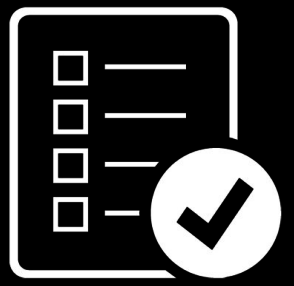
Programming 2.3

Web API - week 3

KdG Karel de Grote Hogeschool

- **Introduction**
- **Tools**
- **Step by step guide**
- **Adding dependencies**
- **Architecture**



ACS Frontend Development

Year 1	HTML / CSS / JS files • Separate folders • Importing using **`script`** and **`link`** elements • No backend	
Year 2	Backend • Gradle-based project • JVM dependencies • Spring Framework	Frontend • **Embedded** NPM-based project • NPM dependencies
Year 3	Backend • Gradle-based project • JVM dependencies • Spring Framework	Frontend • NPM-based project • NPM dependencies • Framework (React, ...)

Why a frontend project?

- Complex frontend logic
  - Frontend-specific Frameworks and APIs
  - Automated testing for frontend logic
- Dependencies are versioned and upgradeable
  - Explicit dependencies
    - We can’t just call any function/variable that’s listed in the HTML before the current script
    - ECMAScript modules solve *some* of these issues

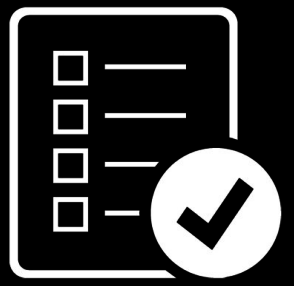
      ⇒ Browser standards are slower to catch up

Why a frontend project?

- A “build” step for JavaScript
  - Build checks at development time (missing imports, undefined variables, …)
  - Transpilation
    - Languages/Syntax: TypeScript, WASM, CSS preprocessors, …
    - Deployment: Tree-shaking, Minification, …

The JavaScript ecosystem has its own tools for all of the above (building, automated testing, handling dependencies, …)

- Introduction
- Tools
- Step by step guide
- Adding dependencies
- Architecture



NodeJS / npm

- NodeJS == Browserless (server-side) JavaScript

- npm == NodeJS Package Manager
  - Included in NodeJS Download

- Download: https://nodejs.org/en/

Browserless JavaScript code can be used to:
- Run checks on our code
- Fetch dependencies
- Run automated tests
- …

npm

- npm is the Gradle for JavaScript
- Generate an npm project using

```
$ npm init
```

- Generates a **`package.json`** file
  - The core of an npm project
- **`"dependencies": { ... }`**
  - Application dependencies
- **`"devDependencies": { ... }`**
  - Development / build dependencies

npm

```
{
    "name": "books",
    "version": "0.1.0",
    "author": "Lars Willemsens",
    "license": "MIT",
    "main": "index.js",
    "dependencies": {
        "animejs": "^3.2.1"
    },
    "devDependencies": {
        "webpack": "^5.58.2",
        /* ... */
    },
    "scripts": {
        "build": "webpack",
        "start": "webpack serve"
    }
}
```

npm

- Specifying versions
- Download (dev)dependencies:

```
$ npm install
```

Or: `npm i`

- Versions of dependencies (and *their* dependencies) are locked in **`package-lock.json`**
  - A generated file that should be added to git
- Dependencies are downloaded to **`node_modules`**
  - This directory should *not* be added to git

npm

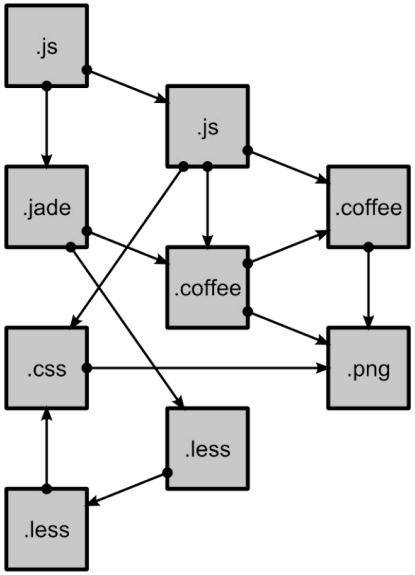
- Scripts can contain any command
- npm packages (that have a CLI) can be called from the scripts section as well

- Execute a script:

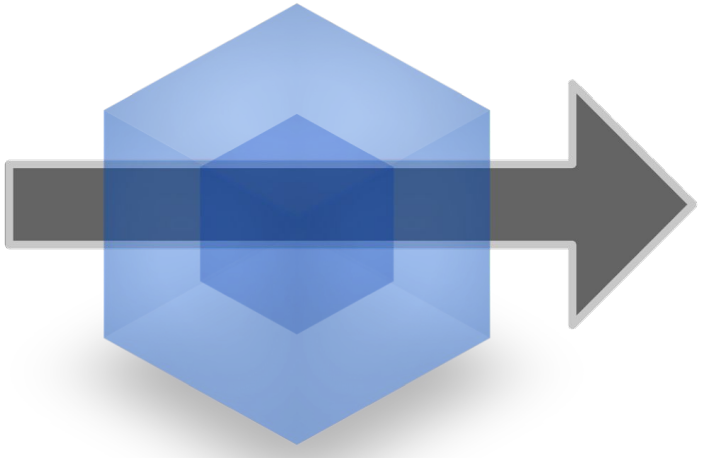
```
$ npm run build
```

```
...
"scripts": {
    "build": "webpack",
    "clean": "rm -rf dist"
}
...
```

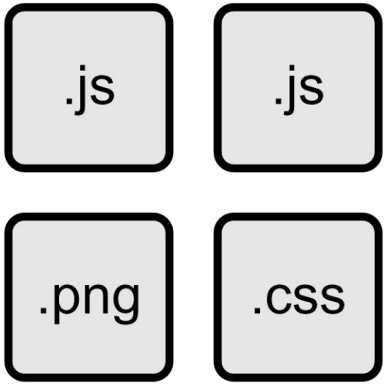
webpack



modules
with dependencies



webpack
MODULE BUNDLER



static
assets

webpack

A bundler for javascript and friends.
Packs many modules into a few bundled assets.

- Today it's essential, tomorrow maybe not
- Also calls our transpilers, minifiers, … (TypeScript, Sass, …)
  - So we'll still need the "build" step even if webpack disappears. Browsers don't speak TypeScript …

webpack

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin')
const path = require('path')

module.exports = {
    entry: './src/main/js/index.js',
    output: {
        filename: 'bundle.js',
        path: path.resolve(__dirname, 'src/main/resources/static/js')
    },
    mode: 'development',
    resolve: { extensions: ['.js'] },
    module: {
        rules: [
            {
                test: /\.css$/,
                use: [MiniCssExtractPlugin.loader, {loader: 'css-loader'}]
            }
        ]
    },
    plugins: [
        new MiniCssExtractPlugin({
            filename: '../css/bundle.css'
        })
    ],
}
```

Simplified example…
some parts are left out.
We'll simply **copy/paste** this file *most* of the time.

webpack

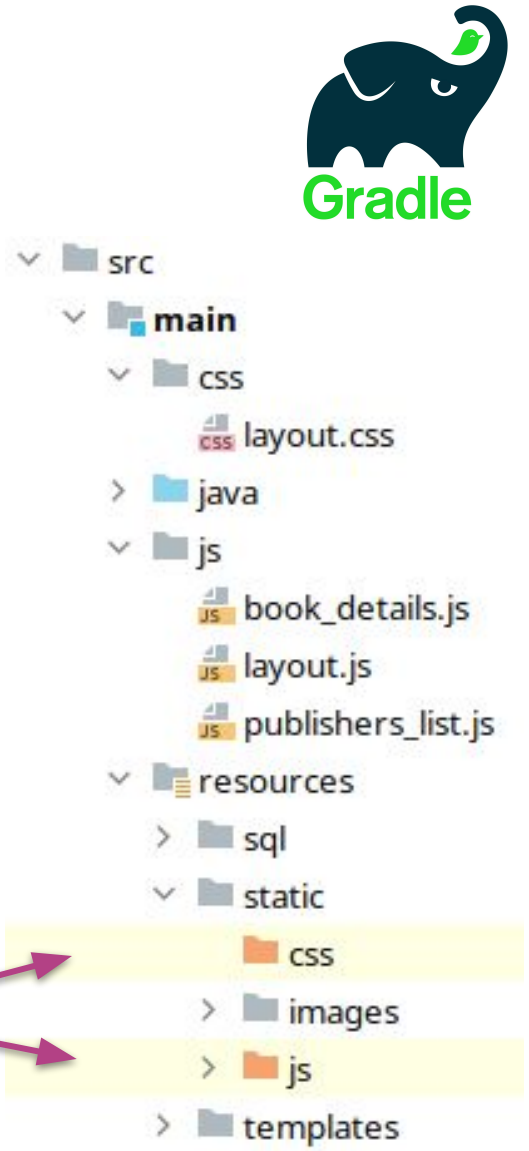
- Webpack works with **entries**
  - Each entry leads to a bundle
  - *Multiple* entries since we have an MPA

Multi-page application (= multiple views in MVC), as opposed to an SPA (single-page application).

- We'll create a single bundle for CSS as well
  - Same advantages (syntax checks, minification, …)

Application structure

- `src/main/java`
- `src/main/js`
- `src/main/css`
- `src/main/resources`



Gradle setup

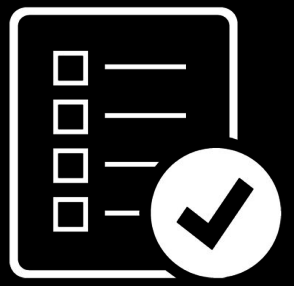
- Gradle plugin for npm
  - Installation and usage
- **build.gradle**:

```
plugins {
    ...
    id 'com.github.node-gradle.node' version '3.2.0'
}
...
```

```
$ ./gradlew npmInstall
```

Expects a **package.json** in the project root directory.

- Introduction
- Tools
- **Step by step guide**
- **Adding dependencies**
- **Architecture**



Steps

1. Generate **`package.json`**
2. Update **`package.json`**
3. Add webpack configuration file
4. Add the Gradle Plugin for Node
5. Move JS and CSS files over
6. Import CSS from the main JS entry
7. Generate the bundles
8. Update your Thymeleaf pages
9. Update **`.gitignore`**
10. Test your app., update your repository

Step 1

1. Generate **`package.json`**
    - Download and install NodeJS: https://nodejs.org/en/
    - Ensure that the commands **`node`** and **`npm`** are available
        - Update your **`PATH`** if necessary
    - From the *project root*, run the following:

```
$ npm init
```

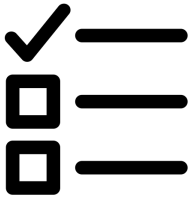
Step 2

2. Update **`package.json`**
   - Ensure that you have these:

```
...
"devDependencies": {
    "css-loader": "^6.4.0",
    "mini-css-extract-plugin": "^2.4.2",
    "source-map-loader": "^3.0.0",
    "webpack": "^5.58.2",
    "webpack-cli": "^4.9.0"
},
"scripts": {
    "build": "webpack"
}
...
```

For now, all that we're adding is webpack related.

Step 3



3. Add webpack configuration file

- Download it from Canvas
- Verify the following:
  - ➢ `./src/main/js`

    `src/main` should already exist, relative to the location of package.json

    The **`js`** subdirectory should be created

  - ➢ `../css/`

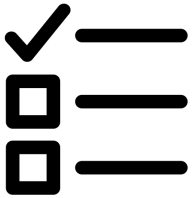
    This path is *relative* to the **`js`** directory, so create **`src/main/css`** as well

Step 4

4. Add the Gradle Plugin for Node
    - Edit **`build.gradle`** and add this plugin:

```
plugins {
    ...
    id 'com.github.node-gradle.node' version '3.2.0'
}
...
```

Step 5

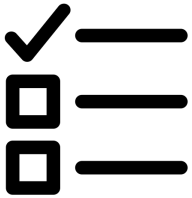


5. Move JS and CSS files over
    - Move JS files
        - From **`src/main/resources/static/js`**
        - To **`src/main/js`**
    - Move CSS code
        - From **`src/main/resources/static/css`**
        - To a single file **`src/main/css/style.css`**

Check the **`fs.readdirSync`** statement in **`webpack.config.js`**:

- All JS files will be picked up from **`src/main/js`**
- They'll be added as separate entries for webpack

Step 6



6. Import CSS from the main JS entry
    - You need a sitewide webpack entry which is tied to your layout (**`th:include`**)
        - Create **`layout.js`** if you don't have one yet
        - It should contain JS code that runs on all pages (i.e., navbar code)
        - From this file import your CSS to be picked up by webpack:

```
import '../css/style.css'
```

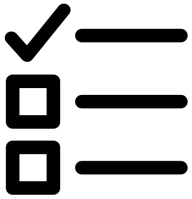
Step 7

7. Generate the bundles
    - Download the npm dependencies:

```
$ ./gradlew npmInstall
```

npm dependencies need to be downloaded whenever the (dev)dependencies change!

Step 7



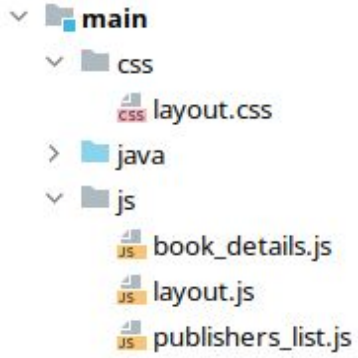
- Use Gradle to run webpack:

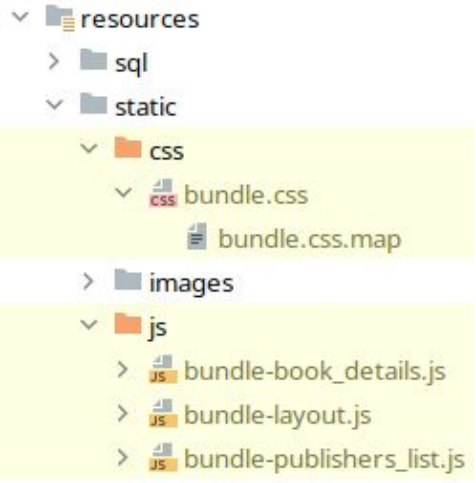
```
$ ./gradlew npm_run_build
```

This command effectively executes **`npm run build`**

Execute this step whenever JS or CSS has been modified!

- Check if your bundles are created:





Step 8

8. Update your Thymeleaf pages
    - Each Thymeleaf page should refer to a bundle instead of an unprocessed JS file

The layout page that contains the navbar:

```
<head>
    <meta charset="UTF-8">
    <title>Layout page</title>
    <script src="/webjars/bootstrap/5.1.3/js/bootstrap.min.js"></script>
    <script src="/js/bundle-layout.js" defer></script>
    <link rel="stylesheet" href="/webjars/bootstrap/5.1.3/css/bootstrap.min.css"/>
    <link rel="stylesheet" href="/css/bundle.css"/>
</head>
```

One of the pages with custom CSS:

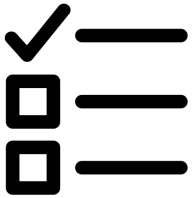
```
<head>
    <title>Book Details</title>
    <script src="/js/bundle-book_details.js" defer></script>
</head>
```

Step 9

9. Update **.gitignore**

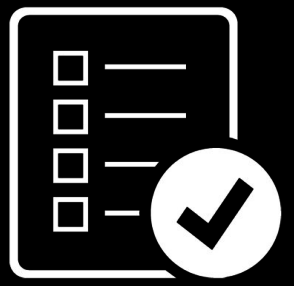
```
node_modules/
bundle*.*

*.db
*.log

HELP.md
...
```

Step 10



10. Test your app., update your repository
    - Make sure your application works as before:
        - Specifically, check your JS functionality
        - Check your custom CSS
    - Git add, commit, and push to GitLab

- Introduction
- Tools
- Step by step guide
- **Adding dependencies**
- **Architecture**



NPMJS

- Like Gradle, npm is repository-based
- https://www.npmjs.com/
- Some suggestions:

axios	animejs	lodash
luxon	sass (*)	flatpickr
bootstrap-icons (*)	chart.js	validator
@popperjs/core		

- Extra:

rxjs	typescript (*)

(*) requires changes to webpack.config.js

AnimeJS

- JS:

```
import anime from 'animejs'

const searchResultAnime = anime({
   targets: searchResultDropDown,
   opacity: 1,
   easing: 'easeInOutQuad',
   duration: 400,
   autoplay: false,
   direction: 'normal'
});
```

```
function hideDropDown() {
    if (isDropDownShowing) {
        isDropDownShowing = false;
        searchResultAnime.finished
                .then(reverseAnimation);
        searchResultAnime.play()
    }
} // similar to showDropDown (TODO: refactor)

function reverseAnimation() {
   if (isDropDownShowing) {
       searchResultAnime.direction = 'reverse'
   } else {
       searchResultAnime.direction = 'normal'
   }
}
```

- CSS:

```
#searchMenuDropdown {
   opacity: 0;
   display: block; // hidden by Bootstrap
}
```

Bootstrap Icons

- JS:

```
import 'bootstrap-icons/font/bootstrap-icons.css'
```

- HTML:

```
<button class="btn btn-outline-success" type="submit">
    <i class="bi bi-search"></i> Search
</button>
```

- webpack config, add a rule for fonts:

```
{
   test: /\.(woff|woff2|eot|ttf|otf)$/i,
   type: "asset",
   generator: {
       filename: '../fonts/[hash][ext][query]'
   }
}
```

Sass

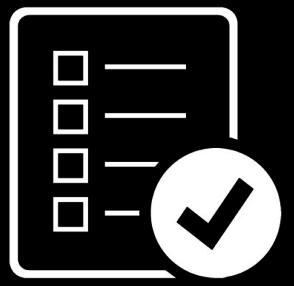
- **`package.json`**
  - Add '**`sass-loader`**' as well a '**`sass`**'
- Rename your **`.css`** file → **`.scss`** extension
  - Update your import: `import '../css/layout.scss'`
- Use some of the Sass features:

```
$highlight-color: red;
div.card {
   background-color: $highlight-color;
}
```

- Replace the webpack rule for css:

```
{
   test: /\.s?css$/i,
   use: [MiniCssExtractPlugin.loader, "css-loader", "sass-loader"],
}
```

- Introduction
- Tools
- Step by step guide
- Adding dependencies
- **Architecture**



Code-reuse and architecture

- Create your own reusable classes, functions, and modules
  - Reuse them in different places
- Syntax: import and export
  - Special case: “default export” (optional)
  - Special case: importing CSS
- Functional and/or technical structure:
  - Functional ex.: dashboard.js, navigation.js, …
  - Technical ex.: rest_service.js, animation.js, …

Project

- Check out the project description on Canvas

- **Important!** Your project is to be submitted before **Sunday, 27 February at 23h59**.