



Faculteit Bedrijf en Organisatie

Titel

Steven Stevens

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Jan Janssens
Co-promotor:
Piet Pieters

Instelling: —

Academiejaar: 2020-2021

Eerste examenperiode

Faculteit Bedrijf en Organisatie

Titel

Steven Stevens

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Jan Janssens
Co-promotor:
Piet Pieters

Instelling: —

Academiejaar: 2020-2021

Eerste examenperiode

Woord vooraf

Samenvatting

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus.

Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Inhoudsopgave

1	Inleiding	13
1.1	Probleemstelling	13
1.2	Onderzoeksvraag	14
1.3	Onderzoeksdoelstelling	14
1.4	Opzet van deze bachelorproef	14
2	State of the art	15
2.1	History	16
2.1.1	Script tags	16
2.1.2	Immediately invoked function expressions	16
2.1.3	CommonJS	17
2.1.4	ECMAScript Modules	17

2.2	Module bundlers	18
2.2.1	Bundled development	19
2.2.2	Unbundled development	20
3	Methodologie	23
4	Conclusie	25
A	Onderzoeksvoorstel	27
A.1	Introduction	27
A.2	State-of-the-art	28
A.3	Methodology	28
A.4	Anticipated results	28
A.5	Anticipated conclusions	28
A.6	References	29
	Bibliografie	31

Lijst van figuren

Lijst van tabellen

1. Inleiding

De inleiding moet de lezer net genoeg informatie verschaffen om het onderwerp te begrijpen en in te zien waarom de onderzoeksvraag de moeite waard is om te onderzoeken. In de inleiding ga je literatuurverwijzingen beperken, zodat de tekst vlot leesbaar blijft. Je kan de inleiding verder onderverdelen in secties als dit de tekst verduidelijkt. Zaken die aan bod kunnen komen in de inleiding (Jowkens, 2009):

- context, achtergrond
- afbakenen van het onderwerp
- verantwoording van het onderwerp, methodologie
- probleemstelling
- onderzoeksdoelstelling
- onderzoeksvraag
- ...

1.1 Probleemstelling

Uit je probleemstelling moet duidelijk zijn dat je onderzoek een meerwaarde heeft voor een concrete doelgroep. De doelgroep moet goed gedefinieerd en afgeleid zijn. Doelgroepen als “bedrijven,” “KMO’s,” systeembeheerders, enz. zijn nog te vaag. Als je een lijstje kan maken van de personen/organisaties die een meerwaarde zullen vinden in deze bachelorproef (dit is eigenlijk je steekproefkader), dan is dat een indicatie dat de doelgroep goed gedefinieerd is. Dit kan een enkel bedrijf zijn of zelfs één persoon (je co-promotor/opdrachtgever).

1.2 Onderzoeksvraag

Wees zo concreet mogelijk bij het formuleren van je onderzoeksvraag. Een onderzoeksvraag is trouwens iets waar nog niemand op dit moment een antwoord heeft (voor zover je kan nagaan). Het opzoeken van bestaande informatie (bv. “welke tools bestaan er voor deze toepassing?”) is dus geen onderzoeksvraag. Je kan de onderzoeksvraag verder specificeren in deelvragen. Bv. als je onderzoek gaat over performantiemetingen, dan

1.3 Onderzoeksdoelstelling

Wat is het beoogde resultaat van je bachelorproef? Wat zijn de criteria voor succes? Beschrijf die zo concreet mogelijk. Gaat het bv. om een proof-of-concept, een prototype, een verslag met aanbevelingen, een vergelijkende studie, enz.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. State of the art

A modern website is created with three main technologies: HTML for content, CSS for styling, and Javascript to make the page interactive. Those three can all be linked, put on a server somewhere and the end-user can load a working website. Even though the large web platforms of today, like Facebook and Youtube, are all based on these technologies, a lot more ingredients are needed to make them work.

If we were to create a static, basic site, the three main technologies of the web are all you need. But it's a whole different story for a developer that wants to create a more complex site, a reactive site, a site that uses open-source packages or just wants to make his own job a little bit easier. There are thousands of libraries, packages and frameworks that enable web development on a whole new level and easier than ever before. However this does pose a new problem.

If we were to create a web application with a single Javascript file and no dependencies i.e no other files that are linked to that Javascript file, link it to our index.html and throw in some CSS, our site would work fine. But what if we introduce a second Javascript file and reference it in our other file. What if we want to install and use an open-source package downloaded from a package-manager like NPM? What if we want to use SASS to extend CSS? This all wouldn't work out of the box. The browser just isn't able to figure out how to stitch all the different pieces together. We need something that will bundle all the different Javascript files and dependencies, something that understands and correctly loads the SASS files or any other file for that matter. We need a module bundler. In essence, they take all the different source files in a project and put them into a single output file that the browser understands.

2.1 History

To understand the rest of this paper, you'll need to know what a module is. Modular programming breaks a program up into chunks based on functionality and often in separate files. These chunks are called modules. Modules are then linked together to form an application.

Node.js, a Javascript runtime for computers and servers, has supported modular programming since the beginning using CommonJS. However support for modules on the web has been slow to arrive. The first module bundlers only came around after 2014. To understand why module bundlers exist, we should first know how web applications were built before them and what problem they solved.

2.1.1 Script tags

Javascript can be linked to, or written directly in, the HTML file of a site using script-tags. Knowing this, we could use a different tag and corresponding Javascript file for each use case. Let's say we have a file with all the code for authenticating a user and another one for general events (button clicks, ... This is fine when we only have two files that aren't that big, but introduces network bottlenecks when scaled to a larger application. The same is true if we were to put all our code in one large JS file and link it to the HTML. The global scope would also get polluted with our custom functions. As some, or all, the functions are available on the global scope, this could introduce security risks and name collisions.

```
<!DOCTYPE html>

<html lang="en">
  <body>
    <h1>Hello world!</h1>
    <script src="js/authentication.js"></script>
    <script src="js/main.js"></script>
  </body>
</html>
```

2.1.2 Immediately invoked function expressions

An Immediately invoked function expression or IIFE is a function that runs as soon as it's defined. Because each IIFE declares a local scope, it solves the problem of polluting the global one. The use of IFFEs led to so-called task runners: they concatenate all your project files together. The big drawback of task runners is that when one file is changed, the whole project has to be rebuilt. You are also required to manually define all the dependencies up front. They make it easier to reuse functions and whole scripts but do nothing for the build output. You can still end up with a very large Javascript file that the user has to download.

```
(function () {
  let firstVariable;
  let secondVariable;
  // ...
```

```
})( );  
  
// firstVariable and secondVariable will be discarded after the  
// function is executed.
```

2.1.3 CommonJS

Before 2009, Javascript ran only in a browser. Node.js introduced a Javascript runtime that could run on computers and servers. This introduced a new set of challenges. As Javascript wasn't run in the browser and therefore no HTML script-tags were around, how could those applications load new chunks of code?

CommonJS introduced the `require` function in Javascript. With it, everything that an external module exports, can be imported. Reusable code can now be imported from any other Javascript file in a project. It makes implementing dependency management easy to understand.

All this came with a big catch: It worked, and still works, great for Node.js applications but it isn't an official feature of Javascript and therefore browsers don't support it. As commonJS doesn't actually bundle the code, web browsers can't make sense of the different modules that are imported. Something has to do it for them.

```
//myFunctions.js  
  
const calculateTotal = (a, b) => a + b;  
  
module.exports = calculateTotal;  
  
//main.js  
  
const calculateTotal = require("myFunctions.js");  
  
console.log(calculateTotal(1, 3)); /*Logs 4*/
```

2.1.4 ECMAScript Modules

CommonJS wasn't an official feature of Javascript. ECMAScript (=Javascript) did introduce its own module system in version 6. ECMAScript Modules or ESM, accomplish the same goals as CommonJS, but with a different syntax. Now modern browsers can make sense of modular applications that only use ESMs.

```
//myFunctions.js  
  
export const calculateTotal = (a, b) => a + b;  
  
//main.js  
  
import { calculateTotal } from "myFunctions.js";  
  
console.log(calculateTotal(1, 3)); /*Logs 4*/
```

2.2 Module bundlers

Developers are always seeking ways to make their lives easier. They wanted to import whatever type of module or any asset for that matter into their project and ship it in a smaller output file than the source to the end-user. This is why the module bundler was born.

The most essential function of a module bundler is following all imports of a project, that can contain many files, and bundling them into a single file called the bundle. They also minify that bundle to be as small as possible, without affecting its functionality. It does this by removing comments, white-spaces, new lines, ...

Unbundled file: 147 bytes

```
const array = ["Hello", "my", "name", "is", "Maxim"];

//Loop over all elements and print
for (const element of array) {
  console.log(element);
}
```

Bundled file: 97 bytes

```
const array=["Hello","my","name","is","Maxim"];for(const element of array){console.log(element);}
```

All module bundlers around today share these functions along with common concepts. We'll look at the 2 most important ones.

Tree shaking

Tree shaking is the process of removing dead code. When a module is imported, perhaps only a part of that module is needed. Maybe only one function of that module is used in the project. With tree shaking, the rest of that module that isn't used, is removed.

Code splitting

Code splitting can be used to divide up the output bundle that is created into smaller files. The partial bundles are then loaded in parallel or when needed. For example: take a website with multiple pages. If the code isn't split, all the code of all the pages will be contained in a single file and downloaded by the user when the site is requested. In many cases, like a small project, this is fine. That's why it's optional. For larger applications however, it could introduce network bottlenecks. If that's the case, the code can be split into a single or multiple files per page. Then the smaller files are only loaded when the corresponding page is requested.

2.2.1 Bundled development

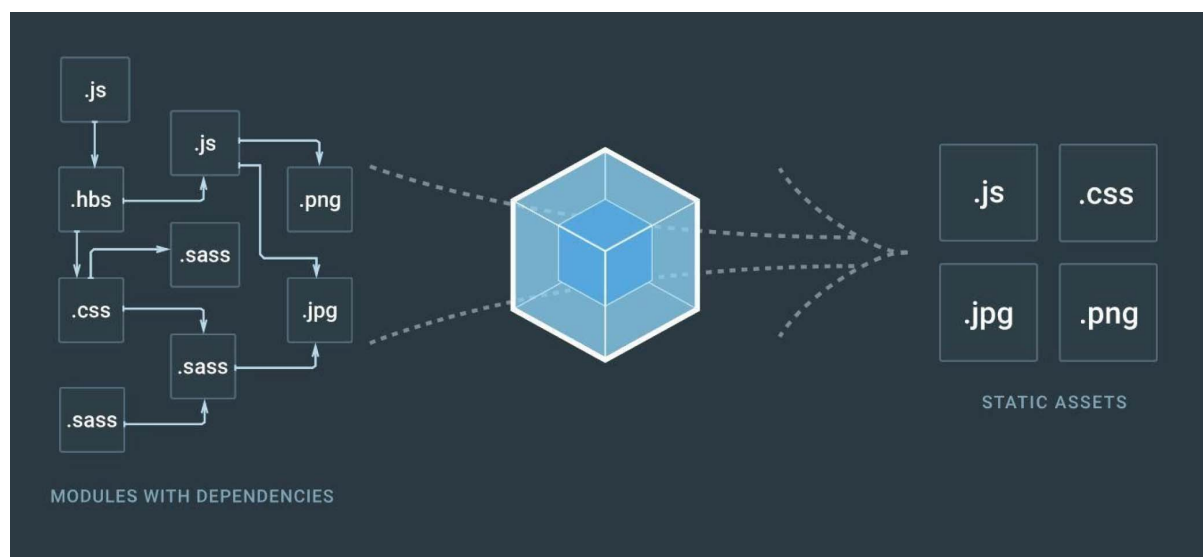
In this section, examples of module bundlers that use bundled development will be discussed. Almost every Javascript web bundler is based on the concepts of bundled development. The alternative and more modern approach is unbundled development. The differences and examples will be explained in the next section.

Webpack

Webpack is the most popular module bundler in the world according to NPM downloads. This has much to do with the fact that it's been around the longest. It comes pre-installed in many web frameworks like create-react-app and Next.js. This way, it's used by many without even knowing it. It comes with an optional built in development server that makes setting up a local development environment easier.

Webpack runs on Node.js. It can do its job without any configuration, however is very configurable if needed. It supports the module types discussed above and more. Because Webpack only understands Javascript and JSON files out of the box, Loaders are used to allow processing of other file types and convert them into valid modules. Using Loaders, other types of modules or even assets like images can be imported and processed by WebPack. On top of that, Webpack can also be extended with plugins. They allow for a wide range of extra functionality like bundle optimization.

The function of a module bundler has already been discussed. But how does Webpack achieve this? Whenever one file depends on another in a project, Webpack sees this dependency and puts it into something called a dependency graph.



This graph is built recursively. When it is time to build the application, it uses the graph to piece all the files together into one output file that is then shipped to the browser. Webpack does this both in development as in production.

Parcel

Parcel is another Javascript bundler. However it doesn't run on Node.js, instead the compiler it uses is built with Rust. Rust is a compiled programming language. Without going into too much detail, this means that Rust code is compiled directly to machine code resulting in faster performance. It does many of the same things that Webpack does without any configuration. When it was released, the main feature was that it didn't need a configuration file and Webpack did. Now however, Webpack can also do its job without one. While a no-configuration approach is great for smaller projects, it isn't feasible in a large application.

Rollup

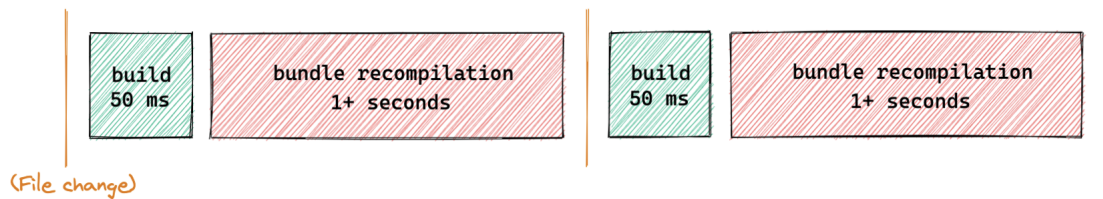
...

2.2.2 Unbundled development

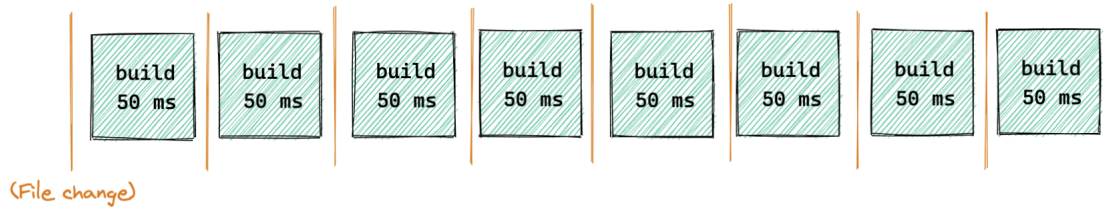
The examples we looked at so far are all module bundlers that use the concepts of bundled development. One very important thing to note is the word development. The differences between both approaches lies only in the development phase. The advantages of unbundled development are only noticeable when the developer or the development team is actually developing the application. When it is time to make a production build that the end-user will be able to use, the following unbundled development build tools still use the same module bundlers discussed in the previous section. So in essence, we can't call the unbundled development build tools module bundlers, because they don't bundle anything. They do use module bundlers.

Because most browsers now support ESM, bundling all Javascript modules of a project into one output file isn't necessary. All non-Javascript modules do have to be translated or built to a valid Javascript module using a built tool (in WebPack this is called a Loader). In bundled development, when the development server is started, the whole project first has to be built and bundled; When a file is changed that file is rebuilt and then the whole project is again rebundled. In unbundled development a file is only built when it's requested, meaning very fast startup time of the server. When a file is built, it is cached indefinitely. The browser will never have to download a file twice until it changes. When a file does change, only that single file has to be rebuilt.

Bundled (ex: Webpack)



Unbundled (Snowpack)



All this results in very fast development performance compared to bundled development tools like WebPack. However, as mentioned before, unbundled development tools still use traditional module bundlers to make a production build. It does this because the advantages of Tree-shaking, a single output file and other features of module bundlers still hold up in the production phase: Tree-shaking results in a smaller output file; Not all modules are written with ESM; and the list goes on. We'll now look at two examples of unbundled development tools.

Snowpack

...

Vite

...

3. Methodologie

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetur quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Maecenas non massa. Vestibulum pharetra nulla at lorem. Duis quis quam id lacus dapibus interdum. Nulla lorem. Donec ut ante quis dolor bibendum condimentum. Etiam egestas

tortor vitae lacus. Praesent cursus. Mauris bibendum pede at elit. Morbi et felis a lectus interdum facilisis. Sed suscipit gravida turpis. Nulla at lectus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent nonummy luctus nibh. Proin turpis nunc, congue eu, egestas ut, fringilla at, tellus. In hac habitasse platea dictumst.

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed, ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

4. Conclusie

Curabitur nunc magna, posuere eget, venenatis eu, vehicula ac, velit. Aenean ornare, massa a accumsan pulvinar, quam lorem laoreet purus, eu sodales magna risus molestie lorem. Nunc erat velit, hendrerit quis, malesuada ut, aliquam vitae, wisi. Sed posuere. Suspendisse ipsum arcu, scelerisque nec, aliquam eu, molestie tincidunt, justo. Phasellus iaculis. Sed posuere lorem non ipsum. Pellentesque dapibus. Suspendisse quam libero, laoreet a, tincidunt eget, consequat at, est. Nullam ut lectus non enim consequat facilisis. Mauris leo. Quisque pede ligula, auctor vel, pellentesque vel, posuere id, turpis. Cras ipsum sem, cursus et, facilisis ut, tempus euismod, quam. Suspendisse tristique dolor eu orci. Mauris mattis. Aenean semper. Vivamus tortor magna, facilisis id, varius mattis, hendrerit in, justo. Integer purus.

Vivamus adipiscing. Curabitur imperdiet tempus turpis. Vivamus sapien dolor, congue venenatis, euismod eget, porta rhoncus, magna. Proin condimentum pretium enim. Fusce fringilla, libero et venenatis facilisis, eros enim cursus arcu, vitae facilisis odio augue vitae orci. Aliquam varius nibh ut odio. Sed condimentum condimentum nunc. Pellentesque eget massa. Pellentesque quis mauris. Donec ut ligula ac pede pulvinar lobortis. Pellentesque euismod. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent elit. Ut laoreet ornare est. Phasellus gravida vulputate nulla. Donec sit amet arcu ut sem tempor malesuada. Praesent hendrerit augue in urna. Proin enim ante, ornare vel, consequat ut, blandit in, justo. Donec felis elit, dignissim sed, sagittis ut, ullamcorper a, nulla. Aenean pharetra vulputate odio.

Quisque enim. Proin velit neque, tristique eu, eleifend eget, vestibulum nec, lacus. Vivamus odio. Duis odio urna, vehicula in, elementum aliquam, aliquet laoreet, tellus. Sed velit. Sed vel mi ac elit aliquet interdum. Etiam sapien neque, convallis et, aliquet vel, auctor non, arcu. Aliquam suscipit aliquam lectus. Proin tincidunt magna sed wisi. Integer blandit

lacus ut lorem. Sed luctus justo sed enim.

Morbi malesuada hendrerit dui. Nunc mauris leo, dapibus sit amet, vestibulum et, commodo id, est. Pellentesque purus. Pellentesque tristique, nunc ac pulvinar adipiscing, justo eros consequat lectus, sit amet posuere lectus neque vel augue. Cras consectetur libero ac eros. Ut eget massa. Fusce sit amet enim eleifend sem dictum auctor. In eget risus luctus wisi convallis pulvinar. Vivamus sapien risus, tempor in, viverra in, aliquet pellentesque, eros. Aliquam euismod libero a sem.

Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introduction

A module bundler is one of the most essential technologies in modern web development. Nobody makes web applications with plain HTML, CSS and JS anymore. All the fancy new tools, frameworks and libraries for a web developer to use, like Angular or React, won't work in the browser without being bundled. So the choice of which bundler to use is very important. While Webpack is the most popular^{1,2,3}. Gebruik dit als de naam van de auteur geen onderdeel is van de zin., there are many others that claim to do the job better. How does Webpack compare to its competitors? Are their claims valid? If so, why is Webpack still the most popular? Thus to summarize: does it make sense for development teams at companies, stand-alone developers or just hobby coders to bundle their code with the same tool as 5 years ago?

Many developers give much thought about which framework to use, which back-end and so forth. The module bundler however often doesn't get that much thought. This is because it's less attractive and in many frameworks comes pre-installed. But as your web app can't run without it, it's one of the most important choices you make. The goal of this paper is to demystify the module bundler and compare the most popular options. Not just in terms of speed and module size, but also its plugins, developer experience,

A.2 State-of-the-art

Much can already be found about module bundlers online. On the website or GitHub repository of the bundler itself, articles written by third-parties and video's on YouTube. While they contain very useful information, they lack an in-depth analysis of the differences between the major players and what that means for the developer and the output product. This is an important goal of the paper.

A.3 Methodology

Firstly, the most popular bundlers will be campered theoretically. How they work and how they may differ. To compare them in practice will require code-bases of many sizes. Luckily there are many open-source codebases on GitHub to chose from. Why many sizes, you ask? Because it could be interesting to see how module bundlers deal with smaller projects compared to larger ones. To see if the advantages of one fade when the project size in- or decreases. It could also be interesting to compare them based on the framework, like Angular or Create-React-App. Those frameworks ship with pre-configured bundlers so it remains to be seen if it's even possible to replace those with others. It's something that has to tested.

Secondly and maybe more important: what about already existing, operational projects? For example: a company that already has a site or some kind of application on the web bundled with WebPack. Is it possible to just switch to another module bundler? What hurdles have to be overcome to do so and what are the benefits?

Testing many code-bases and observing the differences in numerous categories is necessary. One important category is the developer experience: how easy is it to install? How simple or complex are the config file and plugins. There are a lot of things to consider.

A.4 Anticipated results

Many module bundlers claim speedier builds and smaller bundle sizes. So that's an easy prediction to make. Wether that remains true with projects of any size remains to be seen. Will those improvements be great enough to warrant the competitor to be used over Webpack? Developer experience and performance in development mode are defining factors as well. Obviously the former will be quite subjective so that's an important factor to note.

A.5 Anticipated conclusions

It's hard to predict what the conclusion will be. That's the most important reason why I want to conduct this research. Webpack has served us well but as the tech world shifts

more and more towards applications written with web technologies, the time of newer approaches may have come.

A.6 References

1. Search results module bundlers. (n.d.). Google Trends. Retrieved October 2021, from <https://trends.google.com>
2. Nalakath, N. (2021, February 2). Module Bundlers and their role in web development. | Better Programming. Medium. Retrieved October 2021, from <https://betterprogramming.pub/javascript-module-bundlers-2a1e9307d057>
3. The State of JavaScript 2018: Other Tools. (2018). StateOfJs. Retrieved October 2021, from <https://2018.stateofjs.com/other-tools/>

Bibliografie

Jowkens. (2009). CTO Roundtable: Cloud Computing. *Communications of the ACM*, 52(8), 50–56.