Configuration Styles and Assets

webpack
MODULE BUNDLER

SoftUni Team

Technical Trainers

Software University

SoftUni Foundation

Software University

http://softuni.bg

Have a Question?

SoftUni Foundation

sli.do

#webpack

Table of Contents

Loading Styles

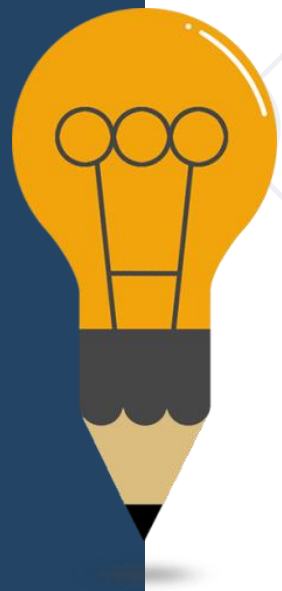
CSS, Less, Sass, PostCSS

Loading Styles

SoftUni Foundation

- Webpack doesn't handle styling out of the box
- In order to **handle styling** in Webpack, you have to use **loaders** and **plugins**
- When a change is made to the CSS Webpack **doesn't force** a full **refresh**. It can patch the CSS without one

Loading CSS



- To load CSS, you need to use **css-loader** and **style-loader** or **MiniCssExtractPlugin**
- Without having an entry pointing to the .css file somehow, webpack is not able to find it
  - src/main.css

```
body { background: cornsilk; }
```

  - src/index.js

```
import "./main.css";
...
```

Less

- Less is a CSS processor packed with functionality
- Using Less doesn't take a lot of effort through webpack as less-loader deals with the heavy lifting
- You should install Less as well given it's a peer dependency of *less-loader*

Loading Less

- The loader supports Less plugins, source maps, and so on
- Installing **less-loader**

```
npm install less-loader --save-dev
```

- Minimal configuration

```
{
  test: /\.less$/,
  use: ["style-loader",
        "css-loader",
        "less-loader"],
},
```

Loading Sass - CSS with superpowers

- Sass is a widely used CSS **preprocessor** - you should use **sass-loader** with it
- Install **node-sass**, as it's a peer **dependency**
- Webpack configuration

```
{
  test: /\.scss$/,
  use: ["style-loader",
        "css-loader",
        "sass-loader"],
},
```

Loading Stylus and Yeticss

- Stylus is yet another example of a CSS processor
- It works well through stylus-loader
- yeticss is a pattern library that works well with it
- To start using yeticss with Stylus, you must import it to one of your app's *.styl* files

```
@import "yeticss"
//or
@import "yeticss/components/type"
```

Stylus Configuration

SoftUni Foundation

```
{
  ...
  module: {
    rules: [
      {
        test: /\.styl$/,
        use: [
          "style-loader",
          "css-loader",
          {
            loader: "stylus-loader",
            options: {
              use: [require("yeticss")],
 ...
```

PostCSS

SoftUni Foundation

- Allows you to **inject** functionality to CSS in through its **plugin system**
- postcss-loader allows using PostCSS with webpack
- Include autoprefixer and precss to your project for this to work

```
{
  test: /\.css$/,
  use: [
    "style-loader",
    "css-loader",
    {
      loader: "postcss-loader",
      options: {
        plugins: () => ([
          require("autoprefixer"),
          require("precss"),
        ]),
      },
    },
  ],
},
```

cssnext

SoftUni Foundation

- PostCSS plugin that allows experiencing the future now with certain restrictions
- You can use it through postcss-cssnext and enable it as follows:

```
{
  use: {
    loader: "postcss-loader",
    options: {
      plugins: () => [require("postcss-cssnext")()],
    },
  },
},
```

Lookups

SoftUni Foundation

- Even though **css-loader** handles relative imports by default, it doesn't touch **absolute imports** (url("/static/img/demo.png"))
- If you rely on absolute imports, you have to **copy** the files to your project
- **copy-webpack-plugin** works for this purpose, but you can also copy the files outside of webpack
- **resolve-url-loader** comes in handy if you use Sass or Less. It adds support for relative imports to the environments

Processing *css-loader* Imports

SoftUni Foundation

- If you want to **process** css-loader **imports** in a specific way, you should set up **importLoaders option** to a **number**

- To import CSS files through the **@import** statement and to **process** the imports through specific loaders, this technique is essential

```
@import "./variables.sass";
```

```
{
  test: /\.css$/,
  use: [
    "style-loader",
    {
      loader: "css-loader",
      options: {
        importLoaders: 1,
      },
    },
    "sass-loader",
...
```

Processing Sass files

SoftUni Foundation

To process the Sass file, you would have to write configuration:

```
{
  test: /\.css$/,
  use: [
    "style-loader",
    {
      loader: "css-loader",
      options: {
        importLoaders: 1,
      },
    },
    "sass-loader",
  ],
},
```

Loading from *node_modules* Directory

SoftUni Foundation

- You can **load** files **directly** from your **node_modules** directory

```
@import "~bootstrap/less/bootstrap";
```

- The tilde character (**~**) tells webpack that it's **not** a **relative** import as by default
- If tilde is included, it performs a lookup **against** node_modules
- With postcss-loader, you can skip using ~ . It can resolve the imports without a tilde

Using Bootstrap through Webpack

SoftUni Foundation

- One option is to point to the **npm version** and perform loader configuration
- The **Sass version** is another option. In this case, you should set **precision** option of sass-loader to at least **8**
- The third option is to go through **bootstrap-loader**. It does a lot more but **allows customization**

Separating CSS and Managing Styles

Why to Separate your CSS?

SoftUni Foundation

- Inlining CSS to JavaScript during development is convenient, but it is not ideal
- This approach doesn't allow cache CSS. You can also get a **F**lash **o**f **U**nstyled **C**ontent (**FOUC**)
- Separating CSS to a file of its own **avoids** the problem by letting the **browser** to manage it **separately**
- It can be potentially **dangerous** to use inline styles within JavaScript in production as it represents an attack vector
- You can use **MiniCSSExtract Plugin**

Managing Styles Outside of JavaScript

SoftUni Foundation

- Referring to **styling** through JavaScript and then **bundling** is the recommended option, it's possible to achieve the same result through an **entry** and **globbing** the CSS files through an entry

```
...
const glob = require("glob");
...
const commonConfig = merge([
  {
    entry: {
      ...
      style: glob.sync("./src/**/*.css"),
    },
    ... },
  ...
```

Managing Styles Outside of JavaScript

- After this type of change, you would **not have** to refer to styling from your application code
- You should get both **style.css** and **style.js**
- If you want **strict** control over the ordering, you can set up a **single** CSS **entry** and then use **@import** to bring the rest to the project through it

Eliminating Unused CSS

PurifyCSS

PurifyCSS

SoftUni Foundation

- Frameworks like Bootstrap tend to come with **a lot of** CSS. Often you use only a **small** part of it. Typically, you bundle even the **unused** CSS. It's possible, however, to **eliminate** the portions you aren't using
- PurifyCSS is a tool that can achieve this by **analyzing** files
- It walks through your code and **figures out** which CSS classes are being used

Enabling PurifyCSS

SoftUni Foundation

- Using PurifyCSS can lead to **significant** savings
- **purifycss-webpack** allows to achieve similar results. You should use the **MiniCssExtractPlugin** with it for the best results

```
npm install glob purifycss-webpack purify-css --save-dev
```

```
const PurifyCSSPlugin = require("purifycss-webpack");

exports.purifyCSS = ({ paths }) => ({
  plugins: [new PurifyCSSPlugin({ paths })],
});
```

Connecting with Configuration

SoftUni Foundation

```
const path = require("path");
const glob = require("glob");

const parts = require("./webpack.parts");
const PATHS = {
  app: path.join(__dirname, "src"),
};
...
const productionConfig = merge([
  ...
  parts.purifyCSS({
    paths: glob.sync(`${PATHS.app}/**/*.js`, { nodir: true }),
  }),
]);
```

- The order matters. CSS extraction has to happen **before** purifying

PurifyCSS Additional Options

SoftUni Foundation

- **minify**: Set to **true** to minify. Default: false
- **output**: Filepath to write purified CSS to. Returns **raw** string if false. D efault: false.
- **info**: Logs info on how much CSS was removed if true. Default: false.
- **rejected**: Logs the CSS rules that were removed if true. Default: false.
- **whitelist** Array of selectors to always leave in. Ex. ['button-active', '*m odal*'] this will leave any selector that includes modal in it and select ors that match button-active. (wrapping the string with *'s, leaves all selectors that include it)

PurifyCSS Additional Options

- **minify**
  - Set to **true** to minify. Default: false
- **output**
  - Filepath to write purified CSS to. **Returns raw string** if false. Default: false.
- **info**
  - Logs info on how much CSS was **removed** if true. Default: false.
- **rejected**
  - Logs the CSS **rules** that were **removed** if true. Default: false.
- Other

Critical Path Rendering

SoftUni Foundation

- Instead of **optimizing** for size, it optimizes for render order and **emphasizes** above-the-fold CSS
- The result is achieved by **rendering** the page and then **figuring out** which rules are **required** to obtain the shown result
- **webpack-critical** and **html-critical-webpack-plugin** implement the technique as a **HtmlWebpackPlugin** plugin
- **isomorphic-style-loader** achieves the same using Webpack and React

Autoprefixing

SoftUni Foundation

- Convenient technique as it **decreases** the amount of work needed while crafting CSS
- Autoprefixing can be enabled through the **autoprefixer** PostCSS plugin
- Writes **missing** CSS definitions based on your minimum **browser** definition
- .**browserslistrc** is a standard file that works with tooling beyond **autoprefixer**

Setting Up Autoprefixing

SoftUni Foundation

- Install ***postcss-loader*** and ***autoprefixer*** first

```
npm install postcss-loader autoprefixer --save-dev
```

- Add a fragment enabling autoprefixing

```
exports.autoprefix = () => ({
  loader: "postcss-loader",
  options: {
    plugins: () => [require("autoprefixer")()],
  },
});
```

Connecting with Configuration

SoftUni Foundation

To connect the loader with CSS extraction, hook it up as follows:

```
const productionConfig = merge([
  parts.extractCSS({
    use: "css-loader",
    use: ["css-loader", parts.autoprefix()],
  }),
  ...
]);
```

Loading Images

url-loader, file-loader etc.

Loading Images

SoftUni Foundation

- Webpack can **inline** assets by using **url-loader**
  - It **emits** your images as base64 **strings** within your JavaScript bundles and **decreases** the number of **requests** needed
- Webpack gives **control** over the inlining process and can **defer** loading to **file-loader**
  - **file-loader** outputs image files and returns **paths** to them instead of inlining
- Be careful **not** to apply both loaders on images at the same time! Use the **include** field for further control if **url-loader** limit isn't enough.

Integrating Images to the Project

SoftUni Foundation

Set up a function as below

```
exports.loadImages = ({ include, exclude, options } = {}) => ({
  module: {
    rules: [
      {
        test: /\.(png|jpg)$/,
        include,
        exclude,
        use: {
          loader: "url-loader",
          options,
...
```

Attaching to the Configuration


```
const productionConfig = merge([
  ...
  parts.loadImages({
    options: {
      limit: 15000,
      name: "[name].[ext]",
    },
  }),
]);
const developmentConfig = merge([
  ...
  parts.loadImages(),
]);
```

Loading SVGs

- Webpack allows a couple ways to load SVGs. However, the easiest way is through *file-loader*

```
{
  test: /\.svg$/,
  use: "file-loader",
},
```

- Refering to your SVG files

```
.icon {
   background-image: url("../assets/icon.svg");
}
```

Useful Loaders

SoftUni Foundation

- **raw-loader** - gives access to the raw SVG content.
- **svg-inline-loader** - goes a step further and eliminates unnecessary markup from your SVGs
- **svg-sprite-loader** - can merge separate SVG files into a single sprite
- **svg-url-loader** - loads SVGs as UTF-8 encoded data urls
- **react-svg-loader** - emits SVGs as React components

Optimizing Images

SoftUni Foundation

- Compression is particularly valuable for production builds as it **decreases** the amount of **bandwidth** required to download your image assets and **speed up** your site or application
- In case you want to compress your images, use **image-webpack-loader**, **svgo-loader** (SVG specific), or **imagemin-webpack-plugin**
  - This type of loader should be applied **first** to the **data**

Utilizing srcset

SoftUni Foundation

- **resize-image-loader** and **responsive-loader** allow you to generate **srcset** compatible collections of images for modern browsers
- srcset gives more **control** to the **browsers** over what images to load and when resulting in **higher performance**

Loading Sprites

- **Spriting** technique allows you to **combine** multiple smaller images into a single image
- webpack-spritesmith converts provided images into a **sprite sheet** and Sass/Less/Stylus mixins
- You have to set up a **SpritesmithPlugin**, point it to target images, and set the name of the generated mixin

```
@import "~sprite.sass";
.close-button {
  sprite($close);
}
.open-button {
  sprite($open);
}
```

Using Placeholders

SoftUni Foundation

- **Image-trace-loader** loads images and exposes the results as image/svg+xml URL encoded data
- It can be used in conjunction with *file-loader* and *url-loader* for **showing** a **placeholder** while the actual image is being loaded
- **lqip-loader** implements a similar idea. It provides a **blurred** image instead of a traced one
- Sometimes getting the only reference to an image isn't enough. image-size-loader emits image dimensions, type, and size in addition to the reference to the image itself

Referencing to Images

SoftUni Foundation

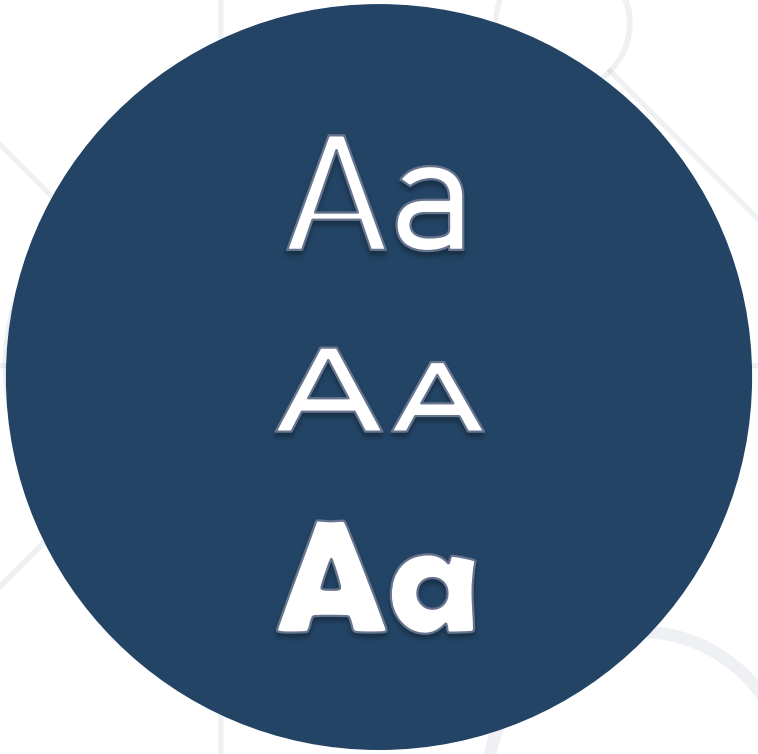
- You can also **refer** to your images within the code. In this case, you have to import the files **explicitly**

```
import src from "./file.png";

// Use the image in your code somehow now
const Profile = () => <img src={src} />;
```

- If you are using React, then you use **babel-plugin-transform-react-jsx-img-import**
- It's also possible to set up **dynamic imports**

```
const src = require(`./files/${file}`);`.
```



Loading Fonts

Loading Fonts

SoftUni Foundation

- If you're currently using Webpack to manage your CSS, importin g font files won't work without a little bit of **extra** configuration
- Loading fonts is **similar** to loading other **assets**. You have to consider the **browsers** you want to support
- The same **techniques** as for **images** apply. You can choose to **inline** small fonts while bigger ones are **served** as **separate** assets

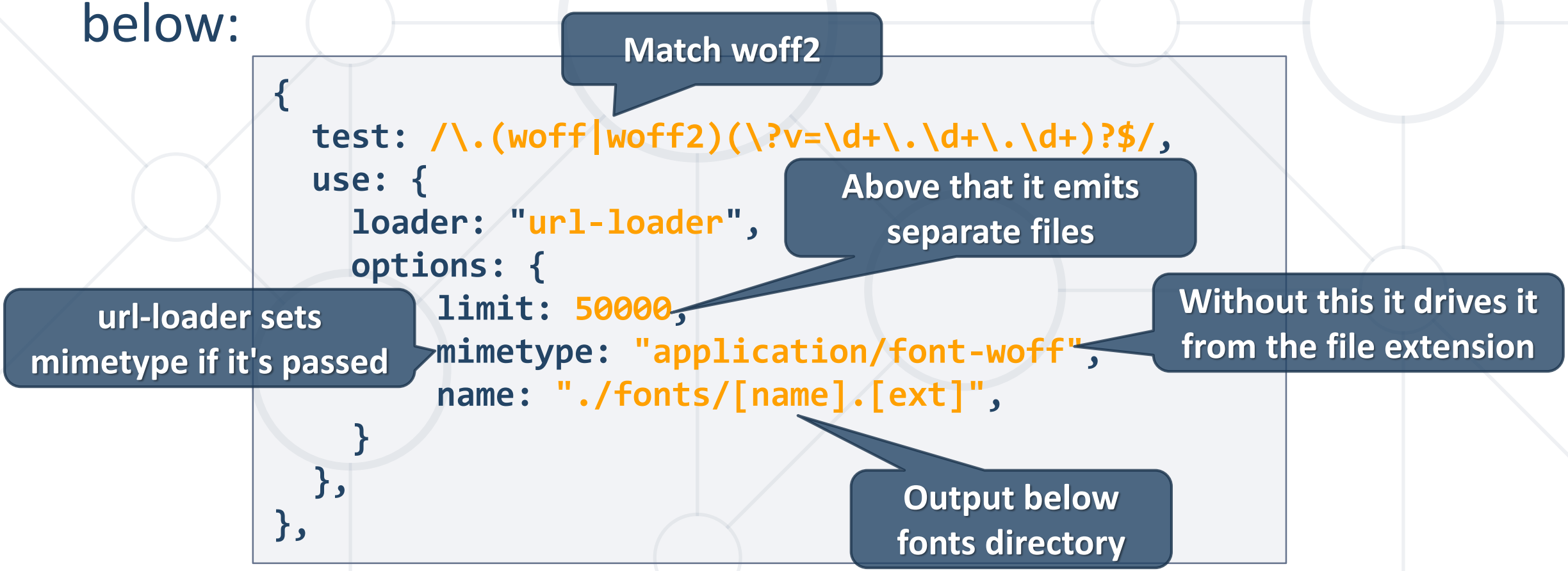
Choosing One Format

- If you exclude Opera Mini, all browsers support the **.woff** format. Its newer version, **.woff2**, is widely supported by modern browsers and can be a good **alternative**
- Going with **one** format, you can use a similar setup as for **images** and rely on both **file-loader** and **url-loader** while using the **limit** option

```
{
  test: /\.woff$/,
  use: {
    loader: "url-loader",
    options: {
      limit: 50000,
...
```

Choosing One Format

- A more elaborate approach to achieve a similar result that includes **.woff2** and others would be to end up with the code as below:



Supporting Multiple Formats

SoftUni Foundation

- If you want to make sure the site **looks good** on a **maximum** amount of browsers, you can use **file-loader** and forget about **inlining**
- It's a trade-off as you get **extra** requests

```
{
  test: /\.(ttf|eot|woff|woff2)$/,
  use: {
    loader: "file-loader",
    options: {
      name: "fonts/[name].[ext]",
. . .
```

Supporting Multiple Formats

SoftUni Foundation

- The way you write your CSS definition **matters**
- To make sure you are getting the **benefit** from the newer formats, they should become **first** in the **definition**

```
@font-face {
  font-family: "myfontfamily";
  src: url("./fonts/myfontfile.woff2") format("woff2"),
    url("./fonts/myfontfile.woff") format("woff"),
    url("./fonts/myfontfile.eot") format("embedded-opentype"),
    url("./fonts/myfontfile.ttf") format("truetype");
    /* Add other formats as you see fit */
}
```

Manipulating file-loader Output Path

- **file-loader** allows **shaping** the output
- This way you can output your fonts below **fonts/**, images below **images/**, and so on over using the **root**
- It's possible to manipulate **publicPath** and override the **default** per loader definition

Manipulating file-loader Output Path

SoftUni Foundation

```
{
  // Match woff2 and patterns like .woff?v=1.1.1.
  test: /\.woff2?(\?v=\d+\.\d+\.\d+)?$/,
  use: {
    loader: "url-loader",
    options: {
      limit: 50000,
      mimetype: "application/font-woff",
      name: "./fonts/[name].[ext]", // Output below ./fonts
      publicPath: "../", // Take the directory into account
    },
  },
},
```

Other Font Loaders and Plugins

SoftUni Foundation

- webfonts-loader
  - Bundles SVG based fonts
- google-fonts-webpack-plugin
  - Downloads Google Fonts to webpack build directory or connect to them using a CDN
- iconfont-webpack-plugin
  - Designed to simplify loading icon based fonts. It inlines SVG references within CSS files
- Other

Live Exercise

Summary

- Styling
  - To load CSS, you need to use **css-loader** and **style-loader**
  - Separating CSS to a file of its own avoids **FOUC**
  - PurifyCSS is a tool that **eliminates** CSS this by **analyzing** files
  - Autoprefixnig
- Loading Assets
  - Webpack allows you to **inline images** within your **bundles** when needed
  - Loading **fonts** is similar to loading other **assets**

Questions?

SoftUni

Software University | SoftUni Svetlina | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids

https://softuni.bg/trainings/2469/webpack-4-august-2019

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

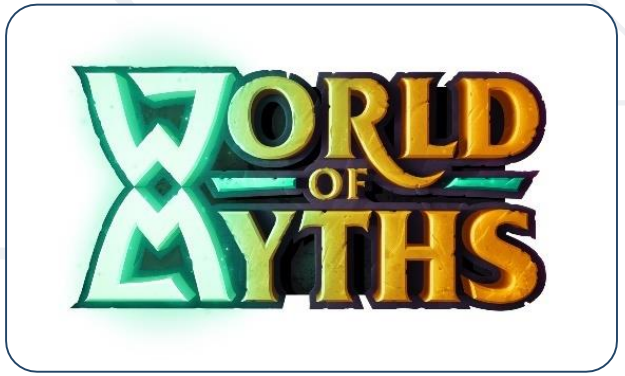
INFRAGISTICS®

LIEBHERR

æternity

SoftUni Organizational Partners

SoftUni Foundation



Trainings @ Software University (SoftUni)

SoftUni Foundation

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license