A comparative analysis of Webpack and Vite as build tools for JavaScript

Tuan Anh Nguyen

Haaga-Helia University of Applied Sciences
Business Information Technology

Bachelor’s Thesis
2024

Abstract

Author(s)
Tuan Anh Nguyen

Degree
Bachelor of Business Information Technology

Report/Thesis Title
A comparative analysis of Webpack and Vite as build tools for JavaScript

Number of pages and appendix pages
32 + 0

There are three main categories of web development: front-end, back-end and full-stack. For front-end developments, build tools are used to boost performance and workflow. Vite and Webpack are two popular build tools for JavaScript projects. The aim of the study is to make a analysis of Vite and Webpack in different aspects: technical concepts, configuration files, performance in measurable metrics and support for JavaScript frameworks. The study includes literature reviewing, qualitative research to evaluate support for frameworks and factors affecting configuration complexity and in Vite and Webpack, quantitative research to measure build time, build size in Vite and Webpack. The research targets Vite and Webpack as JavaScript build tools. Backend build tools and unpopular JavaScript frameworks are ruled out.

The theoretical framework includes knowledge about front-end development and JavaScript, JavaScript libraries and frameworks React, Vue, bundler and its features, build tool, technical details of Webpack and Vite. From the theoretical framework, Webpack and Vite shares similarities in minification, transpiling, source map, environment variable, hot module replacement, tree shaking. Webpack bundles the entire code and assets of the application in development and production build. Vite uses esbuild to pre-bundle dependencies and serve source code on demand via ECMAScript modules. In production build, Vite uses Rollup to bundles code.

Case studies were gathered and made by the author to collect configuration files, build time and bundle size of projects in Vite and Webpack. Using esbuild in development and Rollup in production, Vite was faster than Webpack in development and production build time. The results of build size in production varied in case studies. The configuration files showed that Vite and Webpack supports JavaScript frameworks React and Vue by allowing usage of official as well as third-party plugins. Influencing factors of the configuration file in Vite are plugins, mode of build, port number and environment variables. Those factors in Webpack are entry/output path, loaders options, plugins and resolver.

Key words
Build tool, Bundler, Webpack, Vite, front-end development, JavaScript

Table of Contents

1 Introduction

Web development involves in building and maintaining websites and web applications. Building a website is a process of constructing the functional aspects of a website or web application. It encompasses a wide range of tasks including coding the backend logic that runs on the server, developing the front-end interface that users interact with, and integrating these elements with databases to create an operational application which works over the internet. (Geeksforgeeks 2024, Web Development)

There are three main types of web development: front-end, back-end and full-stack. Front-end development involves building the parts of a website that users interact with directly. It is primarily concerned with the look and feel of a website. Back-end development focuses on the server-side of a website, dealing with the database and server logic. It involves writing APIs, creating libraries, and working with system components without user interfaces or even systems that communicate directly with the front-end. Full-stack development combines both front-end and back-end development. (Hubspot 2024, The Beginner's Guide to Website Development)

Developers use various tools matched their requitements to enhance workflow and performance. For front-end development, build tools are engineered to manage a range of activities, such as converting code from one language to another, combining several files into a single file, compressing code to decrease its size, and automating the testing procedures. (Yurchyshyn 2023)

According to State of JavaScript 2023 survey (State of JavaScript 2023), Vite and Webpack are two of the most popular build tools for JavaScript among Parcel, Rollup, tsc CLI, SWC, esbuild, Turbopack, tsup and Biome. Since 2016 Webpack has been voted to become the most popular build tool overtime. Meanwhile, Vite has become more and more popular and received positive feedback compared to Webpack since 2020.

The purpose of the research is to explore a comparative analysis of Vite and Webpack, focusing on their core technologies, performance, configuration complexity and support for modern frameworks. The result of this research can help both development community and businesses looking forward to optimizing their workflows.

The research methods include literature review on key concepts of Vite and Webpack to understand the core differences between two build tools. Qualitative research is conducted to analyze key factors influencing the complexity of configuration files in Vite and Webpack and explore how Vite and Webpack support modern JavaScript frameworks. Quantitative research is conducted to compare the performance of Vite and Webpack through measurable metrics: build time and bundle size. Case studies on the internet and a case study made by the author will be used to analyzed qualitatively and quantitatively.

The following questions are designed to serve the purpose of the research to explore a comparative analysis of Vite and Webpack:

Q1: What similar and different technologies/concepts are used in Vite and Webpack?

Q2: What are key factors influencing the complexity of modifying a configuration file in Vite and Webpack?

Q3: How do Vite and Webpack support modern JavaScript frameworks?

Q4: What are the performance differences between Vite and Webpack in terms of build time and bundle size?

Table 1. Correlation between research questions, theoretical frameworks and results

Research questions	Theoretical framework (chapter)	Results (chapter)
Q1	2.6	5.1
Q2	2.7	5.2
Q3	2.2	5.2
Q4	2.3	5.3

The thesis focuses on two specific JavaScript build tools: Vite and Webpack. Since the background of the research involves in the evolution of front-end development, backend build tools are not directly related to the research. Vite and Webpack are mainly used in the front-end development. Including backend study will shift the focus out of the main research scope. Analysis of rarely used frameworks and libraries will also be excluded from the thesis. Examining edge cases will broaden the thesis and not focus on the larger part of developers.

2 Theoretical framework

The theoretical framework summarizes knowledge from literatures, blogs, web documents to build a foundation to explain research questions.

2.1 Front-end development and JavaScript

The foundation of a website is constructed by Hypertext Markup Language (HTML) providing definition and structure to web content. In addition to HTML, other technologies like Cascading Style Sheets (CSS) are employed to determine the visual presentation of a web page, while JavaScript is used to enhance its functionality and interactive features. (MDN web docs 2024, HTML: HyperText Markup Language)

JavaScript is a lightweight interpreted programming language which is the most popular for web pages and non-browser environments (MDN web docs 2024, JavaScript). In front-end development, JavaScript is crucial to interactive web experience. Developers use JavaScript to create dynamic and responsive user interfaces. JavaScript combined with HTML and CSS can handle many tasks in front-end development such as validating forms, fetching data from servers asynchronously. This versatility makes JavaScript popular to create modern, highly interactive front-end applications.

2.2 JavaScript frameworks and libraries

JavaScript frameworks are vital to contemporary front-end web development. Developers utilize tested tools, which are provided by JavaScript frameworks, to build scalable and interactive web applications. (MDN web docs 2024, Understanding client-side JavaScript frameworks)

A JavaScript library provides a collection of pre-written code and can be used when necessary. JavaScript frameworks offer a set of tools that assist in structuring and organizing your website or application. (General Assembly 2021)

2.2.1 React

React or React.js is a popular front-end library for modern web development within JavaScript community. It is an open-source JavaScript library developed by Meta and React community to build interactive user interfaces. By using React, a user interface can be made of components which are JavaScript functions. (React 2024)

React offers developers creating single-page application (SPA). A single-page application loads a single HTML file initially. When users interact with the single-page application, specific sections or content of the webpage are updated accordingly. A virtual DOM (Document Object Model), a copy of the actual DOM, is used to track changes efficiently in React. When data state changes, React updates the virtual DOM, then compares it to the actual DOM to identify and apply only the necessary updates. (Herbert 2023)

2.2.2 Vue.js

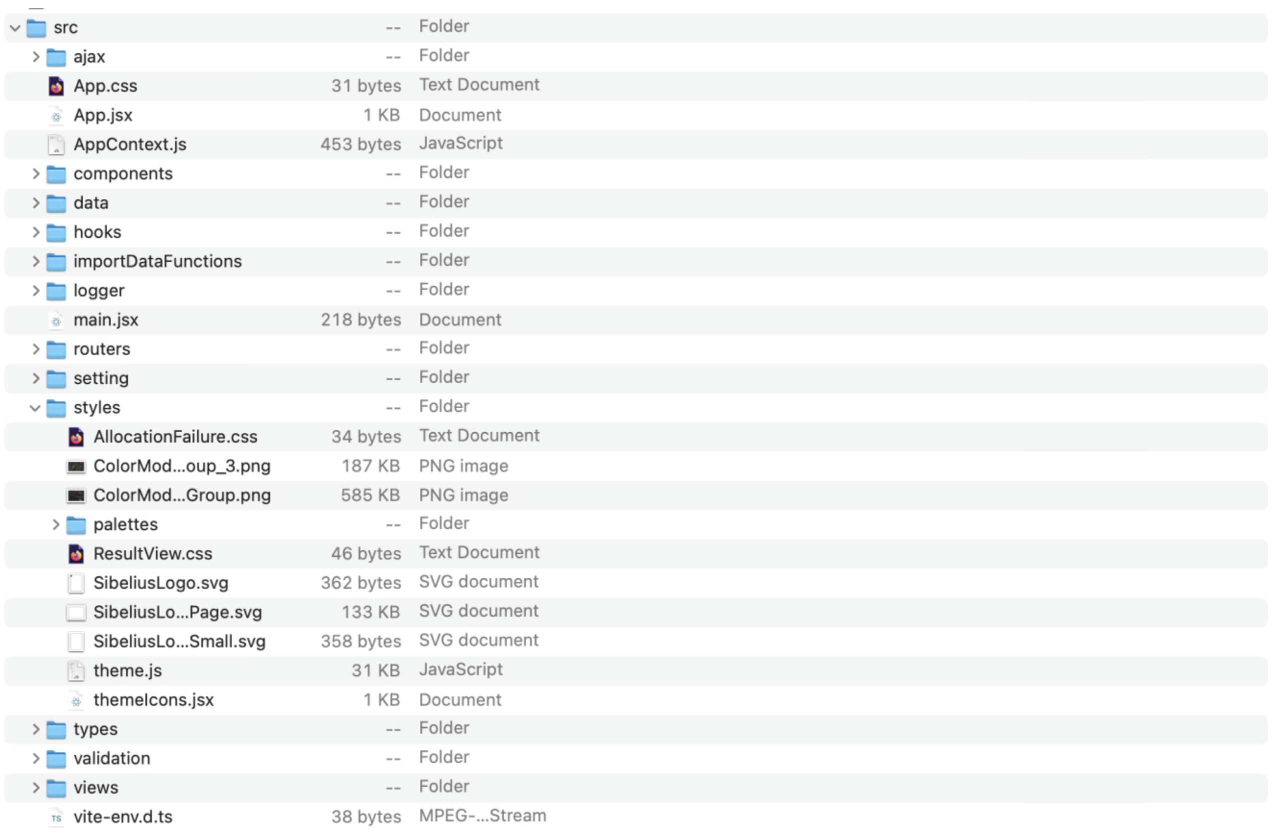
Vue is a JavaScript progressive framework, built on HTML CSS and JavaScript, for building user interfaces by providing a declarative component-based programming model. Vue is developed by Evan You and the Core team. Two main features of Vue are declarative rendering and reactivity. With declarative rendering, Vue enhances standard HTML with a template syntax that lets JavaScript state describe HTML output. Reactivity allows the DOM to be updated whenever there are changes in the state. (Vue.js Introduction 2024)

2.3 Bundler

A bundle commonly refers to a single file hosted on web server that includes all the JavaScript necessary for the application. It is usually compressed to minimize the download time, ensuring that the application loads quickly and efficiently in browsers. (MDN web docs 2024, Package management basics)

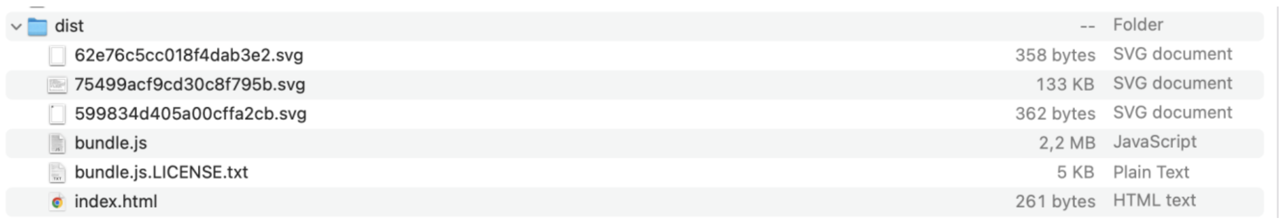
Some features of a JavaScript bundler are module bundling, minification, transpiling, source map, environment variables, tree shaking.

In a JavaScript program running in a web browser, code is written in multiple files called modules. Some external libraries also provide functions as modules. Browser needs to load many modules to run the program. By using code-bundler tool, all needed modules are combined into a single file of code. The bundled file can be used by browser even if the browser does not support modules. Although ECMAScript 6 (ES 6), a standard for scripting languages including JavaScript, modules are wildly supported by web browsers nowadays, developers still use code bundler to enhance experience in production. User experience is improved when a website initially loads a single, medium-sized code bundle rather than multiple smaller modules. (Flanagan 2020, 642)



src	--	Folder
ajax	--	Folder
App.css	31 bytes	Text Document
App.jsx	1 KB	Document
AppContext.js	453 bytes	JavaScript
components	--	Folder
data	--	Folder
hooks	--	Folder
importDataFunctions	--	Folder
logger	--	Folder
main.jsx	218 bytes	Document
routers	--	Folder
setting	--	Folder
styles	--	Folder
AllocationFailure.css	34 bytes	Text Document
ColorMod...oup_3.png	187 KB	PNG image
ColorMod...Group.png	585 KB	PNG image
palettes	--	Folder
ResultView.css	46 bytes	Text Document
SibeliusLogo.svg	362 bytes	SVG document
SibeliusLo...Page.svg	133 KB	SVG document
SibeliusLo...Small.svg	358 bytes	SVG document
theme.js	31 KB	JavaScript
themeIcons.jsx	1 KB	Document
types	--	Folder
validation	--	Folder
views	--	Folder
vite-env.d.ts	38 bytes	MPEG-...Stream

Figure 1. Src folder before bundling



dist	--	Folder
62e76c5cc018f4dab3e2.svg	358 bytes	SVG document
75499acf9cd30c8f795b.svg	133 KB	SVG document
599834d405a00cffa2cb.svg	362 bytes	SVG document
bundle.js	2,2 MB	JavaScript
bundle.js.LICENSE.txt	5 KB	Plain Text
index.html	261 bytes	HTML text

Figure 2. Example of bundled code

As figure 1 shows, the src folder contains multiple files of code. Figure 2 shows that after bundling process, the bundled files containing one single file of JavaScript code are in dist folder.

Minification, or minimization, involves stripping out all redundant characters from JavaScript or CSS code without changing its operational effect. This process eliminates spaces, comments, and semicolons, and includes the adoption of shorter names for variables and functions. As a result, the file becomes much more compact, reducing its overall size. (Cloudflare 2024)

Transpiling is the process of transforming code written with the recent ECMAScript specifications into a format compatible with older browsers. Unlike traditional compiling, which converts code to a lower-level language for machine execution (as seen with programming languages like C), transpiling modifies the code to another form of the same abstraction level. (MDN web docs 2024, Handling common JavaScript problems)

Source map allows developers to maintain a reference back to the original source code, making debugging easier, especially with minified or compiled code. (FireFox 2024)

According to Webpack (Webpack 2024, Environment Variables), bundlers can differentiate between development and production environments, applying different settings or optimizations accordingly. The objectives for development and production builds are significantly different. Robust source mapping, live reloading and HMR are prioritized in development while creating minified bundles, light source mapping and optimized assets to reduce load time are critical points in production. (Webpack 2024, Production)

Tree shaking is used in JavaScript context to mention unused code. Tree shaking analyses import and export statements to find out if modules are imported and exported to being used between JavaScript. Removing unused code can reduce file size for production. (MDN docs 2024, Tree shaking)

2.3.1 Build time

Build time or bundle time is the duration that a bundler takes to compile and bundle all the code and assets of a project to generate bundled outputs. Build time can vary depending on the size and complexity of the project, the configuration of the bundler, and the efficiency of the bundler.

2.3.2 Bundle size

Bundle size is the total size of all files produced by a bundler when it bundles all JavaScript, HTML, CSS, and media. The bundle size impacts the performance of a web application because it takes longer to download, parse and execute in browser, which leads to higher page load time and negative user experience. (Nwamba 2024)

2.4 Hot module replacement

Hot Module Replacement (HMR) allows modules to be updated while an application is running without needing to reload the entire page. Development can be speed up by taking advantage of HMR, such as preserving application state which is lost after a full reload, saving development time by only updating what is changed, immediately updating styles when modifications are made in CSS/JavaScript source code. (Webpack 2024, Hot Module Replacement Concepts)

2.5 Build tool

Build tools are defined in many ways. Roy (Roy 2023, What is a Build Tool? (with Top 5 Build Tools)), describes that build tools are software utilities that simplify the process of converting source code into usable formats like executables and libraries. This is an example of describing a build tool based on its purpose. According to MDN web docs (MDN web docs 2024, Client-side tooling overview), client-side tools can be classified into four categories of problems to be solved: environment, safety net, transformation and post-development. Webpack and Vite are presented as transformation tools. This is an example of describing a build tool based on what problem it solves.

In the context of this research, a build tool is considered as a tool which helps transform source code to generate browser-compatible code for production use.

2.6 Webpack

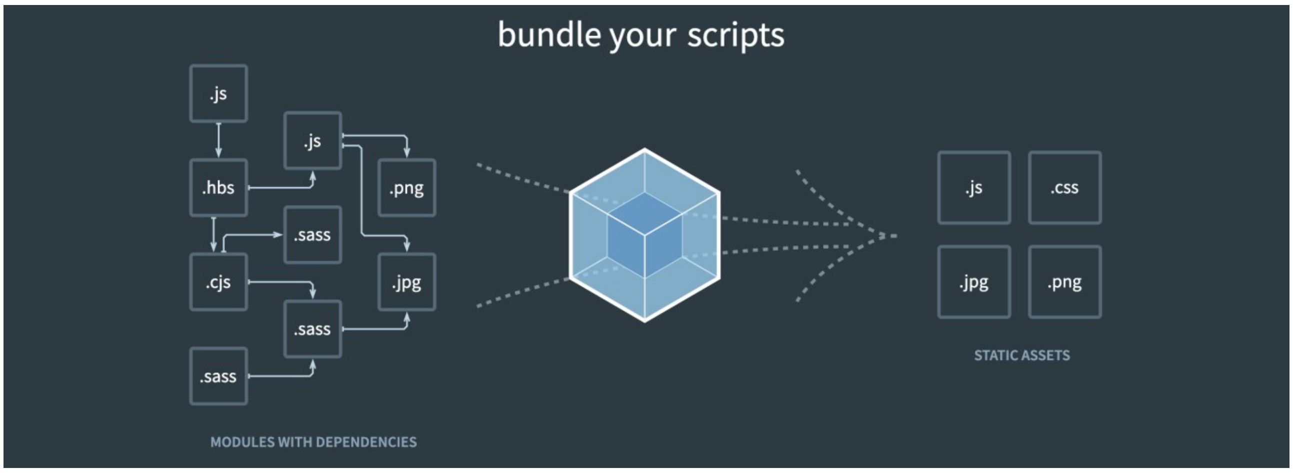


Figure 3. Overview of what Webpack does (Webpack 2024)

Webpack is a powerful and widely used JavaScript bundler and build tool for web applications. (State of JavaScript 2023). Figure 3 illustrates that Webpack processes the application and create dependency graph, then combines modules into one or multiple bundles, which are static assets to be served.

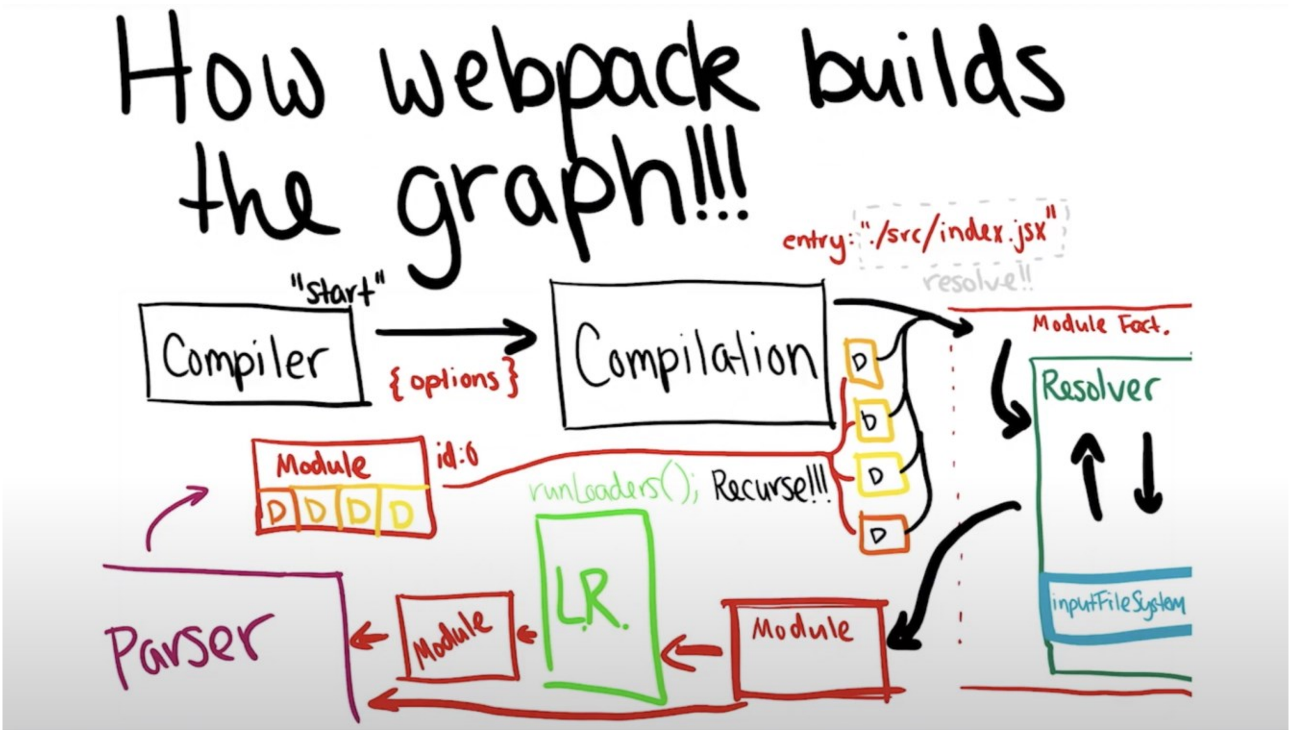


Figure 4. How webpack builds dependency graph (Larkin 2019, 14:14)

Webpack treats any files which depend on another as a dependency. (Webpack 2024, Dependency Graph)

When Webpack processes the application, it uses the configuration file to look for the entry point. The compiler starts from this file. The relative path of this file is transformed to be the absolute path by resolver. The module factory produces a module object with an identification and absolute path so that it can be accessed later. The parser parses the module object to find any dependencies in it. If a dependency is found, it will be attached to the dependency graph. The process of collecting the path, creating module object, parsing, adding to the graph and checking dependency is repeated until there are no more dependencies. After a dependency graph is made, it will be sorted. The sorted dependency graph is used to render dependencies into dependency templates. Dependency templates are code bundles and can be loaded by browsers. (Figure 4.)

Webpack offers loaders to pre-process files and transform files before they are bundled. Loaders can transform files such as from TypeScript (a programming language) to JavaScript. (Webpack 2024, Loaders)

Transpiling is supported in Webpack by using third-party packages, such as babel-loader, esbuild-loader, ts-loader. (Webpack 2024, Transpiling)

From the guide of Webpack (Webpack 2024, Production), minification and source mapping are supported in Webpack. Terser, a minifier for JavaScript code, minify your code in production mode. For CSS minifying, a plugin named css-minimizer-webpack-plugin can be used.

Webpack supports using environment variables. The environment variables in Webpack are different from the environment variables of operating systems. One use case of environment variables is to identify development or production mode. (Webpack 2024, Environment Variables)

In Webpack, sideEffects property in package.json file can be set to tell compiler to safely ignore the file in production mode to perform tree shaking. If a bundler uses ESM completely, it is easy to identify side effects but Webpack is not using fully ESM. (Webpack 2024, Tree shaking)

As presented in chapter 2.5 of this thesis, Hot Module Replacement (HMR) is a feature offered by Webpack. A running application can update modules without having to fully reload due to HMR. According to Webpack (Webpack 2024, Hot Module Replacement Guides), since webpack-dev-server version 4, Hot Module Replacement is enabled by default. When HMR is enabled, the application begins the process by requesting the HMR runtime to check for updates. The runtime downloads updated chunks asynchronously and applies the updates synchronously. The compiler generates updates consisting of a JSON manifest (detailing updated chunks) and the updated chunks themselves, ensuring consistent module and chunk IDs across builds. The runtime manages this process by fetching updates, marking invalid modules, disposing of outdated ones, and replacing them with the new versions, while tracking parent-children module relationships. (Webpack 2024, Hot Module Replacement Concepts)

2.7 Vite

Vite is a build tool which offers fast and lean development experience for modern web projects. (Vite 2024, Getting Started)

Vite takes advantages of two bundlers: Rollup and esbuild. In development environment, esbuild is used for pre-bundling some dependencies. In production build, Rollup is used as a bundler. (Vite 2024, Why Not Bundle with esbuild?)

Esbuild is a JavaScript bundler and minifier written in Go. It is designed to transform, bundle, and optimize modern web applications. Esbuild supports JavaScript, TypeScript, JSX (JavaScript syntax extension), and source maps. (esbuild 2024)

Rollup is a JavaScript module bundler that combines segments of code into complex outputs such as libraries or applications. Unlike older formats like CommonJS or AMD (Asynchronous module definition), Rollup leverages the modern standardized ES6 module format and support tree shaking to remove unused dependencies. This approach allows developers to integrate and use individual functions from their favorite libraries. (Rollup 2024, Introduction)

In development environment, dependency pre-bundling is used. Vite speeds up development server startup by categorizing modules into dependencies and source code. Dependencies, which are stable and often in various formats (e.g. ECMAScript Modules - ESM, CommonJS), are pre-bundled using esbuild. (Vite 2024, Slow Server Start)

When Vite is run for the first time, the project dependencies are pre-bundled before the site is loaded locally. By performing pre-bundling, page load performance is improved by converting many modules into a single module. (Vite 2024, Dependency Pre-Bundling)

Source code, often requiring transformations (e.g., JSX- JavaScript syntax extension, CSS), is served via native ESM. Source code is transformed and served on demand, with the browser handling module loading and only processing dynamically imported code when needed. This approach minimizes upfront processing, optimizes development performance and let the browser take charge partly the job of a bundler. (Vite 2024, Slow Server Start)

Figure 11 and 12 below demonstrates the differences between Bundle based dev server and Native ESM based dev server.

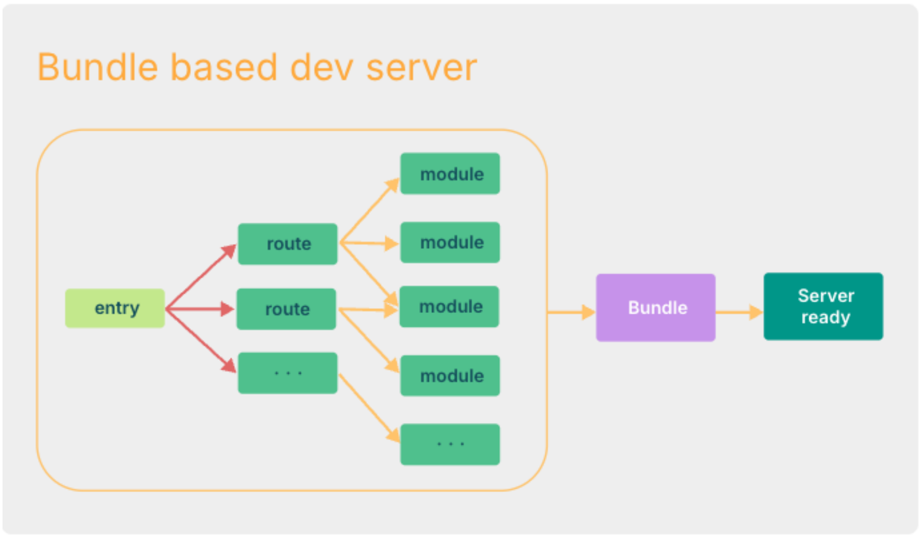


Figure 5. Bundle based dev server process whole application to create bundle to serve (Vite 2024, Slow Server Start)

In a bundle based dev server, the bundling starts from an entry point / entry file, which is indicated in configuration. From the entry file, there are many routes to implement services of the application.

Each route requests one or many modules. The bundler processes all modules to create a bundle of code and serves it. (Figure 5.)

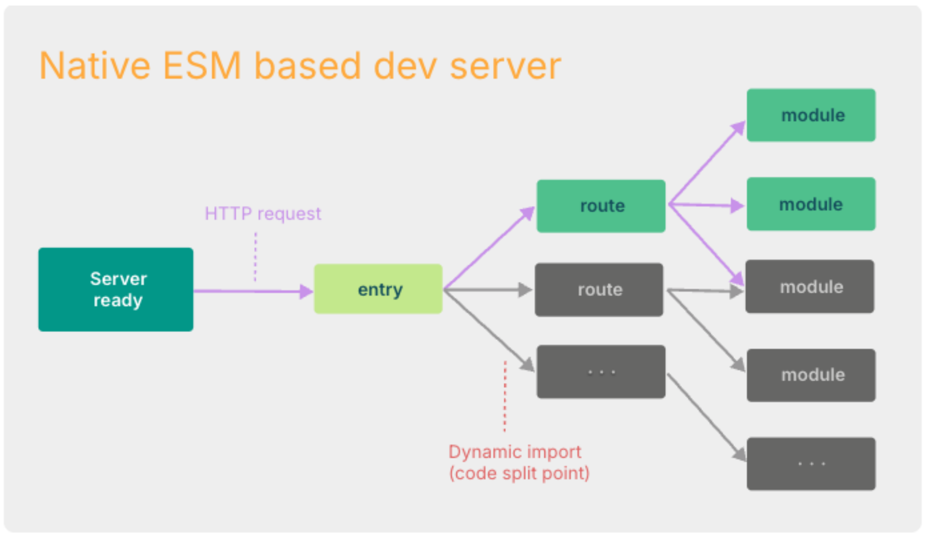


Figure 6. Native ESM based dev server determine which module needs to be reloaded and serve it as requested (Vite 2024, Slow Server Start)

In a native ESM based dev server, the server is started at the entry point. The entry point links to services of the application via routes. Each route is served by one or many modules. Modules are served on-demand, when they are requested by routes, and loaded by browser. (Figure 6.)

Minification is supported in Vite by using build options build.cssTarget and build.minify. By default, build.minify is used. Terser, a minifier for JavaScript code, needs to be installed to be used in build.minify. Source map is supported by using build option build.sourcemap (Vite 2024, Build Options)

Vite only performs transpiling on .ts file (TypeScript file) by using esbuild. (Vite 2024, Features). Vite allows using plugin to add more futures. Babel transpiler is supported by using plugin @vitejs/plugin-react. (Vite 2024, Plugins)

Vite provides environment variables through the special import.meta.env object, which are statically substituted during the build process. (Vite 2024, Env Variables and Modes)

Vite offers an HMR API built on native ESM, enabling frameworks with HMR support to deliver instant and targeted updates without requiring a full page reload or losing application state. When a file is modified, Vite updates the connections from the changed module to the nearest boundary for HMR. This update often affects only the module itself, ensures that only the necessary parts are directly affected by the changes, rather than the entire application. (Vite 2024, Slow Updates)

3 Empirical part and research method

The empirical part and methodology of the research outlines the research questions Q1, Q2, Q3, Q4 in chapter 1.

3.1 Research method

The aim of the research is to explore a comparative analysis of Vite and Webpack. The mixed method between qualitative and quantitative method was chosen to be used in the empirical part. The qualitative method helps answer research question Q2 and Q3, key factors influencing complexity of a configuration file and framework support of Vite and Webpack. The quantitative method helps find out performance differences of Vite and Webpack in terms of build time and bundle size.

3.2 Research design

Case studies were collected for qualitative and quantitative analysis. Case studies were chosen by searching database on Google search, Google scholar, IEEE Xplore to find blogs, papers, articles which discussed Webpack and Vite. Keywords used in searching were: webpack, vite, bundler. Criteria questions to choose case studies were:

Criteria 1: Does the study include Webpack and Vite?

Criteria 2: Does the empirical part of the study describe the setup condition: testing environment, application details, results of build time and bundle size in measurable metrics?

Criteria 3: Does the empirical part include configuration of Vite and Webpack application?

Case studies matched criteria 1 and 2 were used for quantitative analysis. Case studies matched criteria 1 and 3 were used for qualitative analysis.

The quantitative research used bundle size and build time as measurement metric. The size measurement unit is kilobyte (KB), megabyte (MB), and gigabyte (GB). The time measurement unit is the second (s) and millisecond (ms). The lower the size and the time makes higher performance. Different measuring units can be converted for easy comparing by using table 2 and table 3.

Table 2. Size unit conversion

Size unit	Description
Gigabyte (GB)	1 GB = 1024 MB

Size unit	Description
Megabyte (MB)	1 MB = 1024 KB

Table 3. Time unit conversion

Time unit	Description
Second (s)	1 s = 1000 ms

Additionally, a case study was made by the author to compare build time, bundle size, the configuration of a React project using Vite and Webpack. The React project is a front-end application which helps teachers of an academy manage teaching sessions. The testing environment was a MacBook Air 2020 with16 GB RAM, Apple M1 chip, 256 GB storage, run on macOS operating system version 13.3.1. The React based project, used JavaScript and TypeScript, was configured to run on Google Chrome browser using Vite version 5.4.11 then Webpack version 5.97.1. The version of React was 18.2.0. The build time and bundle size were measured 5 times using the same configuration file in each build tool.

3.3 Data analysis method

To analyze the findings from the qualitative research, the case studies were examined by identifying patterns and themes across different configurations of Vite and Webpack when used with frameworks like Vue.js and React. Key factors to evaluate included key factors influencing the complexity configuration, developer experience. Additionally, insight from various case studies was compared to assess similarities and differences in how each tool and framework combination impacts the development workflow.

Findings from the quantitative research were analyzed to evaluate the performance of two build tools in terms of build time and bundle size. Knowledge from the theoretical framework was used to explain the difference in performance.

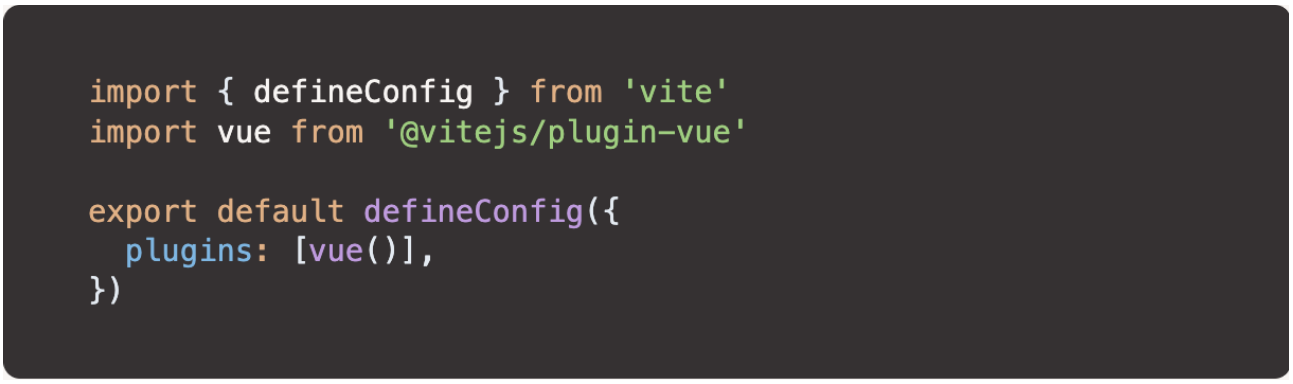
4 Results and findings

This chapter describes found case studies and findings of the empirical part of the research. Since Webpack is the bundler behind create-react-app (CRA), it is worth to examine a case study between CRA and Vite.

4.1 Case study 1: Vite and Webpack comparison

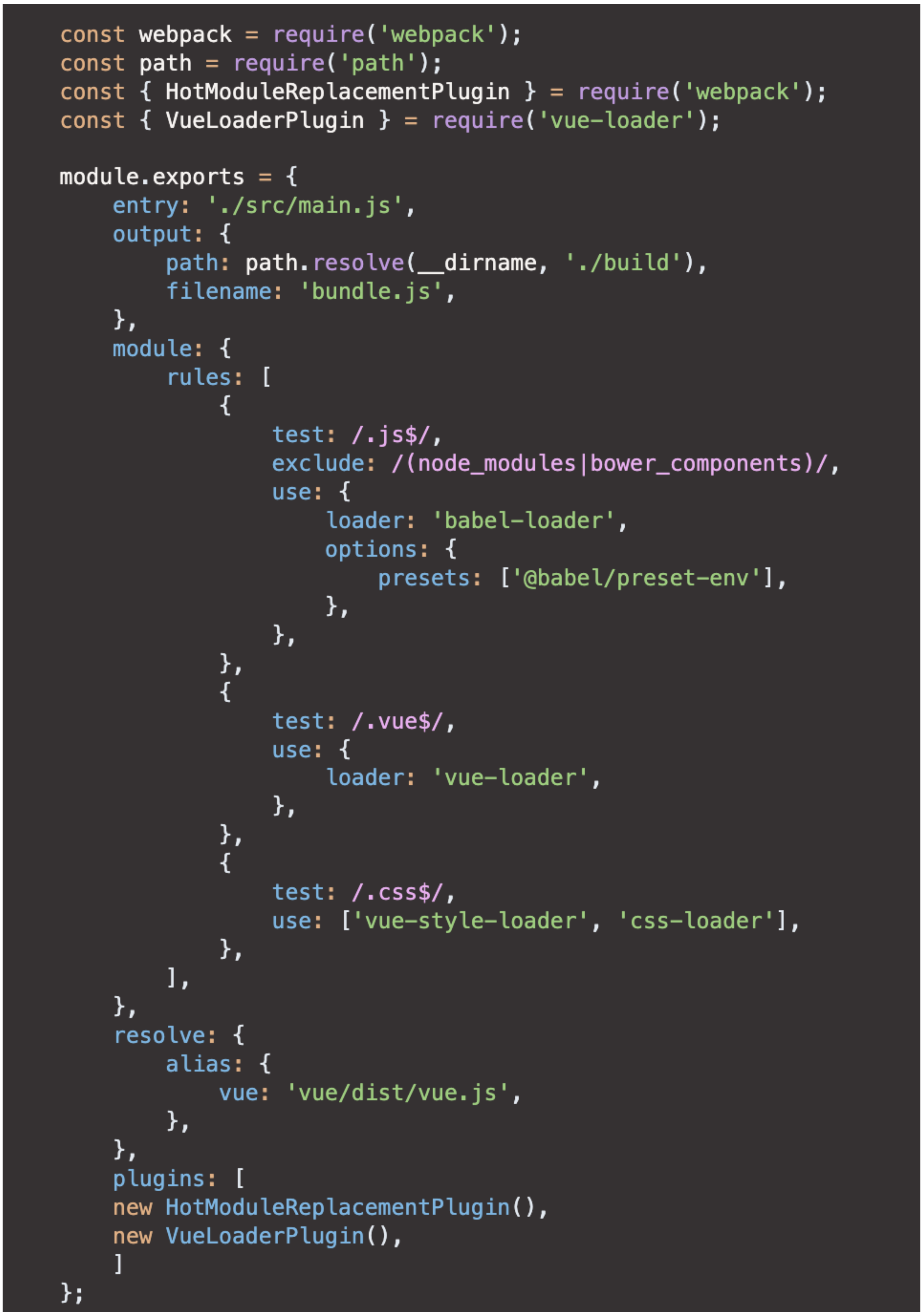
Mostafa Said's article (Said 2024) compared Vite and Webpack based on features, differences, architecture and integration into developer ecosystem. The article aimed to help developers choose right technology based on their needs regardless of the fact that the technology is old or new. Qualitative method was used to analyze configuration files in Vite and Webpack. By measuring build time, bundle size, quantitative method was used.

In chapter 3 of the article, configuration files to setup a basic Vue 3 application in Vite and Webpack were presented.



```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
  plugins: [vue()],
})
```

Figure 7. Basic configuration of Vite for a Vue 3 (Said 2024)

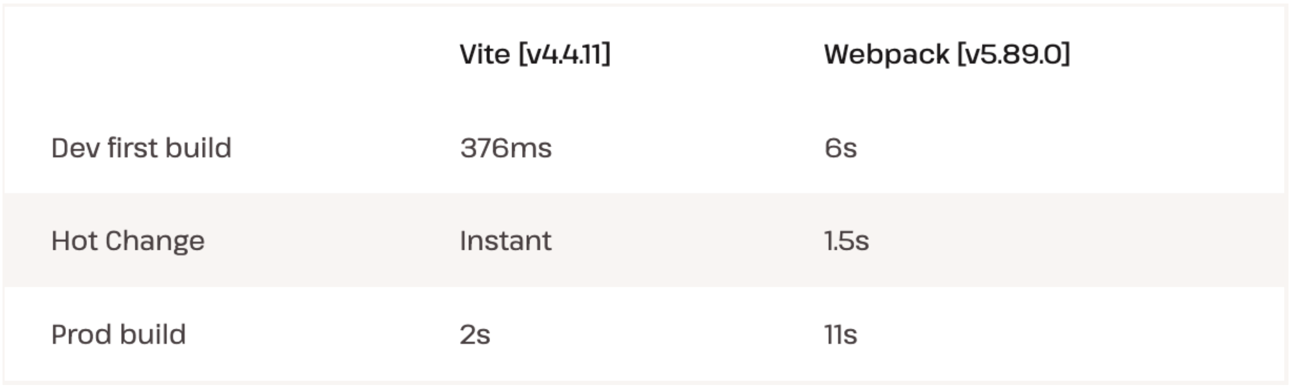


```
const webpack = require('webpack');
const path = require('path');
const { HotModuleReplacementPlugin } = require('webpack');
const { VueLoaderPlugin } = require('vue-loader');

module.exports = {
    entry: './src/main.js',
    output: {
        path: path.resolve(__dirname, './build'),
        filename: 'bundle.js',
    },
    module: {
        rules: [
            {
                test: /.js$/,
                exclude: /(node_modules|bower_components)/,
                use: {
                    loader: 'babel-loader',
                    options: {
                        presets: ['@babel/preset-env'],
                    },
                },
            },
            {
                test: /.vue$/,
                use: {
                    loader: 'vue-loader',
                },
            },
            {
                test: /.css$/,
                use: ['vue-style-loader', 'css-loader'],
            },
        ],
    },
    resolve: {
        alias: {
            vue: 'vue/dist/vue.js',
        },
    },
    plugins: [
    new HotModuleReplacementPlugin(),
    new VueLoaderPlugin(),
    ]
};
```

Figure 8. Possible basic configuration of Webpack for a Vue 3

Figure 7 shows, only Vite's official plugin for Vue.js was imported and installed. Said stated that the configuration is minimal and simple, aligned with Vite's philosophy to simply web development experience. Although Webpack has move to not requiring a configuration file, it is not as automatic as Vite. Figure 8 shows that the configuration of Webpack includes many setups, namely entry and output paths, loaders for different file types, plugins for different features. After analyzing the configuration files based on initial setup options, Said made a conclusion that Vite is good for beginners and quick development because of its simplicity and streamlined development experience. On the other hand, large and complex projects can benefit from Webpack's vast number of options. The sophisticated configuration can be a challenge for new developers and slow down the development.

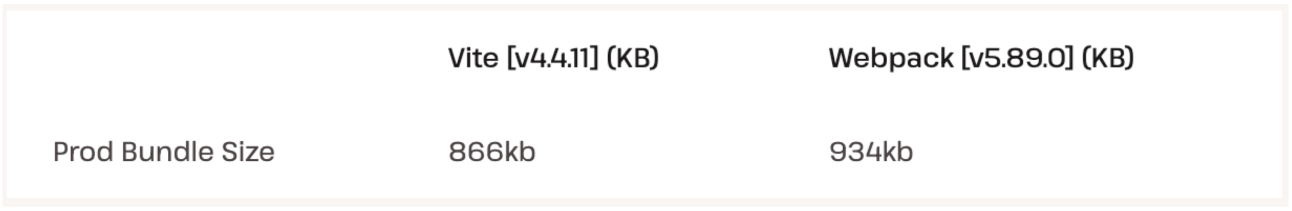
In chapter 5 of the article, a test was made to measure performance of Vite and Webpack. The testing environment, project's scale and specification were mentioned. The test was run on a Macbook Air M1 which had Apple M1 chip and 8-core GPU. The tested application was a medium scale Vue 3 project which contained 10 components, using Vuex and Vue Router. The project also had stylesheets in form of CSS/SASS, images, fonts and some dependencies. The version of Vite is 4.4.11 and that of Webpack is 5.89.0. After quantitatively analyzing, in was concluded Vite was faster than Webpack in building speed. Smaller bundle size by Vite was because of pre-bundling process. Webpack usuall produced larger bundle size but it can be optimized by configurations.



	Vite [v4.4.11]	Webpack [v5.89.0]
Dev first build	376ms	6s
Hot Change	Instant	1.5s
Prod build	2s	11s

Figure 9. Bundling time and hot change comparison in development and production (Said 2024)

In development first build, bundle time was 376 ms using Vite and 6s using Webpack. In production build, bundle time was 2s using Vite and 11s using Webpack. (Figure 9.)



	Vite [v4.4.11] (KB)	Webpack [v5.89.0] (KB)
Prod Bundle Size	866kb	934kb

Figure 10. Bundle size between Vite and Webpack (Said 2024)

Figure 10 shows that production bundle size was 866 KB using Vite and 934 KB using Webpack.

4.2 Case study 2: Create-react-app and Vite comparison

Bhabishya Gurung's research (Gurung 2024) is a comparison of create-react-app (CRA) and Vite for React projects in a bachelor thesis. In the research, a test was conducted to compare performance of CRA and Vite in development and production of React projects. Testing environment was mentioned in chapter 2.5.2 of Gurung's research. The testing environment was Lenovo ThinkBook 14G2 ITL with 2.4GHz i5 processor, 16 GB of RAM, running on Windows 11 and using Google Chrome browser. In chapter 4.2.1 and 4.2.2 of the research, Gurung presented findings from development and production test and analyzed findings quantitatively based on measuring metrics, namely build time, file size, development server startup time and hot reloading time.

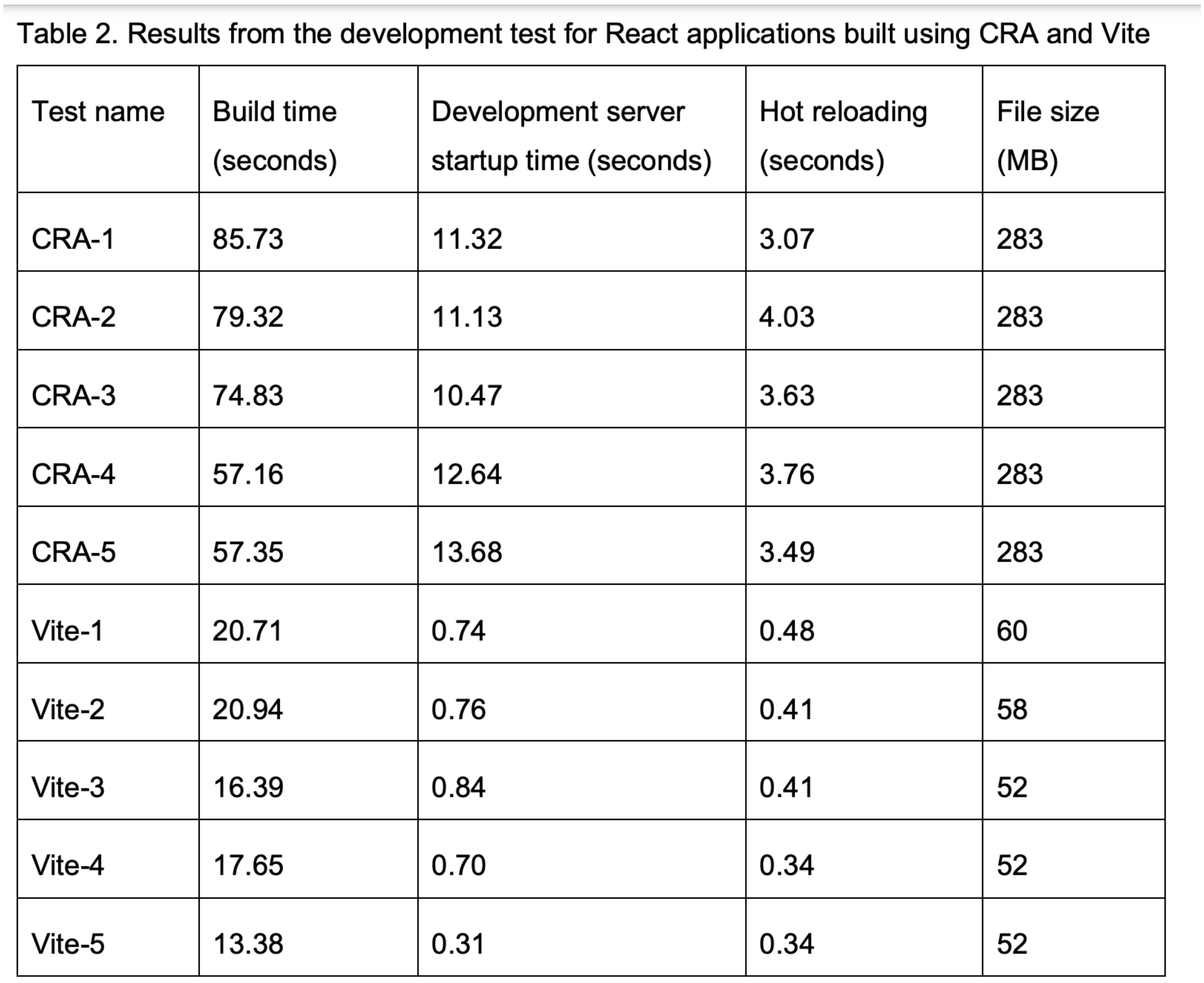


Table 2. Results from the development test for React applications built using CRA and Vite

Test name	Build time (seconds)	Development server startup time (seconds)	Hot reloading (seconds)	File size (MB)
CRA-1	85.73	11.32	3.07	283
CRA-2	79.32	11.13	4.03	283
CRA-3	74.83	10.47	3.63	283
CRA-4	57.16	12.64	3.76	283
CRA-5	57.35	13.68	3.49	283
Vite-1	20.71	0.74	0.48	60
Vite-2	20.94	0.76	0.41	58
Vite-3	16.39	0.84	0.41	52
Vite-4	17.65	0.70	0.34	52
Vite-5	13.38	0.31	0.34	52

Figure 11. Development test results (Gurung 2024)

Build time, development server startup time, hot reloading measured in seconds in CRA were higher than those in Vite. File size measured in MB in CRA was larger than that in Webpack. (Figure 11.)

Gurung concluded that Vite was faster than CRA in development server startup, hot reloading. Hot reloading was the time that the application needed to reload after changing occurred in the code then reflected in the user interface. CRA created file size 5 times larger than Vite did.

Table 3. Development startup time and file size of Trainer app build using CRA and Vite

Trainer app	Development server startup time (sec)	File size (MB)
CRA	14.03	1323
Vite	6.39	197

Figure 12. Production test results (Gurung 2024)

Figure 12 shows the result from measuring development server startup time and file size of the project using CRA and Vite.

From production test results, Gurung stated that CRA took longer time than Vite to run in localhost. The file size of application built by CRA was almost six times larger than Vite. It was argued that the reason why CRA produced larger file size than Vite was because numerous polyfills and packages was included by CRA, while Vite excluded unnecessary dependencies.

4.3 Case study 3: Build a React application using Vite and Webpack

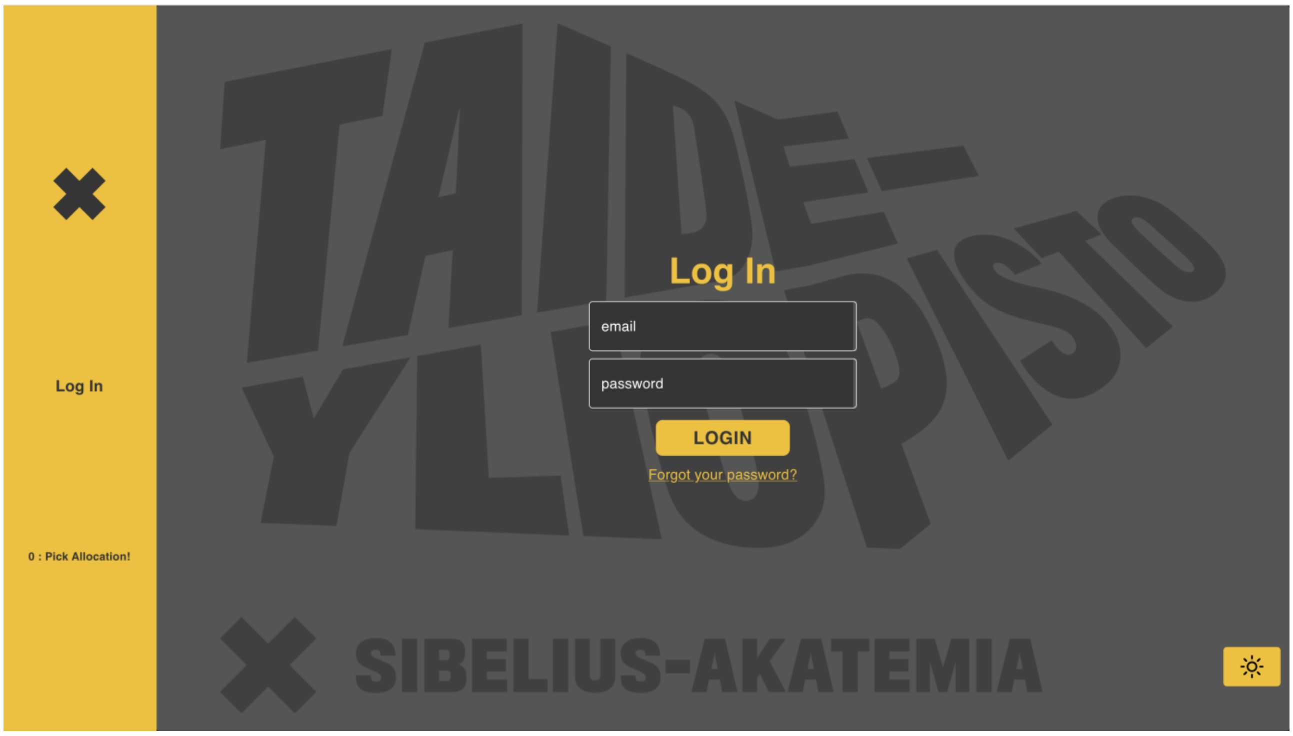
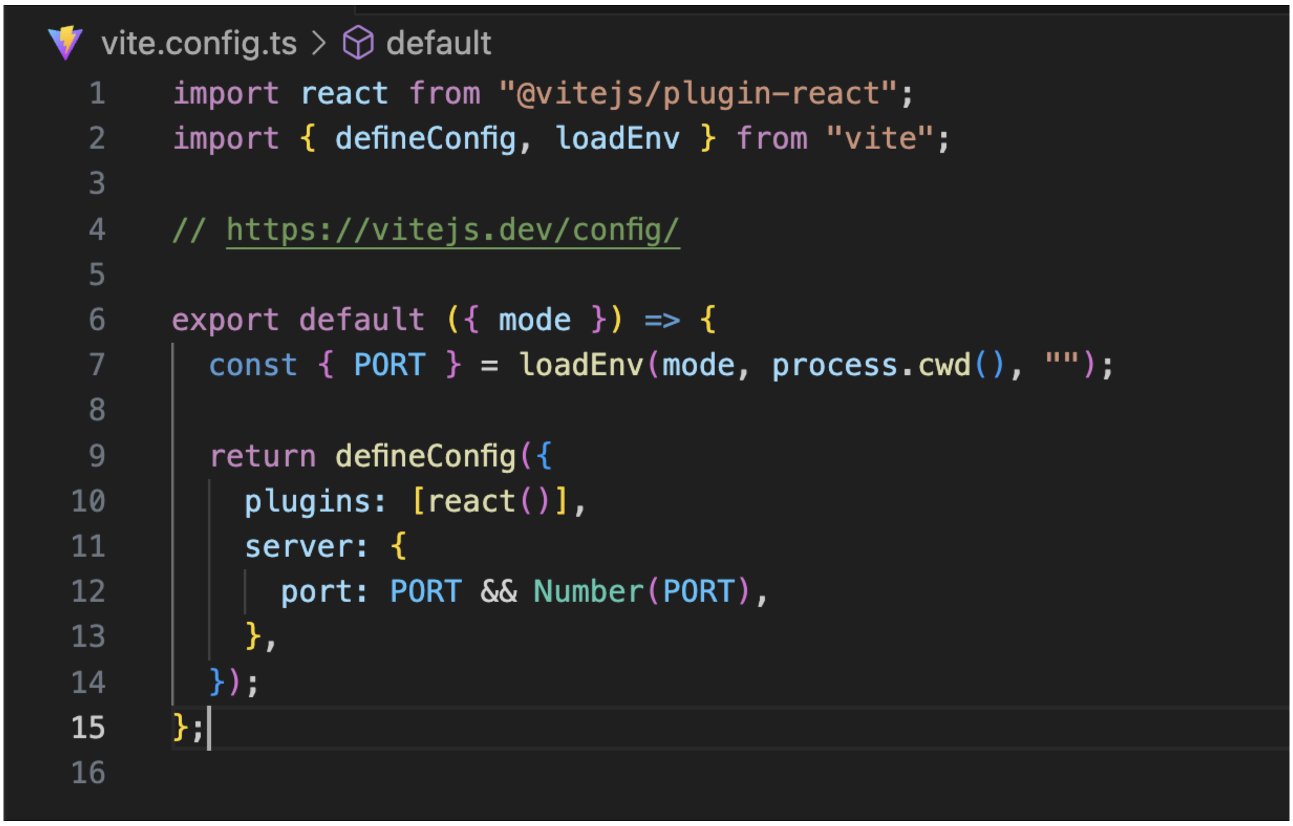


Figure 13. The React application running on web browser (Github, Siba project - Frontend)

Figure 13 shows the login page of the React application running on Google Chrome browser.

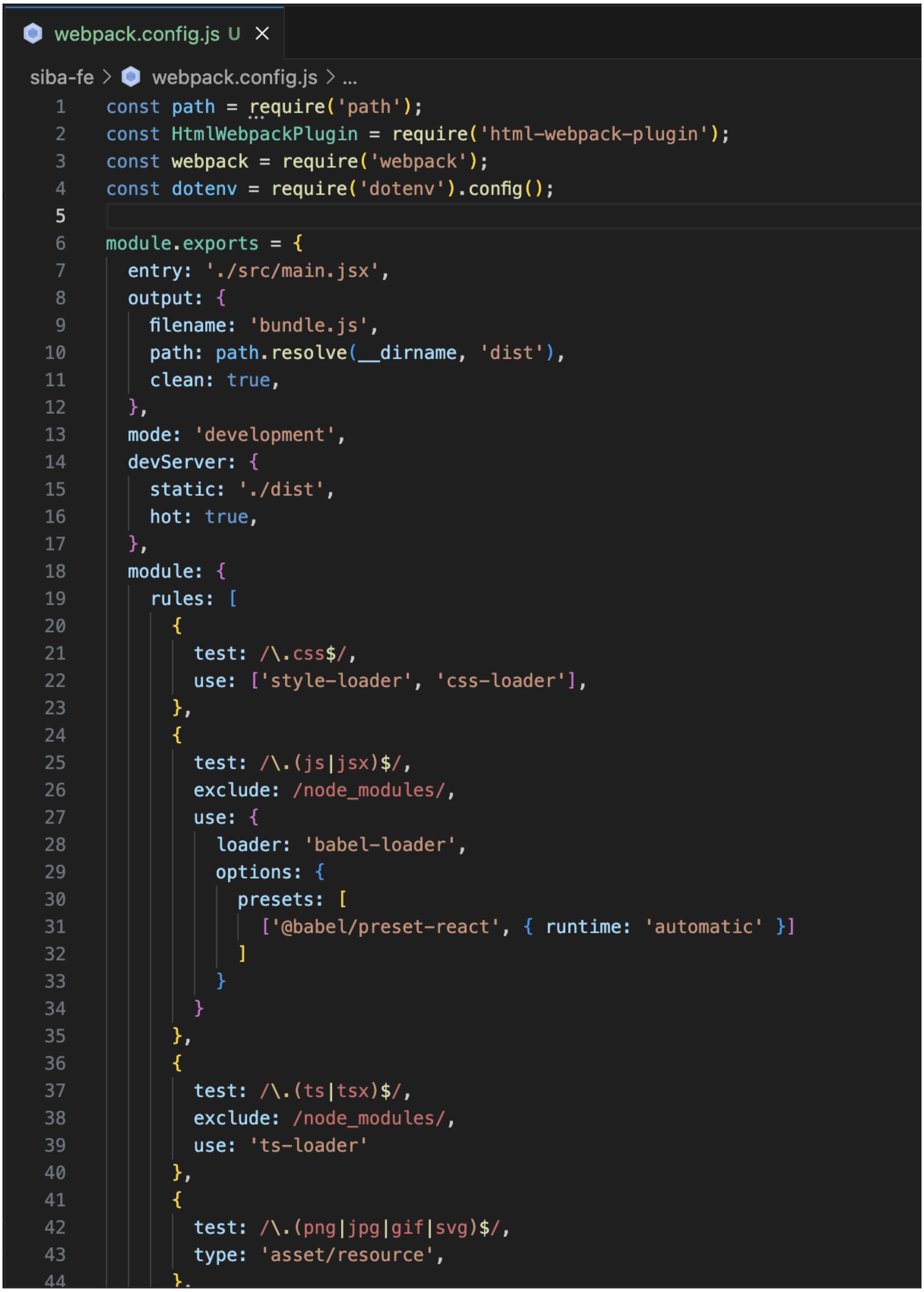


vite.config.ts > default

```
import react from "@vitejs/plugin-react";
import { defineConfig, loadEnv } from "vite";

// https://vitejs.dev/config/

export default ({ mode }) => {
  const { PORT } = loadEnv(mode, process.cwd(), "");

  return defineConfig({
    plugins: [react()],
    server: {
      port: PORT && Number(PORT),
    },
  });
};
```

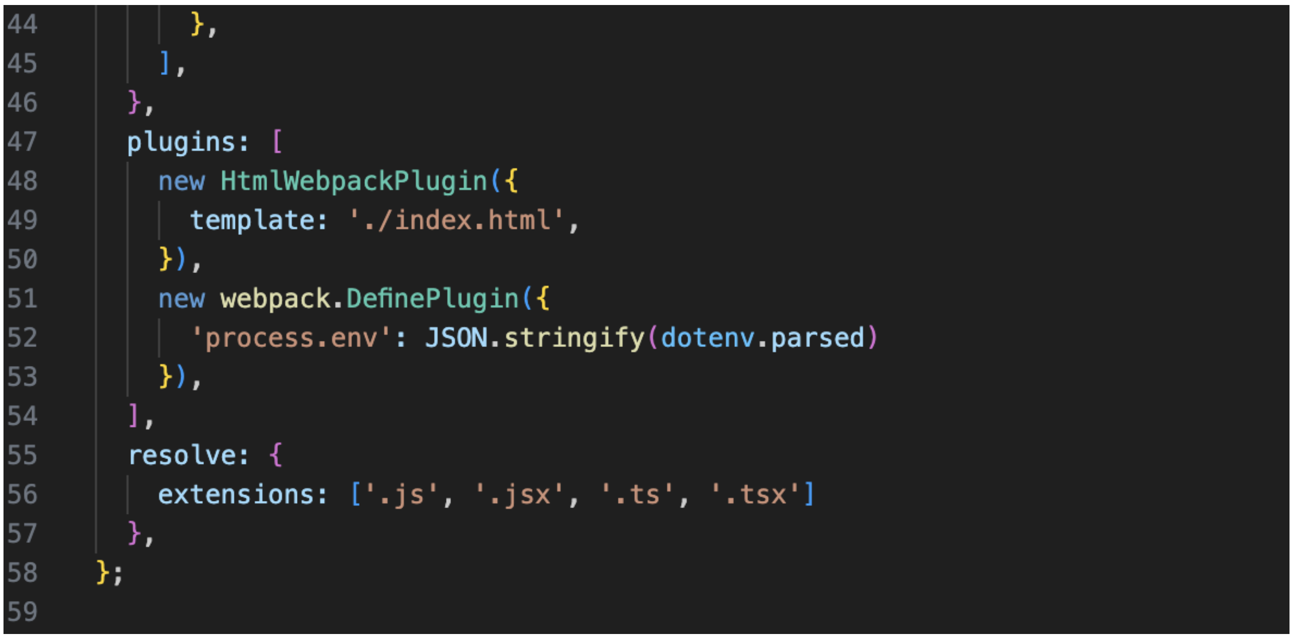
Figure 14. Vite configuration for the React application

From figure 14, some factors influencing the configuration file were mode, port, plugins. First, React plugin was imported, then helper functions defineConfig for intellisense and loadEnv to load environment variables. Mode and port number were loaded from environment variable.



```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const webpack = require('webpack');
const dotenv = require('dotenv').config();

module.exports = {
  entry: './src/main.jsx',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
    clean: true,
  },
  mode: 'development',
  devServer: {
    static: './dist',
    hot: true,
  },
  module: {
    rules: [
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader'],
      },
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: [
              ['@babel/preset-react', { runtime: 'automatic' }]
            ]
          }
        }
      },
      {
        test: /\.(ts|tsx)$/,
        exclude: /node_modules/,
        use: 'ts-loader'
      },
      {
        test: /\.(png|jpg|gif|svg)$/,
        type: 'asset/resource',
      },
```

Figure 15. Webpack configuration file for the React application part 1



```
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html',
    }),
    new webpack.DefinePlugin({
      'process.env': JSON.stringify(dotenv.parsed)
    }),
  ],
  resolve: {
    extensions: ['.js', '.jsx', '.ts', '.tsx']
  },
};
```

Figure 16. Webpack configuration file for the React application part 2

From figure 15 and 16, the configuration file for React application running in Webpack was more sophisticated. First, some core modules and plugins were imported: path is a module to handle file path, webpack is the core module of Webpack, dotenv is a module that loads environment variables from .env file into process.env, HtmlWebpackPlugin is a Webpack plugin which simplifies creation of HTML files to serve webpack bundles.

Then entry point and output directory path with naming option were specified. Mode development was enabled. In devServer, static files location was specified and hot module replacement was enabled.

In Module rules, loaders were indicated for file types. Babel-loader and ts-loader were used here for transpiling.

In plugins, HtmlWebpackPlugin injected bundle into index.html file and webpack.DefinePlugin parse .env file and make environment variables available to the application.

Finally, resolve indicated file extensions so that modules can be imported without file extension in import statements.

Table 4. Results from measuring 5 times building React application by Vite and Webpack

Build tool	Build time development (ms)	Build time production (s)	Build size production (MB)
Vite test 1	120 ms	4.27s	2.9
Vite test 2	152 ms	4.39 s	2.9
Vite test 3	96 ms	4.23 s	2.9
Vite test 4	90 ms	4.19 s	2.9
Vite test 5	98 ms	4.20 s	2.9
Webpack test 1	3570 ms	23.968 s	2.3
Webpack test 2	3386 ms	23.754 s	2.3
Webpack test 3	2920 ms	24.161 s	2.3
Webpack test 4	2958 ms	24.062 s	2.3
Webpack test 5	2945 ms	23.435 s	2.3

After 5 times running test application with Vite and Webpack, the average build time in development was 112.2 ms using Vite and 3155.8 ms using Webpack. The average build time in production was 4.256 s using Vite and 23.876 s using Webpack. Build size in production was 2.9 MB using Vite and 2.3 MB using Webpack. (Table 4.)

Vite was faster than Webpack in build time in both development and production mode. In production mode, Vite generated larger build size than Webpack did.

5 Discussion and Conclusion

This chapter presents a summary from theoretical framework to results and findings from empirical part of the research and discuss them to answer the research questions.

5.1 Similarities and differences between Vite and Webpack

Table 5 below summarize similarities and differences between Webpack and Vite from theoretical framework in chapter 2.

Table 5. Summary of features between Webpack and Vite

Features	Webpack	Vite
Module bundling	Bundle code in both development and production	Pre-bundle dependencies using esbuild, let browser take charge of partly the job of a bundler, serve source code on demand via native ESM in development Bundle code using Rollup bundler in production
Minification	Supported through plugins, e.g. Terser, css-minimizer-webpack-plugin	Natively supported by esbuild
Transpiling	Supported by third-party loaders	Natively supported TypeScript files by esbuild For other files transpiling, supported by using plugins
Source map	Supported	Supported
Environment variables	Supported	Supported
Hot module replacement	Supported	Supported

Features	Webpack	Vite
Tree shaking	Supported	Supported by Rollup, using ES6 module format

Webpack performs code bundling in both development and production build. In development build, Vite uses esbuild to pre-bundle dependencies, lets browser take charge partly the job of a bundler, and source code is served on demand via native ESM. In production build, Vite uses Rollup to bundle whole application before serving it in browser. Minification is supported through plugins in Webpack, such as Terser, css-minimizer-webpack-plugin. For Vite, minification is supported natively by esbuild. Transpiling is supported by third-party loaders in Webpack. Vite natively supports transpiling TypeScript files by esbuild and other files through plugins. Source map, environment variables and hot module replacement, tree shaking are supported in both Webpack and Vite. (Table 5.)

5.2 Complexity of configuration file and frameworks support

Table 6. Some key factors influencing the complexity of configuration files in Vite and Webpack

Build tools	Key factors
Vite	Plugins for features Mode of build Port number Environment variable
Webpack	Entry path Output path Loaders options for file types Plugins for features Resolver for file extensions

From the configuration files in empirical parts, some key factors influencing the complexity of configuration files in Vite are plugins, mode of build, port number, environment variable. For Webpack, those factors are entry path, output path, loaders options for different file types, plugins for features and resolver. (Table 6.)

Compared to Webpack, configuration files in Vite shows simplicity by fewer number of factors. The complexity of configuration files in Webpack can vary a lot based on the number of technology stacks and file types in the project. In the case study 1, loaders for JavaScript and CSS were used in the configuration file, while in the case study 3, loaders for JavaScript and TypeScript were used. In additionally, resolver setup was used in the case study 3 because in the source code, modules were imported without file extension in import statements. The complexity of the configuration file in Webpack offers developers control and customization to details of the project, especially in complex systems which use various technologies.

Case studies 1 and 3 shows that both Webpack and Vite support JavaScript frameworks React and Vue. Official plugins and plugins from third-party packages for those frameworks can be configured to run React and Vue based application.

5.3 Performance in measurable metrics

Table 7. Summary of measuring results from case studies

Measuring metric	Build time development	Build time production	Build size production
Description	Vite took less time than Webpack	Vite took less time Webpack	In some cases Vite outputs smaller file size than Webpack

Vite is faster than Webpack in terms of building time in both development and production. In terms of build size in production, in some cases Vite generates smaller file size than Webpack.

As finding out in the theoretical framework, Vite uses pre-bundling dependencies and native ESM based development server concept while Webpack uses bundle based development server. Vite's development server is started up first then requests needed modules. Webpack takes more time due to generating bundles from whole application before starting up development server.

For build time in production, the difference in build time comes from tree shaking performance of Webpack and Rollup. Rollup takes advantage of using ES6 modules which allows tree shaking processes better than Webpack. Therefore more dead code is removed and build time is reduced in production.

With build size in production, the different outcome in cases can comes from some factors such as minifier, configuration files of bundlers.

5.4 Conclusion

Webpack and Vite have some similarities in providing bundler features such as minification, transpiling, source map, environment variable, hot module replacement, tree shaking. In Vite, tree shaking and bundling without further configuration brings better results than Webpack in development and production build time.

The production build size in Webpack and Vite varies through case studies.

Webpack and Vite support JavaScript framework React and Vue through official and third-party plugins.

The configuration in Vite is simpler than in Webpack. Webpack offers detailed configuration for complex projects.

5.5 Reliability, validity and ethics of the thesis research

Qualitative analysis was conducted based on the theoretical framework to ensure that conclusions reflect the properties of Webpack and Vite. The case study 3 employed test-retest reliability for measurable metrics to ensure the results of the test were stable. The combination of literature review, qualitative and quantitative research supports the validity of the thesis.

The ethics of the thesis contribute to the integrity of the thesis. As the research utilized literature review, public case studies, and data collection through measurable metrics, all sources are cited and used with respect to their original context.

5.6 Limitation of the study and suggestion for future research

The theoretical framework covered the most basic technical concepts and features of Webpack and Vite. There is advanced knowledge from the documentation of Webpack and Vite, as well as Rollup and esbuild. By studying more concepts, there might be a deeper understanding of these tools which facilitates more thorough exploration of two build tools.

The number of testing cases is small and cannot be used to come up with absolute conclusion. More case studies with different project sizes can be setup with difference configuration, optimization to record measurable data for further analysis.

Sources

Cloudflare. 2024. Why minify JavaScript code?. URL: https://www.cloudflare.com/learning/performance/why-minify-javascript-code/. Accessed: 30 November 2024.

Cloudflare. 2024. How to minify CSS for better website performance?. URL: https://www.cloudflare.com/learning/performance/how-to-minify-css/. Accessed: 30 November 2024.

Esbuild. 2024. URL: https://esbuild.github.io/. Accessed: 22 October 2024.

Firefox 2024. Use a source map. URL: https://firefox-source-docs.mozilla.org/devtools-user/debugger/how_to/use_a_source_map/index.html. Accessed: 30 November 2024.

Flanagan, D. 2020. The Definitive Guide, 7th Edition. O'Reilly Media, Inc.

Github. 2024. Siba project - Frontend. URL: https://github.com/haagahelia/siba-fe. Accessed: 11 December 2024.

Geeksforgeeks. 16 August 2024. Web Development. URL: https://www.geeksforgeeks.org/web-development/. Accessed: 30 November 2024.

General Assembly. 2021 What is a JavaScript library?. URL: https://generalassemb.ly/blog/what-is-a-javascript-library/. Accessed: 28 October 2024.

Gurung, B. 2024. A comparative analysis of create-react-app (CRA) and Vite for React.js projects. Haaga-Helia University of Applied Sciences.

Herbert, D. 13 November 2023. What is React.js? Uses, Examples, & More. Hubspot. URL: https://blog.hubspot.com/website/react-js. Accessed: 16 October 2024.

Hubspot 2024. The Beginner's Guide to Website Development. URL: https://blog.hubspot.com/website/website-development. Accessed: 30 November 2024.

Larkin, S. 2019. Everything's a plugin: Understanding webpack from the inside out. URL: https://www.youtube.com/watch?v=H3g0BdyVVxA. Accessed: 20 October 2024.

MDN web docs. 2024. Client-side tooling overview. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Overview. Accessed: 18 October 2024.

MDN web docs. 2024. Handling common JavaScript problems: URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript. Accessed: 30 November 2024.

MDN web docs. 2024. HTML: HyperText Markup Language. URL: https://developer.mozilla.org/en-US/docs/Web/HTML. Accessed: 30 November 2024.

MDN web docs. 2024. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Accessed: 15 October 2024.

MDN web docs. 2024. Package management basics.URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Package_management. Accessed: 17 October 2024.

MDN web docs. 2024. Tree shaking: URL: https://developer.mozilla.org/en-US/docs/Glossary/Tree_shaking. Accessed 12 December 2024.

MDN web docs. 2024. Understanding client-side JavaScript frameworks. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks. Accessed: 15 October 2024.

Nwamba, C. 15 January 2024. Strategies for Crafting High-Performance Web Apps with Smaller Bundle Sizes. DEV Community. URL: https://dev.to/codebeast/strategies-for-crafting-high-performance-web-apps-with-smaller-bundle-sizes-2mmj. Accessed: 17 October 2024.

React. 2024. URL: https://react.dev/. Accessed: 28 October 2024.

Rollup. 2024. URL: https://rollupjs.org/. Accessed: 21 October 2024.

Rollup. 2024. Introduction. URL: https://rollupjs.org/introduction/. Accessed: 1 December 2024.

Roy, S. 2023. What is a Build Tool? (with Top 5 Build Tools). BrowserStack. URL: https://www.browserstack.com/guide/build-tools. Accessed: 18 October 2024.

Said, F. 6 August 2024. Vite vs. Webpack: A Head-to-Head Comparison. Kinsta. URL: https://kinsta.com/blog/vite-vs-webpack/. Accessed: 17 November 2024.

State of JavaScript 2023. Build tools. URL: https://2023.stateofjs.com/en-US/libraries/build_tools/. Accessed: 12 October 2024.

Vite. 2024. Build Options. URL: https://vite.dev/config/build-options. Accessed: 1 December 2024.

Vite. 2024. Env Variables and Modes. URL: https://vite.dev/guide/env-and-mode. Accessed: 1 December 2024.

Vite. 2024. Dependency Pre-Bundling. URL: https://vite.dev/guide/dep-pre-bundling.html. Accessed: 11 December 2024.

Vite.2024. Features. URL: https://vite.dev/guide/features. Accesses: 1 December 2024.

Vite. 2024. Getting Started. URL: https://v2.vitejs.dev/guide/#getting-started. Accessed: 20 October 2024.

Vite. 2024: Plugins. URL: https://vite.dev/plugins/. Accessed: 1 December 2024.

Vite. 2024. Slow Server Start. URL: https://v2.vitejs.dev/guide/why.html#slow-server-start. Accessed: 17 October 2024.

Vite. 2024. Slow Updates. URL: https://vite.dev/guide/why#slow-updates. Accessed: 11 December 2024.

Vite. 2024. Why Not Bundle with esbuild?. URL: https://vite.dev/guide/why#why-not-bundle-with-esbuild. Accessed: 15 November 2024.

Vue.js 2024. URL: https://vuejs.org/. Accessed: 16 October 2024.

Vue.js 2024. Introduction. URL: https://vuejs.org/guide/introduction. Accessed: 16 October 2024.

Webpack. 2024: Transpiling. URL: https://webpack.js.org/loaders/#transpiling. Accessed: 11 December 2024.

Webpack .2024. Environment Variables. URL: https://webpack.js.org/guides/environment-variables/. Accessed: 30 November 2024.

Webpack. 2024. Dependency Graph. URL: https://webpack.js.org/concepts/dependency-graph/. Accessed: 20 October 2024.

Webpack. 2024. Hot Module Replacement Concepts. URL: https://webpack.js.org/concepts/hot-module-replacement/. Accessed: 17 October 2024.

Webpack. 2024. Hot Module Replacement Guides. URL: https://webpack.js.org/guides/hot-module-replacement/. Accessed: 2 November 2024.

Webpack. 2024. Loaders. URL: https://webpack.js.org/concepts/loaders/. Accessed: 30 November 2024.

Webpack. 2024. URL: https://webpack.js.org/. Accessed: 18 October 2024.

Webpack. 2024. Production. URL: https://webpack.js.org/guides/production/. Accessed: 30 November 2024.

Webpack 2024. Tree shaking. URL: https://webpack.js.org/guides/tree-shaking/. Accessed: 12 December 2024.

Yurchyshyn, Y. 24 April 2023. Front-End Build Tools and Workflow Optimization. Romexsoft. URL: https://www.romexsoft.com/blog/front-end-build-tools-and-workflow-optimization/. Accessed: 30 November 2024.