.log("\ncreate-nick-react ", "app-name\n".cyan)
      resolve(false)
    }
  })
}
const cdIntoNewApp = () => {
  return new Promise(resolve => {
    shell.exec(`cd ${appName}`, () => {
      resolve()
    })
  })
}
run()
```

Project folder : **react-cli-v1**

Run the npm commands :

```
const installPackages = () => {
  return new Promise(resolve => {
    console.log("\nInstalling react-router, react-router-dom, react-lazy-route, axios,
sass and mobx...".cyan);
    shell.exec(`npm install --save react-router react-router-dom react-lazy-route axios
mobx mobx-react node-sass-chokidar npm-run-all`, {cwd: appDirectory}, () => {
      console.log("\nFinished installing packages\n".green);
      resolve()
    });
  });
}
```

Edit the package.json to compile sass files :

```
const updatePackage_json = () => {
  let scripts = {
    "build-css": "node-sass-chokidar src/ -o src/",
    "watch-css": "npm run build-css && node-sass-chokidar src/ -o src/ --watch
--recursive",
    "start-js": "react-scripts start",
    "start": "npm-run-all -p watch-css start-js",
    "build-js": "react-scripts build",
    "build": "npm-run-all build-css build-js",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  };
  return new Promise(resolve => {
    console.log("\nUpdating package.json....".cyan);
    shell.exec(`json -I -f package.json -e 'this.scripts=` + JSON.stringify(scripts) +
"'", {cwd: appDirectory}, () => {
      resolve();
    });
  })
}
```

Hier werk ik met de json npm package. Daarmee kan ik gemakkelijk de package.json kan inlezen en de waarde scripts kan veranderen door mijn variabele scripts. Als er in de toekomst dan nog aanpassingen moeten komen kan ik hier gemakkelijk nieuwe dingen toevoegen of wijzigen.

Generate the boilerplate :

```
const generateBoilerplate = () => {
  console.log("\nGenerating boilerplate...".cyan);
  return new Promise(resolve=>{
    fs.unlinkSync(`${appDirectory}/src/App.css`);
    fs.unlinkSync(`${appDirectory}/src/App.js`);
    fs.unlinkSync(`${appDirectory}/src/App.test.js`);
    fs.unlinkSync(`${appDirectory}/src/index.css`);
    fs.unlinkSync(`${appDirectory}/src/logo.svg`);
    fs.copySync('./templates', `${appDirectory}/src`);
    resolve();
  })
}
```

Hier verwijder ik de files die al bestaan en kopieer mijn boilerplate templates in de huidige project folder.

Project folder : **react-cli-v2**

Een component genereren

Om een component te genereren ga ik op dezelfde manier tewerk als bij de create van de boilerplate. Alleen zal hier de uitdaging zijn om binnen mijn template de classnaam te nemen die als waarde wordt ingegeven. Alsook op te vangen wanneer er een pad wordt opgegeven. Nadien wil ik nog optionele flags maken.

Ik zat op een knooppunt binnen de node cli applicatie want ik moest meerdere commando's binnen dezelfde package uitvoeren. Maar door wat google werk was ik op de package 'Commander' gekomen hiermee kan je custom commando's maken met elke optionele flags e.d. Ideaal voor deze toepassing!

Het maken van de commando's met optionele flags :

```
program
  .version('1.0.0')
  .command('init <dir>')
  .option('-T , --typscript', 'Install with typescript')
  .action(createReact);
program
  .command('gc <component>')
  .option('-n, --nofolder', 'Do not wrap component in folder')
  .option('-o, --observable', 'Make observable')
  .option('-s, --style', 'With stylesheet')
  .option('-f, --functional', 'Create functional component')
  .action(createComponent);
program.parse(process.argv)
```

Voor dit te laten werken moest ik ook een antal aanpassingen maken aan de huidige code. Zo wordt de function Run() niet meer opgeroepen maar is dit gebundeld in de functie createReact die de parameter <dir> meekrijgt (automatisch in commander). Ook moest ik van al mijn constante arrow functie gewone funties maken om dit te laten werken…

De createReact functie :

```
async function createReact(dir) {
  appName = dir;
  appDirectory = `${process.cwd()}/${appName}`;
  if(fs.existsSync(appDirectory)) {
    console.log('Directory already exists choose antother name...'.red);
    process.exit(1);
  }
  let success = await createReactApp();
  if (!success) {
    console.log('Something went wrong while trying to create a new React app using
create-react-app'.red);
    process.exit(1);
  }
  await installPackages();
  await updatePackage_json();
  await generateBoilerplate();
  console.log("All done");
}
```

Op deze moment ben ik vergeten al mijn tussenstappen bij te houden om een component te genereren en zullen de optionele flags ook al in de code zitten. Aangezien ik hier vrij lang op zitten zoeken heb was mijn focus eerder op code schrijven en niet de log bij houden hiervoor mijn excuses.

Het generen van een component :

Ik ben begonnen met mijn template op te stellen. Dit door alle onderdelen van een react-component op te delen in constante variabelen zodat ik deze gemakkelijk ik mijn code kon smijten. Een voorbeeld van de hoofd functie :

```
const main = `
class :className extends Component {
 constructor(props){
  super(props);
 }
 render(){
  return (
   <div className=":className">

   </div>
  )
 }
}
:className.propTypes = {
}
`
```

Die ik dan exporteer :

```
module.exports = {
 main: main,
 imports: imports,
 exported: exported,
 functional: functional
}
```

De volledige template code kan je vinden in : **react-cli-commander/component_template**

Dit is de volledige code die ik ben bekomen om een template te maken (met optionele flags om functionele componenten te generen en een observable te maken :

```
async function createComponent(component, cmd){
  newCompPath = component;
  cmd.nofolder ? nofolder = true : nofolder = false;
  cmd.functional ? functional = true : functional = false;
  cmd.observable ? observable = true : observable = false;
  cmd.style ? stylesheet = true : stylesheet = false;
 if(fs.existsSync('./src/components')){
   newCompPath = `./src/components/${component}`;
 }
  let template = await buildTemplate();
  writeFile(template, component)
}
function buildTemplate(){
   let imports = [template.imports.react, template.imports.propTypes];
   if(observable){
     imports.push(template.imports.observable)
   }
   if(stylesheet){
     imports.push(template.imports.stylesheet);
   }
   let body = functional ? [template.functional] : [template.main].join('\n');
```

```
    let exported = observable ? [template.exported.observable] :
[template.exported.default];
    return imports.join('\n') + '\n' + body + '\n' + exported;
}
function capitalize(comp){
  return comp[0].toUpperCase() + comp.substring(1, comp.length);
}
function writeFile(template, component){
  let path = newCompPath;
  if(nofolder){
    strArr = newCompPath.split('/');
    strArr.splice(strArr.length - 1, 1);
    path = strArr.join('/');
    console.log(path);
  }
  let comp = component.split('/');
  comp = comp[comp.length -1 ];
  if(path){
    path = path + '/' + capitalize(comp);
  }else{
    path = capitalize(comp);
  }
  if(stylesheet){
    if(!fs.existsSync(`${path}.scss`)){
      console.log('creating syles');
      fs.outputFileSync(`${path}.scss`, '');
      console.log(`Stylesheet ${comp} created at ${path}.scss`.cyan)
    }else{
      console.log(`Stylesheet ${comp} allready exists at ${path}.scss, choose another
name if you want to create a new stylesheet`.red)
    }
  }
  if(!fs.existsSync(`${path}.js`)){
    fs.outputFile(`${path}.js`, template, (err) => {
      if (err) throw err;
      replace({
        regex: ":className",
        replacement: capitalize(comp),
        paths: [`${path}.js`],
        recursive: false,
        silent: true,
      });
      console.log(`Component ${comp} created at ${path}.js`.cyan)
    });
  }else{
    console.log(`Component ${comp} allready exists at ${path}.js, choose another name if
you want to create a new component`.red)
  }
}
```

Hierbij was vooral de uitdaging om rekenening te houden als iemand een pad opgeeft met de optionele flag -n (-nofolder) waarbij een component niet gewrapt wordt in een folder. Ik had dit eerst met een aparte flag <path> gedaan waarbij een custom path kon opgegeven worden maar vond dit niet gebruiksvriendelijk dus moest ik het laatste deel van het pad het component zelf worden en niet de map worden gemaakt bvb :

Invoer : rct gc **folder/comp** -n → zonder de flag ‘-n’ wordt er een folder gemaakt met de component naam. Maar als dit niet zo is moest de folder gewoon (niet) gemaakt worden dit heb ik dan opgelost moest het laatste deel te splitten van deze string. **Niet de beste oplossing maar mits refractor wordt dit opgelost...**

Npm package publiceren

Na alles getest te hebben heb ik om het plaatje compleet te maken de npm package gepubliceerd. Dit was zeer gemakkelijk en werkt direct… Dit geeft mij ook een drijfveer om in de toekomst hieraan nog te werken en verbeteringen door te voeren. Dingen die op de agenda staan :

– Keuze tussen Redux en MobX bij init

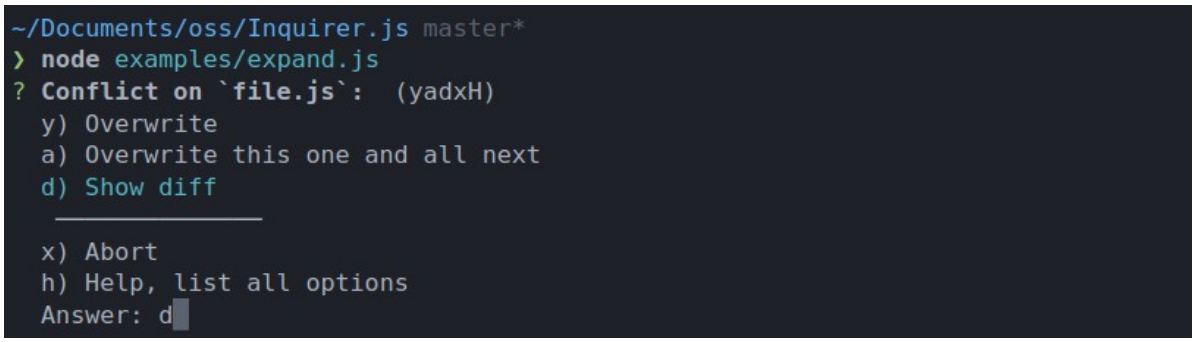
Bij mijn component template genereren zitten deze variabelen er al in maar hiervoor moest ik mijn boilerplate en npm installs ook weer aanpassen en tijd was schaars.

– Typescript optie

Dit wou ik normaal er al van in het begin in stoppen maar was te complex van configuratie om nog af te krijgen en ik heb mijn focus vooral gelegd op de cli.

– inquirerJs gebruiken voor n00bs

Inquirer voor node zijn eigenlijk user interfaces voor de command-line zodat wanneer een user fout input geeft ofzo ik dit mooier kan opvangen plus dat het makkelijker zou zijn om te starten met de tool. Voorbeeld van inquirer :



– Algemene uitwerking van de cli

Dit zijn ondere andere stores generen vanuit de cli, props kunnen meegeven bij component genereren, ...

Features (tot nu toe) van de cli :

Package installeren :

npm install -g cli-react

Project initializeren :

rct <projectNaam> init

Class component genereren :

rct gc <compNaam>

Functional component genereren :

*rct gc <compNaam> -**f***

Componenter obersver maken :

*rct gc <compNaam> -**o***

Component genereren met bijhorende scss file :

*rct gc <compNaam> -**s***

Component generen maar niet in folder wrappen :

*rct gc <compNaam> -**n***

***combinatie mogelijk bvb :**

rct gc <compNaam> -f -o -s -n => creert een functionele component dat observer is met bijhorende stylsheet maar niet gewrapt in een folder.

Zie readme op : https://www.npmjs.com/package/cli-react

Besluit

Ik had het eigenlijk veel moeilijker verwacht om een cli tool te maken. Gelukkige had ik voor ik begon goed geïnformeerd en gelezen over hoe ik dit het best kon aanpakken en op die manier heb ik niet teveel code moeten herschrijven. Het moeilijkste van de opdracht vond ik om tijdens het coderen te denken aan alle use-cases die zouden kunnen voorvallen. Ook een moeilijkheid was om dynamisch files te bewerken door user input.

Ik ben wel zeer tevreden van het eindresultaat en had niet gedacht dat ik het generen van componenten ook ging klaar krijgen. In ieder geval heb ik er zeer veel uit bijgeleerd en eveneens mijn eerste npm package geschreven, nu hopen dat mensen het gebruiken dat het geen eenzame dood tegemoed gaat.

Hierbij de link naar de npm package, de github van de (gepublishte) code en de github van de boilerplate :

boilerplate : https://github.com/nickverstocken/generated-react-boilerplate

cli-react : https://github.com/nickverstocken/cli-react

npm-package : https://www.npmjs.com/package/cli-react