OULU UNIVERSITY OF
APPLIED SCIENCES

Baidauletov Dair, Salehin Fayjus

INTEGRATING MODERN FRONT-END METHODOLOGIES AND WORKFLOW IN THE CONTEXT OF E-COMMERCE SYSTEMS

INTEGRATING MODERN FRONT-END METHODOLOGIES AND WORKFLOW IN THE CONTEXT OF E-COMMERCE SYSTEMS

Baidauletov Dair, Salehin Fayjus
Bachelor’s Thesis
Spring 2020
Information Technology
Oulu University of Applied Sciences

ABSTRACT (DAIR & FAYJUS)

Oulu University of Applied Sciences
Information Technology, Internet Services

Authors: Baidauletov Dair, Salehin Fayjus
Title of Bachelor´s thesis: Integrating Modern Front-End Methodologies and Workflow in the Context of E-Commerce Systems
Supervisor: Teemu Korpela
Term and year of completion: Spring 2020 Number of pages: 97 + 2 appendix

This thesis investigates modern front-end development methodologies and workflow in the context of e-commerce platforms. It aims to improve the development cycles in the agile team's workflow and overall quality and efficiency of delivering e-commerce solutions. The research was commissioned by Vaimo Finland Oy, the omnichannel e-commerce solution provider. The company has been looking for new technologies and tools that can help to build Progressive Web Applications with support for the Magento platform.

The latest web development technologies, namely PWA, were integrated into the e-commerce infrastructure. Vue Storefront was used as a PWA storefront solution with Magento in the back-end. The benefits for clients and development teams were researched and assessed. The development of the components, installation, and configuration of third-party modules, building and deploying procedures were tested, documented and applied during the implementation of the project. Potential improvements in the current workflow with the help of modern web technologies were studied. Automated unit and functional testing, accessibility and performance evaluation, a realization of the design system for creating sustainable component library were tested, and their effects on the collaboration of developers, QA and UI/UX engineers were studied and compared with traditional workflow.

The research resulted in a number of positive discoveries in favor of PWA storefronts. Benefits of headless architecture improved the key performance metrics of the web application and separation of concerns between front-end and back-end developers. The continuous integration time was shortened to less than 10 minutes with the help of headless architecture and existing DevOps practices. Automated web testing has proved to be helpful in improving quality engineering in the development cycles, and developing design systems was found to be useful for improving the collaboration between designers and front-end developers.

Keywords:

Front-end Development, E-commerce, Progressive Web Application, Headless Architecture, Design System, Magento, Vue Storefront, PWA, Automated Web Testing

PREFACE

This thesis was carried out while we were working for Vaimo Finland. We would like to thank them for giving us the opportunity and guiding us throughout the journey, especially our Front-End Architect, Jani Maljanen, for all the technical guidance and our Team Lead, Eino Keskitalo, for helping us organize the thoughts and keeping us on track.

We would also like to thank our thesis supervisor Teemu Korpela for his constant support and guidance. We learned a lot about writing a thesis from our language teacher Kaija Posio. We have acquired most of the technical knowledge reflected in this document over the time we studied at the Oulu University of Applied Sciences, and the teachers made this journey of learning and growing a valuable and memorable experience for us.

Oulu, 12.03.2020.
Dair Baidauletov & Fayjus Salehin.

CONTENTS

TERMS AND ABBREVIATIONS

API	Application Program Interface
CBA	Component-Based Architecture
CDN	Content Delivery Network
CSR	Client-side rendering
CI/CD	Continuous Integration/Continuous Delivery
DOM	Document Object Model
E2E	End-to-end (testing)
ES	ElasticSearch
FCP	First Contentful Paint
K8S	Kubernetes
MVC	Node Package Manager
NPM	Model-View-Controller
PWA	Progressive Web Application
QA	Quality Assurance
RAIL	Response, Animation, Idle, Load
SEO	Search Engine Optimization
SOA	Service-Oriented Architecture
SoC	Separation of Concerns
SPA	Single-Page Application
SSR	Server-side rendering
TTFB	Time to first byte
UI/UX	User Interface/User Experience
VSF	Vue Storefront
VSF-API	Vue Storefront API

1 INTRODUCTION (Dair)

This thesis was commissioned by Vaimo Finland Oy, a part of the international full-service omnichannel agency Vaimo. Their Finnish business unit, located in Oulu, Finland, specializes in building and maintaining e-commerce solutions based on the Magento platform and providing other e-commerce services for various enterprises.

Although tailored mainly towards enterprise-level e-commerce businesses, Magento holds the leading position as the world's most demanded e-commerce solution.[1] Despite its extensibility, Magento is not equipped to deliver web technologies that are starting to gain more attention from clients all over the globe. Mainly, the Progressive Web Application (later referred to as PWA), which unites multiple new web features and design patterns, including offline access to a website, push notifications, and faster loading. Businesses see the benefits in improving the speed of their services and increasing engagement and conversion rates with these technologies.[2] The concept of PWA was first proposed in 2007 by Steve Jobs,[3] while the term was established by the Google Chrome developer Alex Russel and designer Frances Berriman in 2015.[4] Since then, building Progressive Web Applications have turned into a dynamic and growing domain of web development. Service workers, one of the principal features of PWA, can now be found on approximately 15% of all page loads.[5] With leaders of the industry, namely Google, Microsoft, and Apple, actively supporting the direction of modern web approaches, it is evident that e-commerce will change in favor of PWA.[4]

This change in demand could result in a positive effect on the development of the applications, allowing developers to create platform-agnostic solutions with a single codebase, along with bringing native experiences to the web. The majority of PWA solutions available today are based upon headless architecture, where the server-side and business logic are detached from the client-side logic. This significant difference can ease the separation of concerns (SoC) among front-end and back-end developers in the software development workflow. Most of the PWA storefront solutions known today are powered by either React (Magento PWA Studio), Vue (Vue Storefront), or Angular (Spartacus) and served with Node.js.[6] It enables developers to take full advantage of the software technology stack, developer community, and maximize the efficiency of front-end development with the help of GraphQL or open-source NPM packages, and without the development being restricted by Magento's front-end architecture. Magento and other PHP frameworks are not capable of providing Single-Page Applications (SPA) by default. SPAs can increase the productivity of front-end development. Current Magento front-end is firmly affected by "building giant XML trees on the run" that requires both memory and the CPU.[7] Hence, "those trees are built up from files on disk and on each request," slowing the local development, especially if the changes are only front-end specific. While the application's business logic remains attached to the platform of choice, it is now possible to expose specific API endpoints without the platform or other vendors constraining and influencing the implementation of front-end.

Today several maintained frameworks facilitate building PWA e-commerce storefronts. They include Deity Falcon, Front-commerce, Magento PWA Studio, Scandi PWA, and Vue Storefront, with latter supporting most of the existing e-commerce platforms and holding the strongest developer community.[8] Since creating a PWA solution from scratch can be expensive[9], it is in businesses' and service providers' favor to implement one of the existing and maintained technologies, most of which are integrable into multiple prevalent e-commerce platforms, such as Magento, Prestashop, Shopware. Most of the known PWA storefront solutions have been developed and found commercial use during the last quarter of the past decade. They can be divided into two different types – the first one being the standalone PWA storefront (figure 1) for e-commerce with an API connection to back-end of a chosen system (Vue Storefront, Deity Falcon and FrontCommerce) and the second one being the suite of tools to help building PWA experiences (Magento PWA Studio, Scandi PWA).[10]

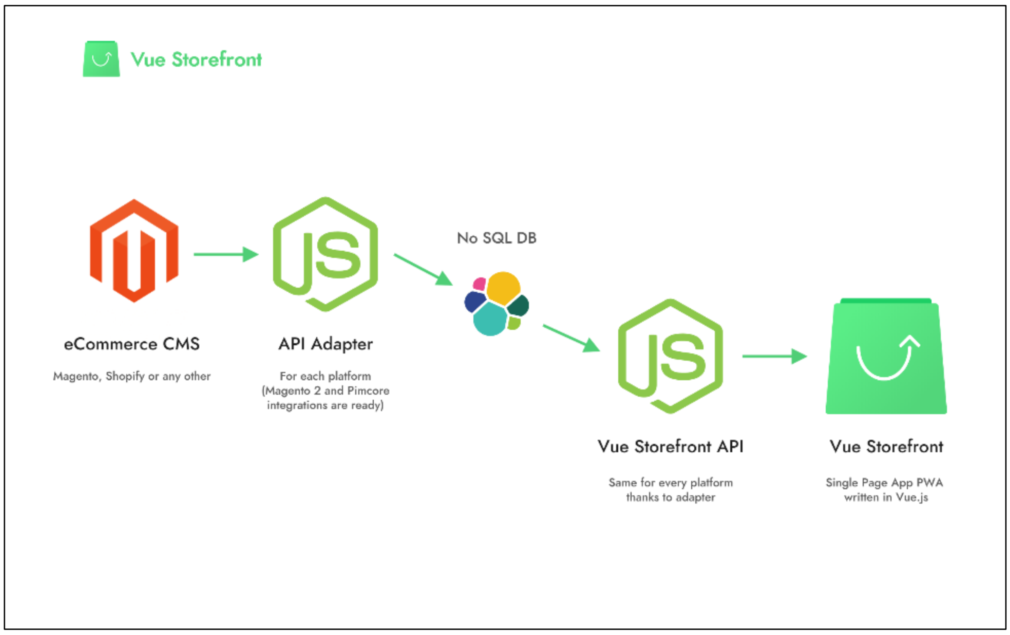

Vue Storefront
No SQL DB
eCommerce CMS
Magento, Shopify or any other
API Adapter
For each platform (Magento 2 and Pimcore integrations are ready)
Vue Storefront API
Same for every platform thanks to adapter
Vue Storefront
Single Page App PWA written in Vue.js

FIGURE 1. Headless architecture in PWA solutions – Vue Storefront example.[11]

In this thesis, modern web development practices will be integrated into a complex e-commerce platform with the help of PWA solutions gaining recognition in the field. Magento will serve as an e-commerce platform, and Vue Storefront will be used as a PWA solution.

In the process of developing an e-commerce website, the efficiency and quality of front-end deliverables will be assessed, along with the effect on the workflow within the development team. The client's satisfaction and webshop's conformity to the latest technologies are the main aims of the project, whereas introducing modern web solutions into an e-commerce company's set of provided services is the primary aim of the research. The supporting aim and point of investigation includes workflow optimization in the e-commerce system's development cycles through the adapted technologies and additional tooling for automated testing and design improvements. The effects of the Vue Storefront on the collaboration and independence of front-end and back-end developers, QA and UI/UX engineers will be studied, analyzed, and compared to traditional approaches.

2 OVERVIEW OF EXISTING METHODOLOGIES (Dair)

Magento is a monolithic application developed towards the total support of any business needs. It brings together a centralized catalog, multi-lingual support, currency and tax rates, the high-end Search Engine Optimization (SEO) engine, and many other features, making it one of the ultimate choices for clients.[12] In addition to that, Magento is highly customizable. It empowers developers to “reassign every single bit” of its core logic. However, the advantages of the monolith do not always outweigh some of the subsequent drawbacks and limitations affecting both the resulting product and development process. “As processes in a monolith are deeply interdependent, a bug in any module can potentially bring down the whole system”.[13] When an application becomes complex, the impact of new changes cannot always be guaranteed. As a result, new changes require deploying the entire application and extensive manual testing.[13]

There are two alternatives to the monolithic architecture – headless and service-oriented architecture (SOA).[13] In headless architecture, the front-end of the web application is decoupled from the back-end and communicates via the aggregation layer – API. In this approach, which has been adopted in most of the e-commerce PWA solutions, front-end is a standalone application that runs on a separate instance and can be deployed and accessed even if the back-end is out of reach. Service-Oriented Architecture is based on the decoupling of back-end logic into microservices (figure 2). Since SOA addresses improvements for the business logic of the application, its absence does not prevent the implementation of PWA with headless architecture alone. Therefore, describing PWA solutions in the e-commerce systems built with microservices is beyond the scope of this research.

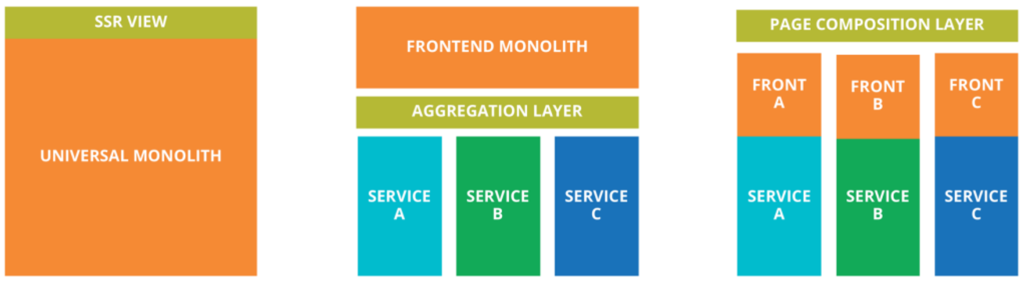


FIGURE 2. Software architecture types based on modularity.[14]

2.1 Disadvantages of current technologies (Dair)

Most often, development on the Magento website is done by e-commerce agencies with technical competence that is crucial in ensuring the proper execution of the project implementation and future maintenance. Projects with no or little back-end customization can nevertheless be expensive to launch, and front-end implementation may comprise the most substantial portion of the budget. With growing codebase, maintenance of the completed projects becomes slow and cumbersome, increasing costs for clients.[15]

According to Jisse Reitsma, the founder and developer at Yireo, the current Magento 2 front-end is repulsive for front-end developers.[16] He implies that Magento projects "might have become more expensive simply because the implementation of front-end is unnecessarily complex." One of the advantages of modern front-end built with React or Vue is the short learning curve and their popularity. Developers with no or little experience with JavaScript frameworks can learn them in a shorter amount of time. Nowadays, front-end developers are more likely to have experience with React or Vue than with Magento 2, Knockout, or Require. This allows them to use already existing knowledge when starting to work with PWA solutions. In contrast, learning Magento 2 front-end requires finding and learning the Magento-specific knowledge.

MAGENTO MVC (DAIR)

Magento uses a configuration-based Model-View-Controller architecture (MVC).[17] Although MVC enables simultaneous development by decoupling applications into smaller elements, it influences the architecture of every module used in a project. Decomposition of every feature into three artifacts and the resulting scattered code may affect flexibility in customizing the front-end. It is often the case where front-end developers are expected to modify only the View layer. While MVC helps to build loosely coupled modules and integrations, the elements within those modules can be interdependent, i.e. the controllers invoke templates, and data is fetched with methods from the controller.[18] Developers are also expected to know about working with all of the MVC elements when building a feature, which might include modifying a database or business logic.[19] Headless architecture decouples the front-end application from the back-end. It provides opportunities for a better separation of concerns as front-end developers would rely exclusively on the data coming from the API. A front-end application could have data handling and templating patterns that are not restrained by an external logic, which means that the system could benefit from looser coupling, and there is more control to the storefront of the e-commerce website.[18]

2.2 Disadvantages of current technologies for clients (Dair)

Some of the challenges that may hinder reaching e-commerce system's full potential from the client's perspective include:[20]

- **Maintenance**. Maintenance costs increase as the application grows. It becomes more time-consuming to modify the codebase.
- **Speed**. The size of the application and catalog might negatively affect the start-up time. According to Web Almanac, only 23% of websites built on Magento take less than a second to display a meaningful content, such as text or an image (First Contentful Paint). Almost half of them take less than 3 seconds, and the rest 31% are ranked as slow.[21]
- **Flexibility**. Although easy in the early stages of development, monolithic systems have difficulty in adopting the latest and advanced technologies.

It usually takes 3-6 months on average to develop a production-ready Magento store.[22] The implementation costs highly depend on the number of dedicated developers, integrations, extensions, SEO, and front-end customizations. In-house Magento developers are less common than service providing agencies with expertise in digital commerce. Joining e-commerce may not be a small investment for SMEs. While developing the website is essential to any client, maintenance is of equal importance. When the website is deployed to production, clients may want to create technical or functional spikes[23] or request performance profiling, cosmetic fixes, support during campaigns, and guidance with technical issues. However, the expensive support costs and slow modifications with a monolithic system may discourage both clients and service providers from taking action. As a result, clients become dependent on costly support, even when they want to make small front-end changes, such as fixing minor cosmetic issues or implementing a website facelift. Reducing the time required for maintenance changes could have a positive impact on both parties. Clients would feel more encouraged to make new requests, whereas service providers and developers would benefit from the client's determination and easiness of the system.

When implementing the front-end according to web design, it is a challenge to make the result identical to the mockup, which leaves the client unhappy. Since the design is just a general guideline, different scenarios and use cases should be considered during the development of UI to prevent bugs. The lack of prototyping the resulting output on the code level is affecting the client's expectations in reality. An increasing collaboration between developers and designers could be time-consuming and inefficient for the development teams.

2.3 Disadvantages of current technologies for front-end developers and development workflow (Dair)

Front-end development is firmly affected by the following factors:

- **Naturally not optimized for front-end development**. Magento's front-end makes use of relatively old and deprecated JavaScript libraries, such as jQuery (v1.11) or Knockout.js, prioritizing the browser support over simplicity and developer-friendliness. This slant and other nuances, such as slow local development, result in inefficient and outdated front-end practices.
- **Lack of documentation**. Magento has an active developer community. Despite that, there is a lack of well-documented official sources and best practices which can improve the learning curve or help beginners study effectively on their own.[24] On the other hand, Magento has been actively in use through the past decade, and sources, such as GitHub, Stack Overflow, are full of issues, guides, and commits related to different development topics. The same cannot be said about Vue Storefront and other new PWA solutions.
- **Cumbersome internal development**. Developers need to redeploy the entire application on every change. This inconvenience unnecessarily delays transitions in the agile development workflow.
- **Reliability**. Even though the platform is modular and loosely coupled, the runtime is interdependent. Error in any module can potentially bring down the entire process.

In the study of Web Almanac, it was discovered that nearly 10% of home pages visited on the internet are built on an e-commerce platform, Magento adopting 1% of the total number.[21] With monolithic architecture and other traditional software development procedures inherited in Magento, developers may benefit from the modern web technology stack and JavaScript frameworks. Nowadays, jQuery can be found on almost 85% of all desktop and 83% of all mobile home pages.[21] According to data collected by the State of JavaScript Survey,[25] the popularity of front-end frameworks and open-source libraries that make building single-page applications easier, such as React, Vue, AngularJS, and Backbone.js, increased over the last couple of years.[26] These frameworks have a strong and growing community and have a higher developer satisfaction.[27]

Switching from jQuery to a newer client-side library can take time, depending on how large the application is, and many websites are built with jQuery combined with the newer client-side libraries.

Development teams, including front-end and back-end developers, QA and UI/UX engineers, can face different challenges when working with Magento. Magento architecture might dictate the way user stories and technical stories are defined in the agile development model.[28] Magento MVC, while enhancing the code structure, fails to entirely separate back-end from front-end due to tight coupling.[19] Therefore, user story splitting might negatively affect the performance and collaboration of developers. Given there are two tasks for one feature, separated into the back-end and front-end part, it is likely they would end up blocking each other, forcing either an increased collaboration or reassignment to one person. These issues are hard to identify during the planning phase. As a result, it is easier to produce repeating code, whether it is related to styling or back-end functionality.

2.4 Summary of disadvantages (Dair)

Issues described above identify the main drawbacks of the current state of delivering e-commerce solutions with Magento:

1. Clients may be unwilling to invest more due to the fact that small front-end changes are expensive to implement.
2. Clients are not satisfied with the performance and speed of the website.
3. Clients are not satisfied with the look and feel of the user interface.
4. Front-end developers may be affected by outdated practices since modern technologies are not easily integrable into Magento.
5. Front-end developers may be affected by imperfect segmentation of MVC patterns and separation of concerns.
6. QA and UI/UX engineers may find it hard to integrate automated testing and design systems into the infrastructure.

Magento, like any other monolithic application, is simple to deploy and easy to test. However, the CI/CD operations take long since the application has to be reinstalled entirely, despite the size of changes (see Magento architecture diagram in figure 3). One of the main aspects of agile development is how people work together.[29] In the agile web development teams are expected to collaborate with UI/UX engineers and refer to the specification. Besides, front-end developers are expected to do code reviews to ensure best practices and prevent code repetition. QA engineers or automated web testing are not capable of ensuring that the latest changes do not affect other functionalities of the system.

Performance is one of the most important factors that support the change towards headless architecture, PWA, and other modern web technologies. The most common and fast way to measure key indicators of performance and other metrics of an individual website are tools, such as WebPageTest or Google Lighthouse.[30] Google Lighthouse is an open-source tool for generating automated website metrics and audits on performance, accessibility, SEO, best practices, and PWA compliance.

To conclude, there is a strong demand to switch to PWA storefront solutions. Even Magento has its own PWA solution released in beta. While the field might not be as mature, clients are ready to invest in production-ready solutions like Vue Storefront.

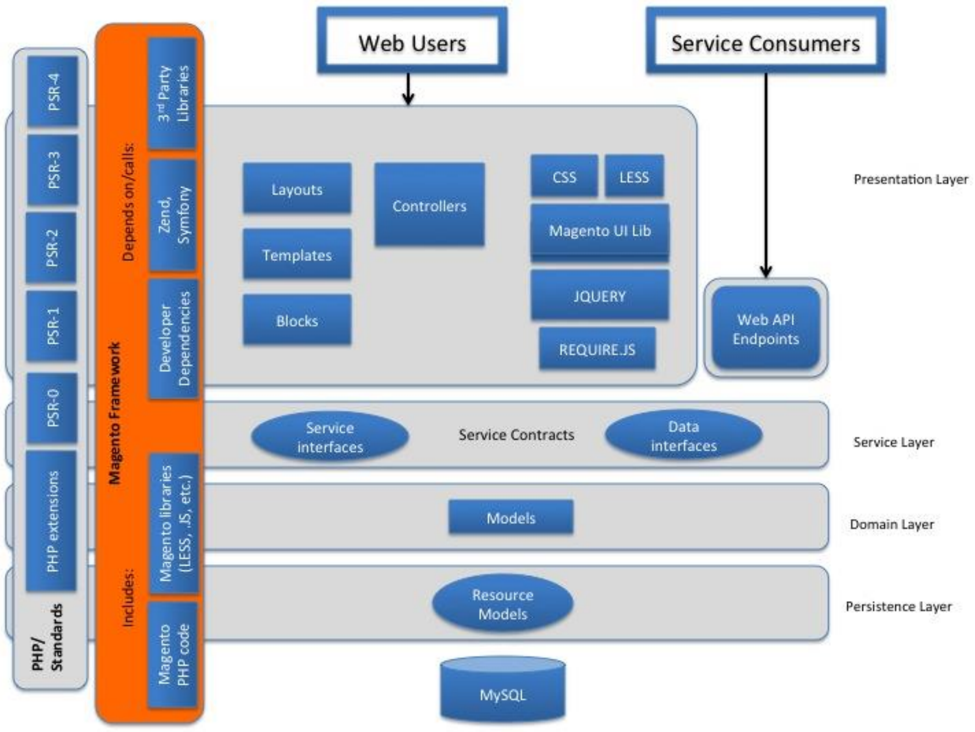


FIGURE 3. Magento architecture.[31]

3 STATE OF MODERN WEB FRONT-END (Fayjus)

Front-end technology, tools, and methodologies have been evolving rapidly in recent years. Web applications have progressed from simple static web pages to a very complex SPA and API first systems. As a result of this phenomenon, it has become tough to continuously keep up with all the changes and it might be considered overwhelming to developers.[32] Front-end architecture is a collection of tools and processes that aims to improve the quality of front-end code while creating a more efficient and sustainable workflow.[33] Therefore, the study and understanding of architectural patterns available in contemporary modern web front-end systems is needed to plan, design, and oversight a complex e-commerce system.

In order to discuss the state of modern web front-end and architectural patterns, it is crucial to understand the environmental and situational changes that are driving their evolvement. One of the most important driving forces for front-end architectural changes is the fact that the web has evolved to support more than just desktop computers. Today the web is accessed from a variety of end-user devices such as a mobile and a tablet. The mobile internet consumption has crossed the 50% mark and is expected to grow more than twice by 2022.[34] At the beginning of past decade, Google introduced its mobile-first strategy to address the growing web users using mobile devices. A plethora of Web APIs, such as File System, Camera, PWA, Hardware sensors, has evolved in order to provide a rich and uniform user experience across platforms.[35] Other notable driving forces include the importance of user experience as a deciding factor when it comes to service design[36] and upgrades in the network infrastructure.[37]

According to the *web.dev* platform maintained by Google, today's web applications have the following potentials: [38]

1. Achieving fast load times by optimizing front-end resources, setting a performance budget, and building a performance culture so that the application is available to a larger set of audience and maintains a higher conversion rate.
2. Making accessible applications and services by leveraging the semantic HTML for a diverse set of users.
3. Providing a consistent, reliable performance regardless of the quality of the network by using service workers and caching strategies for the offline or limited network usage.

4. Being immune to common web security vulnerabilities with security features such as HTTPS and CORS.
5. Maintaining discoverability with the proper usage of SEO.
6. Turning the application into a PWA to deliver native-like capabilities, reliability through building and enhancing them with modern APIs.
7. Constantly Measuring and optimizing user-centric performance metrics, namely First Contentful Paint (FCP), Largest Contentful Paint (LCP), First Input Delay (FID), Time to Interactive (TTI), Total Blocking Time (TBT), Cumulative Layout Shift (CLS).

In the book *Frontend Architecture for Design Systems,* it is mentioned that the four pillars of front-end architecture are – code, process, testing, and documentation.[39] Throughout the *Modern front-end architecture* section, these four pillars, along with relevant tools, technologies, and methodologies, would be studied, compared, and analyzed so that e-commerce systems, regardless of their chosen platform or complexity, could be developed and delivered leveraging the potentials of modern web.

3.1 Modern Web front-end architecture (Fayjus)

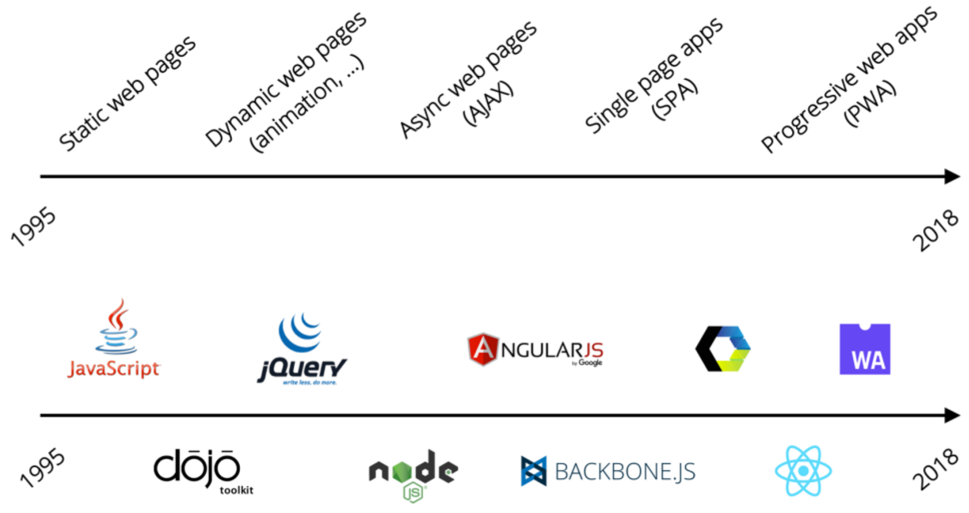


FIGURE 4. Evolution of front-end technology. [14]

From Figure 4, it could be derived that technologies have been changing at a high rate lately and have gone through a number of major revolutions. The technologies are estimated to continue this evolving process since there is no sign of slowing down. It might seem overwhelming to keep track of all the changes within the industry and next to impossible for a service provider to keep their development teams up-to-date while still delivering solutions to clients at a satisfactory rate. Even though each technology that is evolving is different from the other, they tend to share similar elements that became stable in front-end architecture over time and could be counted as the standards of modern web front-end. Some of these elements shown in figure 5 could be found in almost all the variants of modern web application architecture.[40]

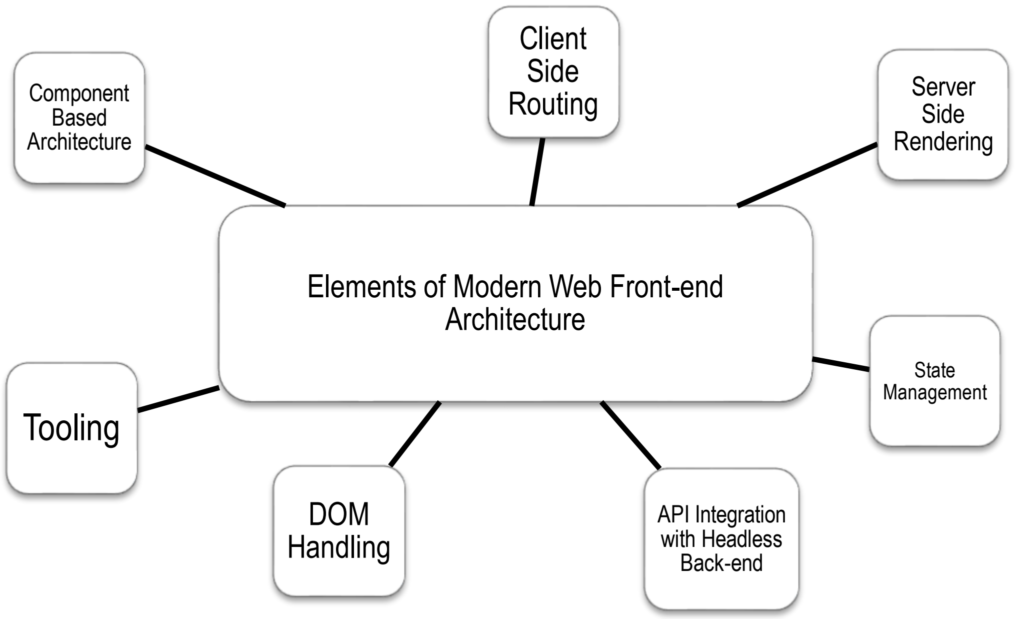

Component Based Architecture
Client Side Routing
Server Side Rendering
Elements of Modern Web Front-end Architecture
State Management
Tooling
DOM Handling
API Integration with Headless Back-end

FIGURE 5. Elements of modern web front-end.

These elements comprise of methodologies and patterns that have the potential to solve complex front-end problems that are elaborated individually in the upcoming sections. As mentioned in the *Introduction,* there are a number of uprising e-commerce PWA solutions available today as platforms. Each of these platforms have their front-end architecture which varies heavily from one another. Nevertheless, they often share similar traits as the reference architecture drawn in figure 6.

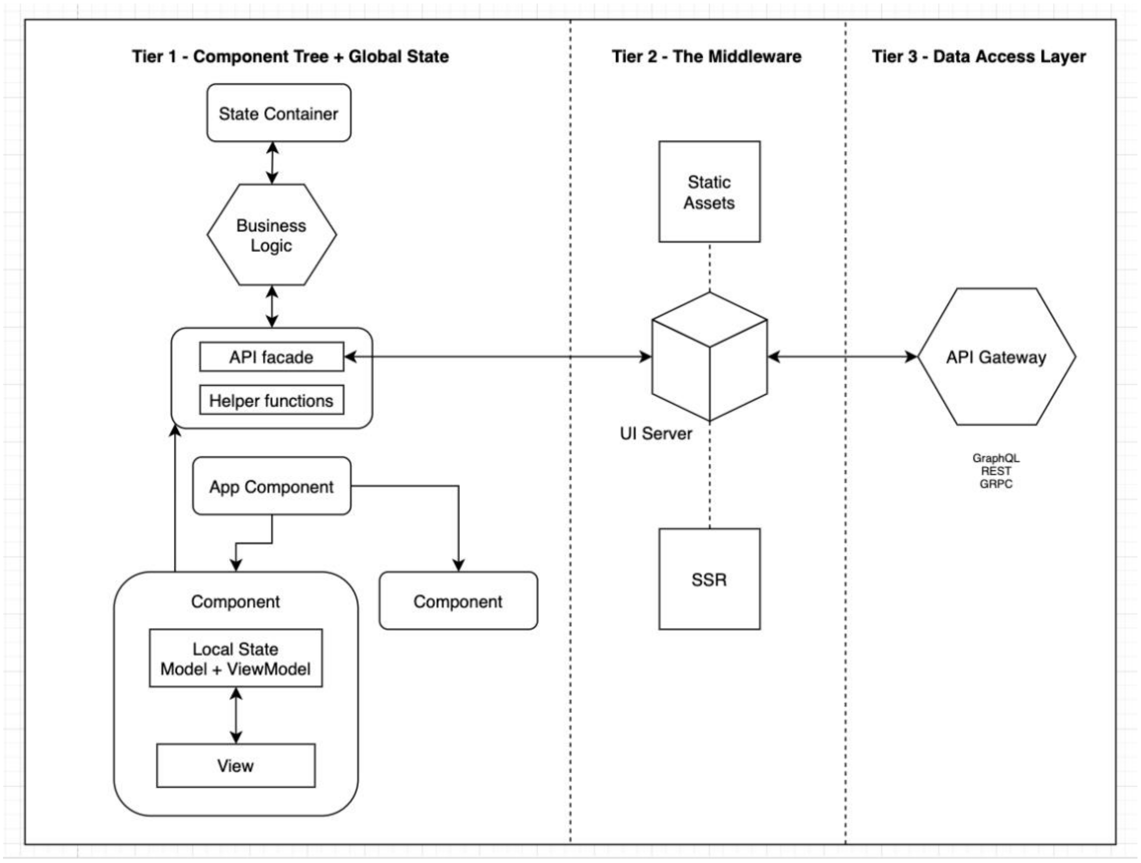


FIGURE 6. The reference architecture for modern web front-end. [40]

3.1.1 Component-based architecture (Dair)

Most of the popular JavaScript frameworks follow either MVC (AngularJS, Ember.js) or component-based architecture (React, Vue). In a component-based architecture (CBA) all the architectural layers of the module, such as design, business logic and external services, are defined in one place – component, similarly to feature-based architecture. CBA helps contain the logic within component and reduce scattered code. Components are able to communicate with other components or services independently. This makes them reusable, since it is easy to inject a component into different segments of the interface or use multiple components simultaneously.[41]

In CBA responsibilities are sliced *vertically*, whereas in MVC they are sliced *horizontally*. This means that the components deliver the whole feature, not just one architectural layer.[42] While CBA refines modularity and decouples the application in a way that makes it easier for software developers to reflect on a code, it may not work perfectly for each case. Code readability might suffer since all logic is contained within components. In addition to that, it might be tempting for developers to over-engineer simple features and treat every element of the UI as a component. [41][43]

Below is the example of a Vue component implemented in Vue Storefront. The Logo component is called to display website's logo image. In the component's template a route to the homepage is defined which wraps the `img` tag:

```
<template>
  <router-link :to="localizedRoute('/')" :title="$t('Home Page')" class="…">
    <img
      :width="width"
      :height="height"
      src="/assets/logo.svg"
      :alt="$t(defaultTitle)"
    >
  </router-link>
</template>
```

And since the component is reusable, it accepts parameters for the width and height of the image to match different use cases:

```
props: {
  width: {
    type: [String, Number],
    required: true
  },
  height: {
    type: [String, Number],
    required: true
  }
}
```

Components are easy to reuse since they contain all the required logic within them. The best example of a reusable component is when it is fully documented, thoroughly tested and designed to be as generic as possible, without any project-specific functionality.[44] CBA shortens the production and delivery time. It also frees up the resources required to ensure a quality delivery.[45]

3.1.2 State Management (Dair)

State management – is a set of processes and techniques that help to keep the data of different UI components synchronized.[46] For the state management to work, the application needs to have:

- a global state available to the entire application
- an ability to read and update it

State management is especially useful in single-page applications, where the UI content is constantly updating on user request or on other background processes.[47] Complex applications with interdependent components may suffer from poor data persistence, constant information flow and input-output (I/O) operations. State management is aimed to solve the arising issues in modern front-end development, by introducing a global store. For instance, state management unifies properties stored in various components in the application that may depend on each other. Components no longer have to depend on each other's properties since they are stored in a global state value.[48] This is especially convenient when components are not direct relatives in the component tree.[49] With state management libraries the properties of the application reside in the external store, available to all components in the application.

The implementations of Redux and Vuex, state management libraries for React and Vue, have been inspired by Facebook's Flux architecture, the original library for React state management. In the Flux pattern user interactions (Actions) are provided to the dispatcher, which invokes the store mutations. Store, in its turn, emits the *change* event causing components (View) to update state values and re-render (figure 7).[50]

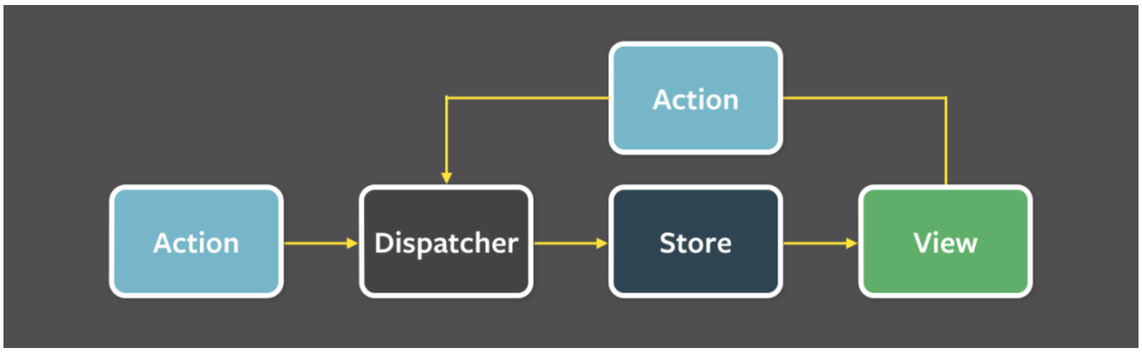


FIGURE 7. Flux data flow diagram. [50]

Vuex is a state management library for Vue. It is also used in Vue Storefront. Below is the example of how global state of the application can be declared. The breakpoints object is defined in the `uiStore` store which contains the state of different pieces of UI, such as overlay visibility, wishlist visibility:[51]

```
export const uiStore = {
  namespaced: true,
  state: {
    microcart: false,
    breakpoints: {
      small: 768,
      medium: 992,
      large: 1024
    },
    submenu: {
      depth: false,
      path: []
    }
  },
  mutations: {
    setMicrocart (state, action) {
      state.microcart = action === true
      state.overlay = action === true
    }
  },
  actions: {
    toggleMicrocart ({ commit, state }) {
      commit('setMicrocart', !state.microcart)
    }
  }
}
```

The `uiStore` is then fetched in different components, such as product page, as the computed property:

```
computed: {
  breakpoints () {
    return this.$store.state.ui.breakpoints
  }
}
```

State management libraries are designed to be flexible and not to restrict the structure of the application. They work best when used in the project from the beginning since they are not easy to integrate into the existing solutions.

3.1.3 Headless architecture (Dair)

As mentioned in the *Overview of Existing Methodologies*, headless architecture decouples front-end into a separate application which communicates with the system via an aggregation layer (API). This architecture is adapted in most of the popular PWA solutions. In this architecture front-end is completely detached and can even be distributed via Content Delivery Systems (CDN).[18] The headless architecture and the SPA enable a faster development, since the front-end application is more lightweight to be served locally.

3.1.4 Routing (Fayjus)

For web applications, routing is the process of mapping a URL to specific page resources.[52] In traditional MVC based architectures, routing was generally handled by the back-end part of an application. With every new request with the URL, a new HTML file is generated and send to the browser in server-side routing. This approach is not very efficient since the same assets are requested and fetched with every new request. In Single Page Applications, routing is handled by the client side of the application. Therefore, every time the client requests a new page, the front-end router handle the request and update both data and the UI accordingly. In the modern web front-end perspective, a router is a tool that synchronizes the currently displayed view with the address bar. In other words, it is the component of a modern web application that triggers the update for the view when there has been a change to the URL address.[53]

Modern UI frameworks usually provide either a built-in routing solution or a library. Angular has a built-in module named *Angular Router* that could be used to navigate users from one view to another. React has a separate library named *react-router* that provides extensive features such as Dynamic, Nested, and Responsive routing. For Vue, the official routing solution is *Vue Router.*

Some of the advanced routing features available from libraries such as Vue Router are dynamic route matching, nested routing, passing props to route components, HTML5 history mode, navigation guards, transitions, data fetching, scroll behavior controlling and lazy loading routes.[54] Furthermore, these features could also be integrated into front-end state management to create a persistent navigation experience. The benefits of client-side routing over server-side routing is that the routing libraries have these capabilities that could be used to create an advanced navigation logic for delivering better user experiences.[55]

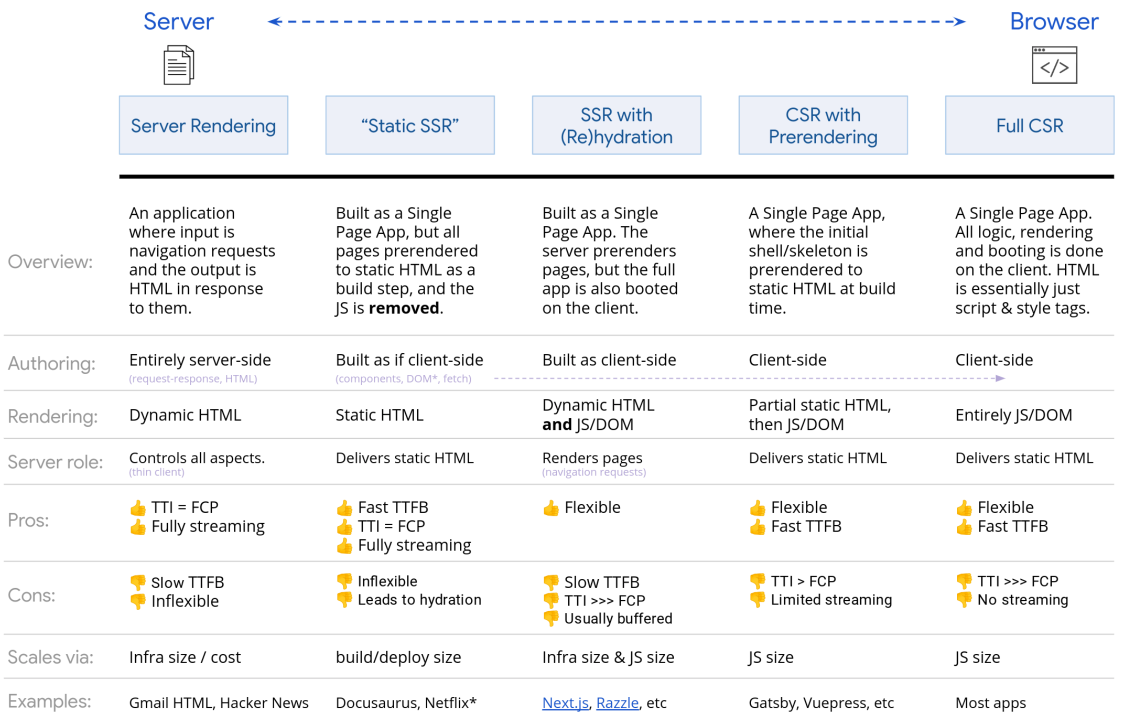
3.1.5 Rendering (Fayjus)

Rendering is the process by which contents are generated within an application. Rendering is one of the crucial architectural decisions that developers need to take while building an application since it impacts both the performance and discoverability of an application.

Server-side rendering (SSR) is the process of generating the HTML along with data on the server-side of an application. This technique has been very popular with server-side programming languages, e.g. Java, PHP and Python, where data is injected in HTML through a server-side templating engine and then sent to the user. The benefits of this approach are the following:

1. Search Engine Optimization: Content is crawlable by search engines, so websites and pages will appear in Google search results.
2. Social Media Optimization: When links are shared on social media, a preview will show up with the page title, description, and image.
3. Performance: Server-side rendered pages will load faster because the content is available to the browser immediately when it is reached.
4. User Experience: Similarly, to performance the content is available sooner, so the user is not waiting around looking at blank pages or loading spinners.[56]

However, since the appearance of SPA frameworks, it has become a common practice to render the front-end on the client-side, use AJAX requests to fetch data and content, and update the front-end dynamically. Nevertheless, the downside of this approach is that it violates the four above mentioned benefits of server-side rendering. Multiple new techniques of rendering have evolved, such as SSR with hydration and client-side rendering (CSR) with pre-rendering to overcome these shortcomings. The comparison between these rendering techniques are shown in figure 8.


Server
Browser

	Server Rendering	"Static SSR"	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is **removed**.	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side (request-response, HTML)	Built as if client-side (components, DOM*, fetch)	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML **and** JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. (thin client)	Delivers static HTML	Renders pages (navigation requests)	Delivers static HTML	Delivers static HTML
Pros:	👍 TTI = FCP 👍 Fully streaming	👍 Fast TTFB 👍 TTI = FCP 👍 Fully streaming	👍 Flexible	👍 Flexible 👍 Fast TTFB	👍 Flexible 👍 Fast TTFB
Cons:	👎 Slow TTFB 👎 Inflexible	👎 Inflexible 👎 Leads to hydration	👎 Slow TTFB 👎 TTI >>> FCP 👎 Usually buffered	👎 TTI > FCP 👎 Limited streaming	👎 TTI >>> FCP 👎 No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	Next.js, Razzle, etc	Gatsby, Vuepress, etc	Most apps

FIGURE 8. Comparison between different rendering mechanism.[57]

3.1.6 Handling DOM (Fayjus)

Most browser screens today are refreshed at 60fps, which means in order to provide the end-user with a smooth experience, the interaction and animations happening within the web application should match up with the browser refreshing rate. Therefore, the application needs to go through the pixel-to-screen pipeline (shown in figure 9) within a 10ms time budget for a smooth browsing experience.[58]

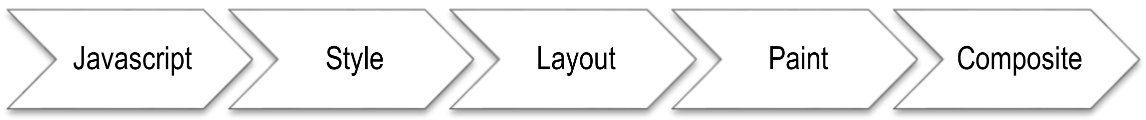


FIGURE 9. Pixel-to-screen pipeline

One of the key benefits of using modern UI libraries, such as Angular, React or Vue, is that these libraries or frameworks handle the communication between the DOM and data model of the application. [59] With previous generation libraries, such as jQuery, the interaction with DOM API was simplified, and developers had a direct control over DOM when building interactive web features. However, in large scale applications, developers often tend to mix business logic with DOM handling.[40] This kind of direct access and interaction with DOM API could turn out to be very expensive, and an application of large scale could easily hit the FPS bottleneck and result in an unreliable user experience.[60]

Modern UI frameworks, such as React and Vue, solve this issue by abstracting DOM handling with the Virtual DOM and with an optimization process called reconciliation.[61] The Virtual DOM is a JavaScript object that replicates an actual DOM of the HTML document. Every time data is changed within the application, the nodes in the Virtual DOM are updated instead of the actual one. Later, it is compared with the actual DOM, and specific changes are patched instead of refreshing the whole tree.

With such abstraction in DOM handling provided by modern JavaScript frameworks, it is easier to focus on the core business logic than optimizing performance, experience bottlenecks, and deliver better performance and productivity.[59]

3.1.7 Tooling (Fayjus)

It is essential to use the proper tools for automating tedious tasks on behalf of developers. A good set of tooling is another crucial architectural decision that could potentially impact all parts of the development lifecycle. According to the *Book of Modern Front-end Tooling,* front-end tools could be divided into sections depending on the part of the development lifecycle they are impacting, as demonstrated in figure 10.[62]

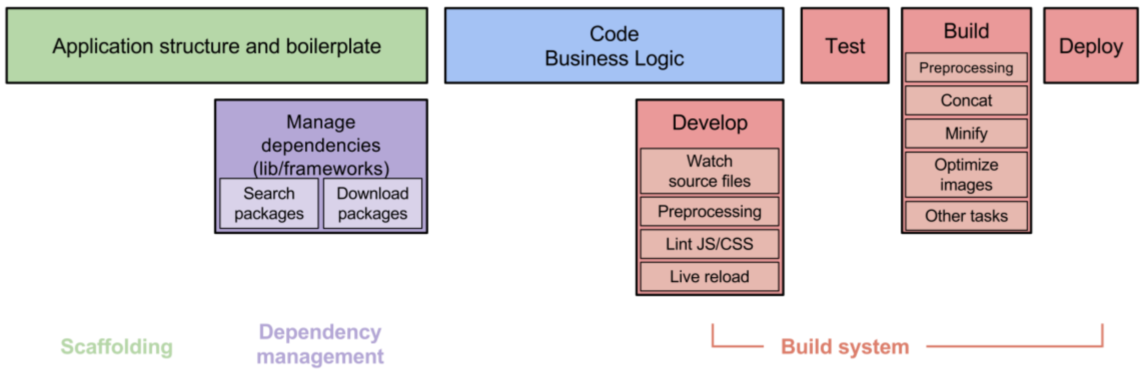


FIGURE 10. Application development lifecycle and tooling

As depicted in the figure 10, the tools are firstly categorized into three sections, Scaffolding, Dependency management, and Build System. Scaffolding or bootstrapping is the process where the application structure and boilerplate are created. This could often be a painful experience for developers since they might always not be aware of the best practices and decisions made during this process which has a great impact on the application as it scales. The second section is labeled as Dependency management. In large collaborative applications, developers often reuse work from other projects or third-party modules to simplify the development process. The tools related to this section handles the process of finding and adding these modules to the project and managing versions automatically. The final section is the Build System. Tools that are categorized here impacts the development and delivery workflow directly. During developments, tools could watch source files, provide lint, and hint and live reload as the developers write the application. Moreover, during the delivery, the code could be tested, build, and deployed automatically with the right tools. In Table 1, A list of available tools in each of these sections, their objective, and impact area are provided. The usage data for the tools are collected from *Modern Front-end Tooling Survey 2019. [63]*

Table 1. Modern front-end tools, objective, usage and impact area

Section	**Subsection**	**Objective**	**Popular Tools & Usage**	**Impact Area**
Scaffolding		Kickstarting new projects, prescribing best practices, and tools	Yeoman, vue-cli, create-react-app	Project Management

Dependency Management	JavaScript Package Manager	Managing and installing dependencies	NPM (65.39%), Yarn (29.78%)	Project Management
Build System	CSS Preprocessor	CSS preprocessor is a program that generates CSS from the preprocessor's unique syntax. Most CSS preprocessors will add some features that do not exist in pure CSS, such as mixin, nesting selector, inheritance selector. These features make the CSS structure more readable and easier to maintain.[64]	Sass (77.27%), PostCSS (28.72%), Less (14.04%)	Development
	Task runners	Automating tedious tasks such as compiling template, CSS and JavaScript, auto prefixing, minifying files. Optimize images.	NPM Scripts (64.33%), Gulp (18.54%), Grunt (2.53%)	Development, Test, Build, Deploy & Performance
	JavaScript Module Bundler	Providing a module system to the application and bundling multiple JavaScript modules into a single JavaScript file for the browser to run	Webpack (73.34%), Parcel (2.83%), RequireJS (1.23%)	Development, Build
	JavaScript Linting	Checking code to make sure it is error-free and adheres to a particular style guide, which eventually improves overall readability and consistency	ESLint (76.07%), JSLint (4.76%), JSHint (2.23%)	Development

	JavaScript Testing	Creating and performing automated Unit, Integration, and Functional tests.	Jest (44.86%), Jasmine (19.47%), Cypress (12.58%)	Test
	Performance Testing	Web Performance Testing is executed to provide accurate information on the readiness of an application through testing the website and monitoring the server-side application. This is done by simulating load as close as possible to the real conditions in order to evaluate if the application will support the expected load.[65]	Lighthouse (52.11%), WebPageTest (24.29%)	Performance
	Accessibility Testing	Web accessibility testing is a subset of usability testing where the users under consideration have disabilities that affect how they use the web. The end goal, in both usability and accessibility, is to discover how easily people can use a website and feed that information back into improving future designs and implementations.[66]	Colour Contrast Checker (22.20%), Screenreader (15.44%), WAVE (9.48%)	Accessibility

3.2 Testing and design system (Dair)

Other benefits of modern front-end architecture include creating easy and automated web testing and accessibility audits written in JavaScript. This means that front-end developers will have the ability to write unit and functionality tests by themselves, improving the quality engineering during the development.

Likewise, component-based architecture facilitates developers to apply *atomic design* principles and implementing design systems. This may result in an improved collaboration between UI/UX engineers and front-end developers. UI/UX engineers are able to see, comment and advice the improvements that will only require a change to a single entity and they can track the development of components during the early stages.

3.2.1 Automated Web Testing (Dair)

Software testing is the activity of running a series of dynamic executions of software programs after the software source code has been developed.[67] UI implementation and functionality are generally of highest importance in web development testing. Testing is performed to uncover and correct as many potential errors as possible before delivering features to the client. By testing, development teams can make sure that the website is functioning properly, and features are production-ready.[68] Testing can be divided into *manual* and *automated*. Manual testing requires manual execution of test cases within the web application through a web browser. Manual execution is slow and can be expensive, since it requires additional environments for testing and can be prone to human factor as the tester might make typographical errors or omit steps in the test scenario.[69]

Nowadays, most of the testing methods can be automated. The most important automated test types for the website include:

- Unit tests – individual functions, modules and components of the application. It is done by supplying inputs and predicting output in the code.
- Integration tests – testing of the processes across multiple units (modules, components).
- Functional tests (end-to-end and smoke testing) – black-box testing which is performed in the browser[70]

There is a significant amount of testing frameworks with a rich set of tools that are compatible with Node.js. A web application may take a leverage of unit testing suites (Jest, Mocha, Jasmine) in combination with end-to-end (E2E) testing tools (Puppeteer, TestCafe, Selenium, Cypress). Vue Storefront uses Jest and Cypress by default, both considered to be most favored by developers in terms of developer awareness, interest and satisfaction.[71]

While manual, exploratory testing and usability testing are not discouraged, automated tests help identify scenarios that can break the system. Front-end developers working on the new functionality can write tests themselves and include them in the code review phase, so that it can be checked by other developers. This change can result in better testing and improvement of quality of the deliverables.

3.2.2 Automated Web Accessibility Testing (Dair)

In addition to technical excellence and potential advantages for the development workflow, the modern front-end technology stack can help improve web accessibility. Developing towards total accessibility of the website is still a moral and ethical practice but the demand is growing. In Europe, for example, *EU Web Accessibility Directive* was updated in 2018, making all public sector websites and applications in EU member states to implement, enforce and maintain accessibility standards or risk legal penalties.[72]

Accessibility of the web application does not exclusively imply users with physical disabilities, but also the practice of making websites accessible for users with different external constraints – whether it is a small screen resolution, an older browser or a slow network connection.[73] The guidelines set by the World Wide Web Consortium's (W3C) *Web Content Accessibility Guidelines* (WCAG) can be followed to implement accessibility support in the web application. The guidance focuses on four principals of web accessibility:

- Perceivable – information should be presented with alternatives, i.e. images should have meaningful captions or other alternatives; elements of UI should be distinguishable.
- Operable – giving users enough time to take actions, make functionality work for devices that function similarly to the keyboard. Applications should be easy to navigate.
- Understandable – dynamic changes happen in a predictable way and the application assists users in taking decisions.

- Robust – compatibility with current and future assistive technologies (AT), such as screen readers or alternative input devices (head pointers, eye tracking).[74][75]

There are many accessibility testing tools for evaluating websites. They include analyzing tools, such as Axe Accessibility Testing – technology used in Google Lighthouse, color use evaluators, screen readers (NVDA, Chrome Vox) and other assistive technology emulators.[76]

Web accessibility audits provide detailed assessments of the website and identify needed improvements for the product to achieve compliance with standards, such as ADA, Section 508 and WCAG 2.0.[77] It benefits both creators and users of the website, since it helps writing semantic HTML that improves accessibility, usability and SEO, demonstrates good ethics and morals and is protected by law in some states.

3.2.3 Design System Management (Dair)

A *design system* is a collection of reusable components that can be integrated into the UIs of multiple systems.[78] Major companies, e.g. Airbnb, Uber and IBM, were among the first to incorporate their unique design systems. One of the core values of a working design system is a single source of truth available for design teams when drafting a UI. This allows for a scalable UI language and streamlined UX guidelines. Leveraging a single design source can assist in creating consistent user experiences for the companies that aim to persist an identical user experience among multiple services, for instance, cross-channel marketing, such as multichannel or omnichannel. Building a library of design patterns, rules, and UX guidelines prevents inconsistencies when shipping e-commerce products at a large scale.[79]

A *style guide* is a static document that describes the design system. It demonstrates the look and feel of the UI components, describes use cases for UI patterns, correct typographic scales, and other visual characteristics of the design components.[80] While style guides provide classification of components, implementing and maintaining a design system is a complex process.[81] However, style guides are a minimum requirement for a working design system. Maintaining a design system encompasses pattern libraries, style guides, as well as any other artifacts, and serves as a rule book for the development teams. Design systems direct designers to create designs in an intentional way.[82]

ATOMIC DESIGN (DAIR)

Atomic design is a methodology, created by Brad Frost, which divides UI elements into five distinct groups:[83]

1. Atoms – basic HTML elements (buttons, labels, inputs).
2. Molecules – relatively simple groups of UI elements functioning together as a unit (lists, menu items, search box).
3. Organisms – relatively complex UI components composed of groups of molecules and/or atoms and/or other organisms (header, footer, modal)
4. Templates – page-level objects that place components into a layout and articulate the design's underlying content structure (home page, category page, account page)
5. Pages – specific instances of templates that show what a UI looks like with real representative content in place.

UI methodologies, such as atomic design, bring logic and structure to individual screens of the application. In the open-source web one of the most popular solutions for implementing style guides and project-specific component libraries is Storybook.[84] Storybook is the industry-standard component explorer for developing UI components in isolation.[85] It has a rich add-on library for assessing responsiveness, accessibility, different use cases, and various properties of the components. It works with multiple JavaScript frameworks, namely React, Vue and Angular, and has a support for design tool integrations such as InVision's *Design System Manager* (DSM) and Figma.[86] Storybook can serve as a bridge between designers and developers, allowing certain level of control over the end result[87] via "design tokens".[88]

Connecting design and development helps teams to identify edge cases before the actual issues occur and respond to flaws during the early stages of development.[89] Neither developers nor designers can predict each possible scenario of how the UI will look for every use case. Screen resolutions or dynamic contents can break the design. Therefore, there is a need for developers and UI/UX engineers to have a framework for collaborating during the implementation of design to build more thoughtful and defensive UIs.[89]

3.3 Workflow optimization (Dair)

Headless architecture decouples front-end application from the back-end, making a way for a faster delivery and better separation of concerns. This can positively affect the way user stories are defined in agile development, improving the independence of related features. Front-end development patterns, e.g. CBA and state management, allow building complex features without producing sophisticated code. In addition to that, web testing and accessibility audit automation and the possibility to implement fully functional design systems with modern open-source tools like Storybook is of great value.

3.3.1 Workflow optimization with automated web testing (Dair)

In agile development, regular testing is an essential part of quality assurance. Testers (QA engineers) usually belong to the team along with developers and continuously test every change in the code.[90] Agile methodology helps to detect bugs and identify errors in the early stages of development as well as to receive feedback from QA engineers iteratively.[91]

Automated tests can improve the continuous delivery of e-commerce projects and are crucial to integrating *continuous testing* into the CI/CD pipeline.[92] They help detect bugs and keep track of the system status on every deployment. In addition to that, developers writing unit and E2E tests are involved in improving quality engineering, along with QA engineers. This shared responsibility increases the value and time that development teams spend on improving the quality of front-end deliverables.

Performance testing identifies technical limitations that may negatively affect different users.[93] The developers should be able to prevent an outage, predict recovery time, and retrieve practical information on the resources of the system. QA engineers can evaluate performance and request performance profiling when needed.

3.3.2 Workflow optimization with design systems (Dair)

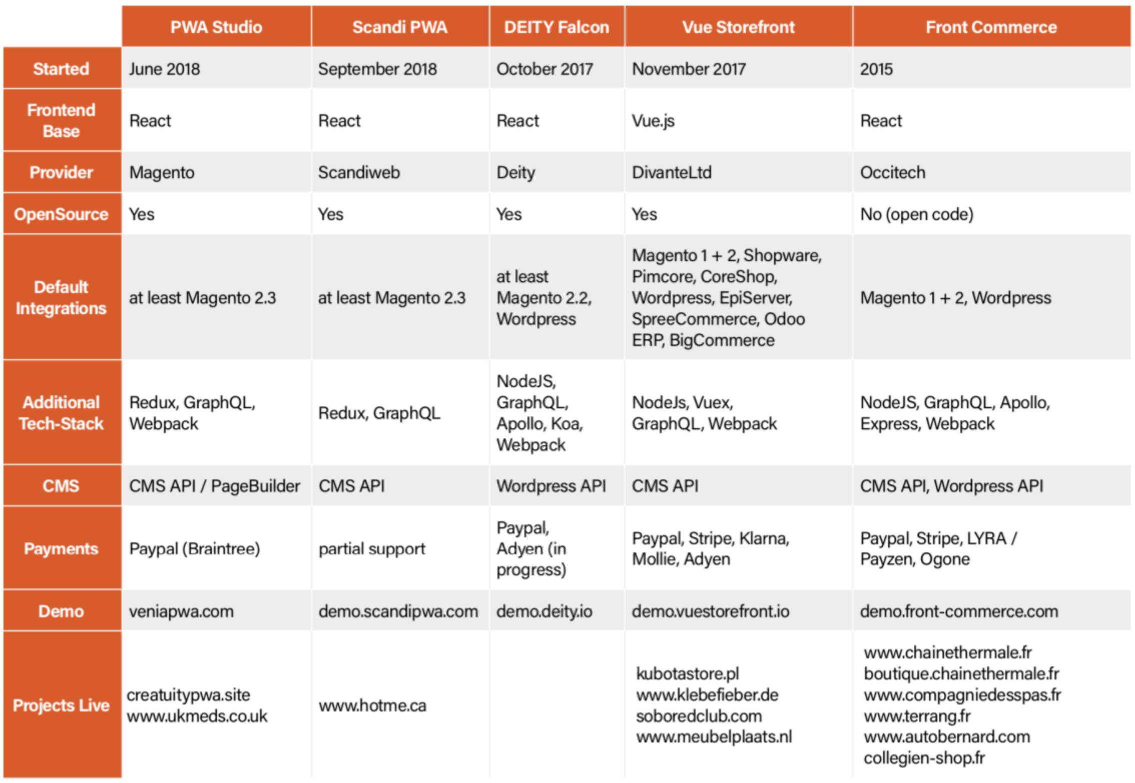
During development of the UI, lack of communication between developers and designers may hinder the process and lead to various misunderstandings regarding the implementation.[94] When design teams focus on building pages, without the atomic design principles, it affects the actual implementation and results in UI inconsistency and bugs. In this case, design does not help developers in defining how to break the UI down into smaller entities and build components in the code level.

Benefits of implementing design systems for the workflow optimization are following:

1. Design systems allow designers to create a consistent UI/UX, which is fast to build and develop due to the narrow focus on components and their reusability.
2. Components are built with different types and states of the program in mind, which reduces bugs.
3. Components can be tested individually.
4. Practice of systematic design reduces time of iterative processes in the web development teams. Designers have an opportunity to directly collaborate with developers in certain matters.

3.4 Emerging E-commerce Platforms Based on Modern Web Front-end (Fayjus)

The exact architecture of a web front-end system might depend on specific project requirements. Nonetheless, as presented in the *Modern front-end architecture*, some baseline architectural patterns are common throughout most modern web front-end solutions. It might turn out a costly and repetitive procedure to integrate modern front-end methodologies, tools, technologies, and architect a solution from scratch for every different project developed by a solution provider. This triggering concern led many service providers and developer communities to create their own baseline modern PWA framework and platforms and making them open-sourced. A comparison between all such front-end PWA based platforms for e-commerce that are available today is shown in figure 11.



	PWA Studio	Scandi PWA	DEITY Falcon	Vue Storefront	Front Commerce
Started	June 2018	September 2018	October 2017	November 2017	2015
Frontend Base	React	React	React	Vue.js	React
Provider	Magento	Scandiweb	Deity	DivanteLtd	Occitech
OpenSource	Yes	Yes	Yes	Yes	No (open code)
Default Integrations	at least Magento 2.3	at least Magento 2.3	at least Magento 2.2, Wordpress	Magento 1 + 2, Shopware, Pimcore, CoreShop, Wordpress, EpiServer, SpreeCommerce, Odoo ERP, BigCommerce	Magento 1 + 2, Wordpress
Additional Tech-Stack	Redux, GraphQL, Webpack	Redux, GraphQL	NodeJS, GraphQL, Apollo, Koa, Webpack	NodeJs, Vuex, GraphQL, Webpack	NodeJS, GraphQL, Apollo, Express, Webpack
CMS	CMS API / PageBuilder	CMS API	Wordpress API	CMS API	CMS API, Wordpress API
Payments	Paypal (Braintree)	partial support	Paypal, Adyen (in progress)	Paypal, Stripe, Klarna, Mollie, Adyen	Paypal, Stripe, LYRA / Payzen, Ogone
Demo	veniapwa.com	demo.scandipwa.com	demo.deity.io	demo.vuestorefront.io	demo.front-commerce.com
Projects Live	creatuitypwa.site www.ukmeds.co.uk	www.hotme.ca		kubotastore.pl www.klebefieber.de soboredclub.com www.meubelplaats.nl	www.chainethermale.fr boutique.chainethermale.fr www.compagniedesspas.fr www.terrang.fr www.autobernard.com collegien-shop.fr

FIGURE 11. Comparison between emerging e-commerce front-end platforms leveraging modern front-end.[95]

Since these solutions are still at an initial phase with growing communities and implementations, a closer look at the community statistics as shown in Table 2 might also assist in deciding the baseline solution.

Table 2. Community statistics for emerging e-commerce front-end platforms.[96]

	Angular Storefront	**Deity Falcon**	**Front Commerce**	**PWA Studio**	**Scandiweb PWA**	**Spartacus SAP**	**Vue Storefront**
Community Status	Small	Small	Selected Partners	Strong	Small	Small	Strong
GitHub Stars	-	313	-	420	35	192	4.903
Contributors	8	17	-	74	5	38	142
Accepted Pull Requests	-	242	-	572	16	1.115	1.627
Number of forks on GitHub	-	47	-	217	14	85	946

As it appears in Figure 11 And Table 2, among all the open-source front-end platforms available today, Vue Storefront has the most significant number of live projects and the highest community involvement. It is only outnumbered by Front Commerce when it comes to live projects, but as the latter is not an open-source platform, it might not be an ideal platform for many solution providers. The PWA studio provided by Magento also shows an active community status just as Vue Storefront. However, PWA studio has support for only one payment provider so far, which should be taken into account before moving forward with the framework.

4 IMPLEMENTATION (Fayjus)

The previous chapters of this thesis have established the following motivations for implementing a design system and modern web front-end based e-commerce system:

1. The business and technological drawbacks of using outdated front-end systems in e-commerce applications.
2. The shortcomings and experiences of existing solutions.
3. The availability of modern front-end methodologies, tools, workflow, and platforms and their impact on developing and delivering performant applications.

The aim of this chapter is to document the implementation process of a modern e-commerce storefront leveraging modern web front-end.

4.1 Bootstrapping the Front-end (Fayjus)

According to the investigation in *Emerging E-commerce Platforms Based on Modern Front-end* (Ch. 3.2), **Vue Storefront** (VSF) has the most significant number of live projects and the highest community involvement among the list of e-commerce PWA platforms as of today. It was decided that VSF would be used as the base front-end platform among all the platforms introduced and compared in the previous chapter. The architecture for setting up a VSF based e-commerce application is shown in figure 12:

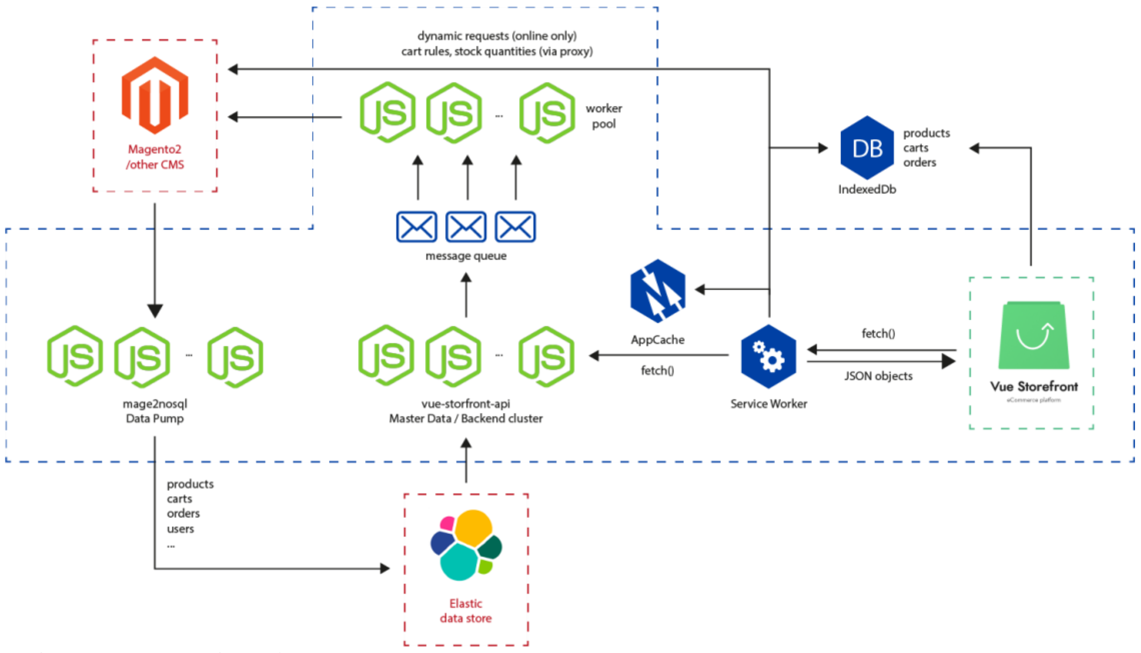


FIGURE 12. Vue Storefront Based Application Architecture.[97]

From the architecture, it can be derived that Magento 2 is still acting as the back-end of the e-commerce application. The data from Magento 2 is pumped into an ElasticSearch (ES) – Elastic data store, a NoSQL database. From there, the Vue Storefront API (VSF-API) application serves the data to front-end, Vue Storefront. That is the most common flow for a VSF application. In some specific cases, both VSF and VSF-API can also dynamically request and fetch data from Magento REST API.

The bootstrapping and deployment process of Magento 2, Vue Storefront API and Elastic data store would not be documented as it is out of scope of this thesis. So, only the process of bootstrapping VSF would be described. A VSF application has three most important parts in its project architecture and the SoC should be maintained while development:

- **Vue Storefront Core** (`core`): The core part of VSF is the section where most of the business and data resolving logic resides, which contains all the entry points, SSR behavior, build process, in-app libs, and helpers. The modification of the core files was prohibited in the official VSF documentation during implementations in order to stay up-to -date with its features and security.[98] During the implementation, this guideline was strictly followed. In cases where core needs modification, it was overridden within a theme following the guidelines

- **Vue Storefront Module** (`core/modules` and `src/modules`): The modules of VSF contain the e-commerce business logic. Each module contains an encapsulated feature, such as Cart, Wishlist, Category or third-part integration.
- **Vue Storefront Themes** (`src/themes`): This part of VSF is the actual store implementation that contains the markup and style of the store. The business logic implemented in core and modules could also be extended here. VSF comes with a default theme out of the box that can be fully customized and extended. Figure 13. depicts SoC between VSF Core and VSF themes and the flow of business logic from the core to the components,

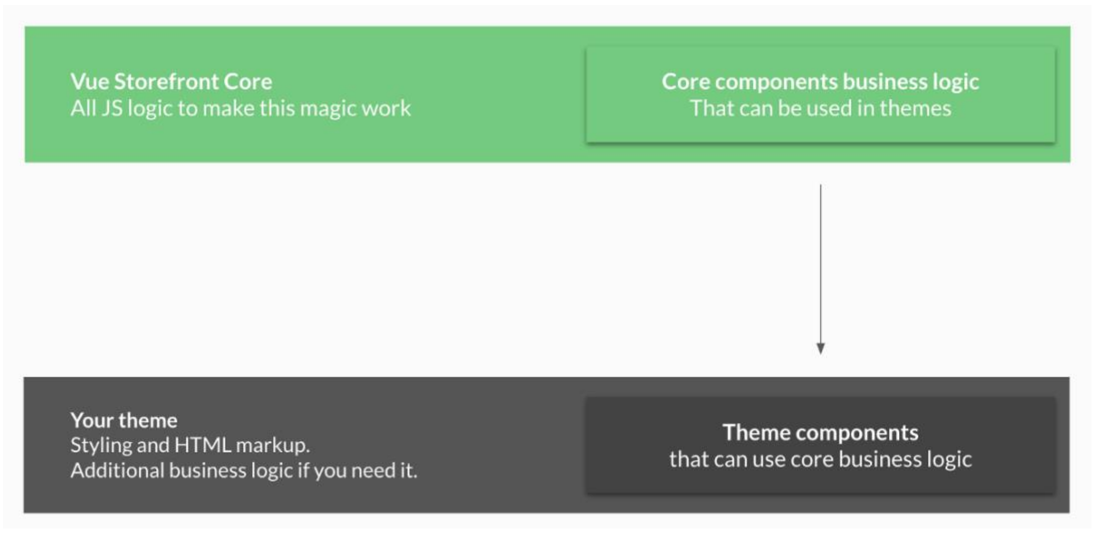

Vue Storefront Core
All JS logic to make this magic work
Core components business logic
That can be used in themes
Your theme
Styling and HTML markup.
Additional business logic if you need it.
Theme components
that can use core business logic

FIGURE 13. Vue Storefront – Connecting the core with theme.[99]

Apart from that the VSF application also contains a JSON configuration file as `config/default.json` which is responsible for configuration such as the active theme, back-end API addresses and multistore setup. The configurations determine the connection between VSF and VSF-API.

In the official documentation, two methods for setting up VSF are provided, VSF comes with an installer for a more user-friendly installing approach. However, a manual installation process has been followed since it provides more control over the configurations. In order to get a VSF application up and running in a local environment along with a custom theme, the following steps were followed in compliance with the VSF version 1.11.0:

1. The step one of the installation process was to clone the official VSF repository:

```
git clone https://github.com/DivanteLtd/vue-storefront.git test

cd test
```

2. After that the `default.json` file was duplicated and renamed as `local.json` from configuration folder. The `default.json` file contains the configurations for the development and production environment. On the other hand, the `local.json` file was created and intended to contain configurations for the local environment:

```
cp config/default.json config/local.json
```

At this point, the API and ElasticSearch configuration could be considered the most crucial step. VSF will throw the default error page upon landing, due to the unavailability or misconfiguration of these options. The VSF-API and ElasticSearch instances could run locally or remotely in theory. Nevertheless, during this implementation process, the local API and ElasticSearch instances were used.

```
"api": {
  "url": "http://localhost:8080"
},
"elasticsearch": {
  "httpAuth": "",
  "host": "/api/catalog",
  "index": "project_magento_en",
  "min_score": 0.02,
  "csrTimeout": 5000,
  "ssrTimeout": 1000,
  "queryMethod": "GET",
  "disablePersistentQueriesCache": true,
  "searchScoring": {
    "attributes": {
      "attribute_code": {
        "scoreValues": { "attribute_value": { "weight": 1 } }
      }
```

```
   },
   "fuzziness": 2,
   "cutoff_frequency":  0.01,
   "max_expansions": 3,
   "minimum_should_match": "75%",
   "prefix_length": 2,
   "boost_mode": "multiply",
   "score_mode": "multiply",
   "max_boost": 100,
   "function_min_score": 1
  },
  "searchableAttributes": {
   "name": {
    "boost": 4
   },
   "sku": {
    "boost": 2
   },
   "category.name": {
    "boost": 1
   }
  }
 },
```

Also, store configurations, e.g. multistore and store information were considered and configured according to the documentation.

```
 "defaultStoreCode": "",
 "storeViews": {
  "multistore": true,
  "commonCache": false,
  "mapStoreUrlsFor": ["en","fi"],
  "en": {
   "storeCode": "en",
   "storeId": 1,
   "name": "English Store",
   "url": "/en",
   "appendStoreCode": true,
   "elasticsearch": {
     "host": "/api/catalog",
     "index": "purewaste_magento_en"
   },
   "tax": {
     "defaultCountry": "FI"
   },
   "i18n": {
     "fullCountryName": " Finland",
     "fullLanguageName": "English",
     "defaultCountry": "FI",
     "defaultLanguage": "US",
     "defaultLocale": "en-US"
   },
   "seo": {
     "defaultTitle": "Vue Storefront"
   }
  },
  "fi": {
   "storeCode": "fi",
   "storeId": 2,
   "name": "Finnish Store",
   "url": "/fi",
   "appendStoreCode": true,
   "elasticsearch": {
     "host": "/api/catalog",
     "index": "purewaste_magento_fi"
   },
   "tax": {
     "defaultCountry": "FI"
   },
   "i18n": {
     "fullCountryName": "Finland",
     "fullLanguageName": "Finnish",
     "defaultCountry": "FI",
     "defaultLanguage": "FI",
     "defaultLocale": "fi-FI"
   },
   "seo": {
```

```
    "defaultTitle": "Vue Storefront"
   }
  }
 },
```

3. After the configurations, the VSF application could be launched in the localhost, and the development process could be started. There were two modes for launching the application. In the *Legacy* mode, the application could be launched with yarn and without any docker container. However, the *Standard* mode launches the application inside a docker container. During this implementation process, the application was launched in the Legacy mode since the development team did not have much docker expertise and the differences in mode do not affect the local FE development or the CI/CD workflow designed by the DevOps team. The following commands were needed to run the application locally in the Legacy mode:

```
yarn build
yarn dev
```

4. The application was up and running in *localhost:3000* at this point. The theme that was visible is the default one shipped with VSF. Most projects may require a custom theme to be developed to establish the visual identity from the design. All themes in VSF were found under the `src/themes` folder. A new theme was created following the VSF documentation by copying the default theme (`default` folder in `src/themes`) and renaming it to the desired theme name. The result folder structure looked as it is in figure 14.

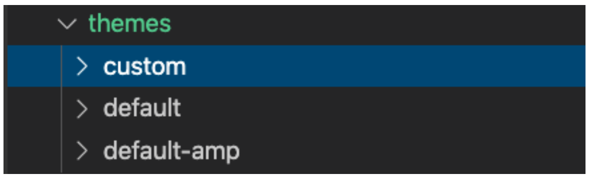


FIGURE 14. VSF themes folder structure

Next the name property in theme's `package. json` was updated and configurations from `config/default.json` and `config/local.json` were updated accordingly:

```
{
  "name": "theme-custom",
  "version": "1.0.0",
  "description": "Default theme for Project X",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "cd ../../../ && node core/scripts/server"
  },
  "author": "Vaimo",
  "license": "MIT",
  "dependencies": {
    "bodybuilder": "2.2.13",
    "vue": "^2.6.6",
    "vue-carousel": "^0.6.9",
    "vue-no-ssr": "^0.2.2",
    "vue-progressbar": "^0.7.5",
    "vue-zoomer": "^0.3.6",
    "vuelidate": "^0.6.2",
    "vuex": "^3.0.1"
  },
  "publishConfig": {
    "access": "public"
  }
}
```

```
"theme": "theme-purewaste"
```

After making these changes, it was needed to run the `yarn install` command, so that the newly created theme is found and installed.

4.2 Elements of Vue Storefront Architecture (Fayjus)

In this section, elements of Vue storefront architecture such as state management and CSS methodology, would be investigated for a better understanding during development.

4.2.1 State Management in Vue Storefront (Fayjus)

VSF uses Vuex, a state management pattern library created specifically for Vue JS applications. The state management pattern and its effectiveness in creating a modern web front-end have been explained in detail in *State Management* (chapter 3.2.2). The state management pattern implementation of VSF could be visualized from the figure 15.

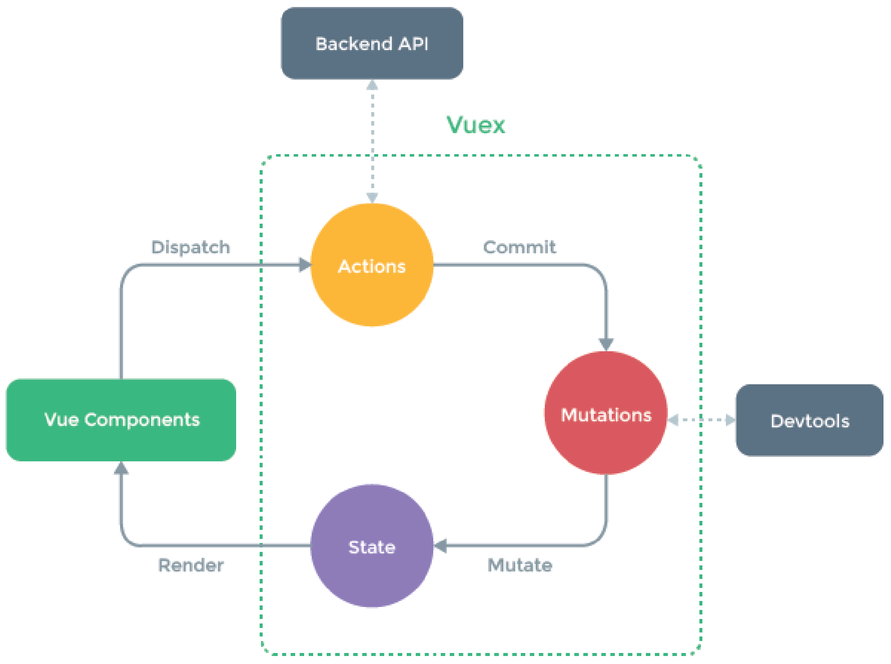


FIGURE 15. State Management Pattern as in Vuex.[100]

The actions, mutations and states are usually stored inside VSF `modules` under the store directory. For example, user related actions, mutations and states are stored inside `core/modules/user/store`. Also, the VSF theme might also contain Vuex stores that is relevant to determine the UI behavior. One such store is, `themes/{themeName}/store/ui.ts`, where all the UI related actions, mutations and states are defined. A list of all the states used in the application initially could be found in `core/state/index.ts` and it is shown below.

```
const state = {
  version: '',
  __DEMO_MODE__: false,
  config: {},
  cart: {},
  checkout: {},
  cms: {},
  compare: {},
  product: {},
  shipping: {},
  user: {},
  ui: {},
  newsletter: {},
  wishlist: {},
  attribute: {
    list_by_code: {},
    list_by_id: {},
    blacklist: [],
    labels: {}
  },
  category: {
    current_path: '',
    current_product_query: {},
    current: {
      slug: '',
      name: ''
    },
    filters: {}
  },
  stock: {
    cache: []
  },
  storeView: {},
  twoStageCachingDelta1: 0,
  twoStageCachingDelta2: 0,
  twoStageCachingDisabled: false,
  userTokenInvalidated: null,
  userTokenInvalidateAttemptsCount: 0,
  userTokenInvalidateLock: 0
}
```

4.2.2 CSS Methodology in Vue Storefront (Fayjus)

VSF uses a combination of *Atomic CSS* methodology and *Scoped CSS* to implement all the CSS rules in the default theme. The following guidelines were established to build the CSS technology stack shown in Table 3:[101]

1. CSS should be easily maintainable to imply the smallest possible files footprint.
2. Flexbox and reusable atomic classes should be used to prevent the rapid growth of CSS files.
3. Nesting of classes should generally be avoided (the maximum nesting level is 1) to maintain understandability and to provide an easier debugging experience.
4. All CSS files should be put inside a `src/themes/{theme_name}/css` folder.
5. Atomic classes should be created on-demand to avoid unused CSS.

Table 3. CSS Technology Stack in VSF default theme.

Technology	**Solution**
CSS Preprocessor	SASS
Layout	Flexbox
Methodology	Atomic CSS, Scoped CSS
Automation	PostCSS

4.3 Developing Reusable Components (Fayjus)

During the implementation process, a customized theme was implemented from the design. VSF provided most of the components required for the custom theme as part of the default theme. So, in order to make the custom theme look as it is in the design, it was mostly stylings that needed to be changed in the cases where the business logic and structure are the same. Still, there were some elements in the design for which custom components were needed to be created from scratch. This section would describe the challenges and processes of such components.

VSF, by default, had a sidebar menu implemented in the default theme. However, as per the design, the custom theme needed a horizontal menu for Desktop while keeping the sidebar for a mobile. The design for the custom component is shown in figure 16.

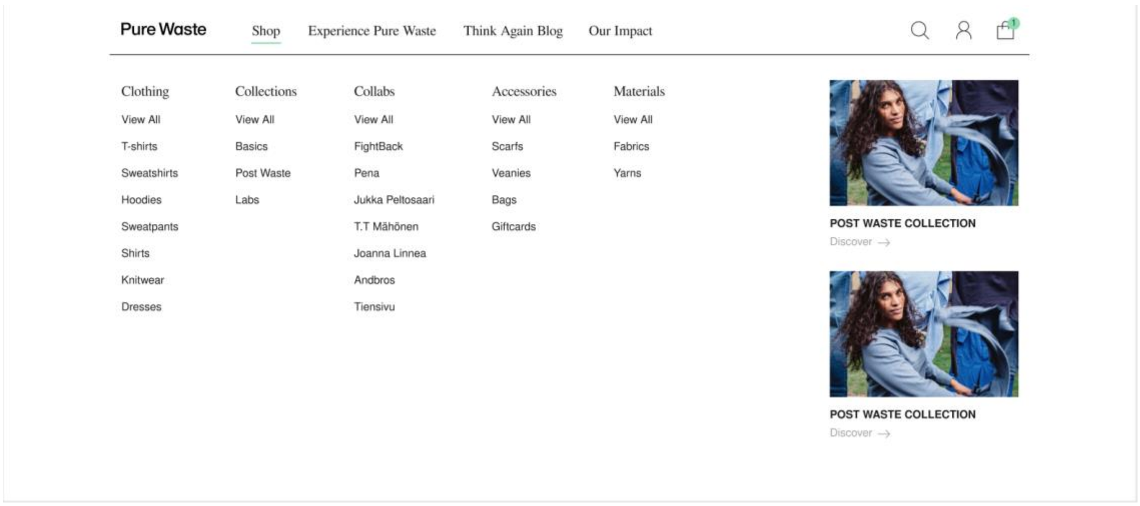
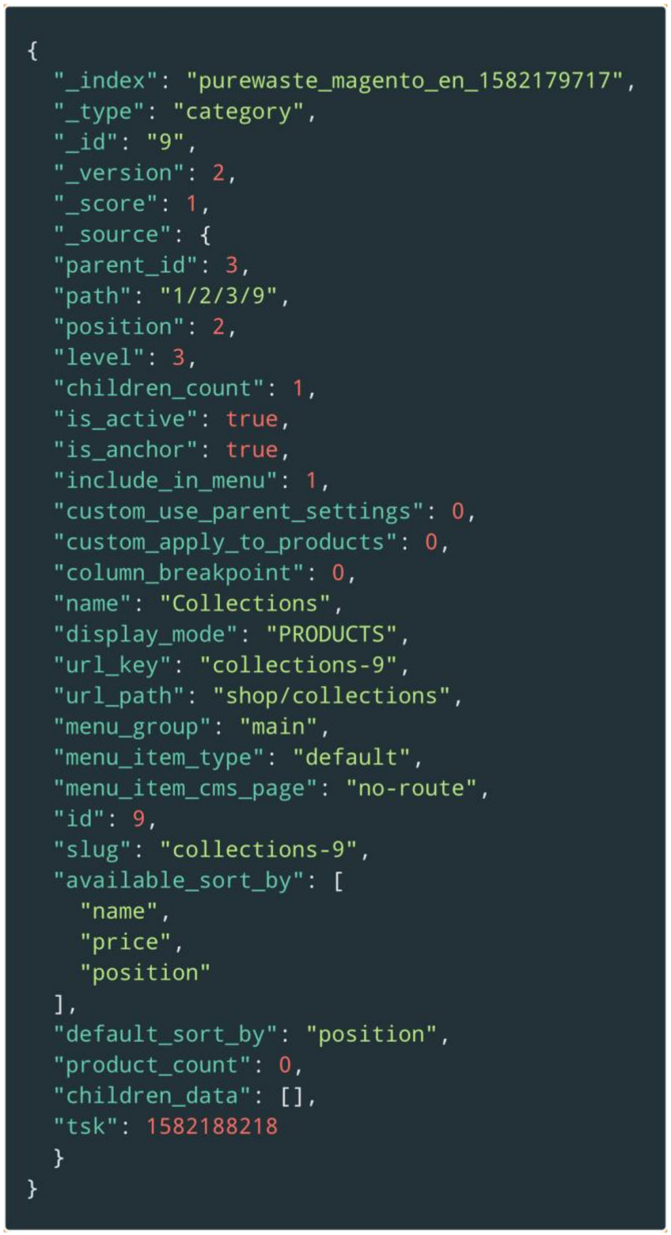


FIGURE 16. Design for Horizontal Menu from Implementation.

The implementation had the following challenges that were needed to resolve:

1. Getting category data into the component to show the menu
2. Deciding whether to render the component conditionally or hiding it depending on the breakpoint
3. Passing around an event from one component to another component
4. Keeping the component loosely coupled to make it reusable

In order to explain the origin of the category data that was needed for the FE implementation as per the first challenge mentioned above, it was needed to dive a bit into the Magento and Elastic data store side. In the Magento side, a module named *Vaimo Menu* is used to have a better customization control on Category and Menu provided by default Magento. Initially, it seemed that it would need API or Adapter level modifications from the VSF-API side to make it compatible with Vaimo Menu. But the magento2-vsbridge-indexer module that was installed within Magento to index magneto data to ES was dynamic enough to index custom category data into the ES store without further modifications. So, after indexing was done by the magento2-vsbridge-indexer module, the data given in figure 17 was already available for VSF-API to fetch and provide to VSF:



```
{
  "_index": "purewaste_magento_en_1582179717",
  "_type": "category",
  "_id": "9",
  "_version": 2,
  "_score": 1,
  "_source": {
  "parent_id": 3,
  "path": "1/2/3/9",
  "position": 2,
  "level": 3,
  "children_count": 1,
  "is_active": true,
  "is_anchor": true,
  "include_in_menu": 1,
  "custom_use_parent_settings": 0,
  "custom_apply_to_products": 0,
  "column_breakpoint": 0,
  "name": "Collections",
  "display_mode": "PRODUCTS",
  "url_key": "collections-9",
  "url_path": "shop/collections",
  "menu_group": "main",
  "menu_item_type": "default",
  "menu_item_cms_page": "no-route",
  "id": 9,
  "slug": "collections-9",
  "available_sort_by": [
    "name",
    "price",
    "position"
  ],
  "default_sort_by": "position",
  "product_count": 0,
  "children_data": [],
  "tsk": 1582188218
  }
}
```

FIGURE 17. Example category data in ES for VSF-API to fetch and provide to VSF.

Since the data from Magento was already in ES, an inquiry at the VSF-API level was needed to check the content of the data provided to VSF before proceeding to implementation. By default, VSF-API extracts certain fields of a given entity from the Elastic data store and can ship them to the front-end. The fields of an entity could be included or excluded from the VSF-API configuration (`config/default.json`). For the category entity, the fields shown below are included by default and could be fetched as the API response:

```
 "category": {
   "includeFields": [ "children_data", "id", "children_count", "sku", "name", "is_active", "parent_id", "level",
"url_key", "menu_color", "image"]
 },
```

As the origin of data coming to the VSF front-end was understood, the actual front-end implementation process was started. From the VSF part, the default category module found at `core/modules/catalog/` has all the business and data fetching logic already. Since the application uses a state management pattern, the data fetching is done with an Action which is found in `core/modules/catalog/store/category/actions.ts`. The action is named as `list` and it would be needed inside the Horizontal Menu components to trigger the fetching of category data. Also, inside the getter file, `core/modules/catalog/store/category/getter.ts`, there is a function called `getCategories` that would be used in the component to fetch data.

The first step taken towards the implementation consists of planning and sketching the composition of the components, as the plan for this implementation is to make it also compliant with *Atomic Design* principles described in *Design System Management* (section 3.1.10). The composition shown in figure 18 and figure 19 was planned in order create the horizontal menu functionality.

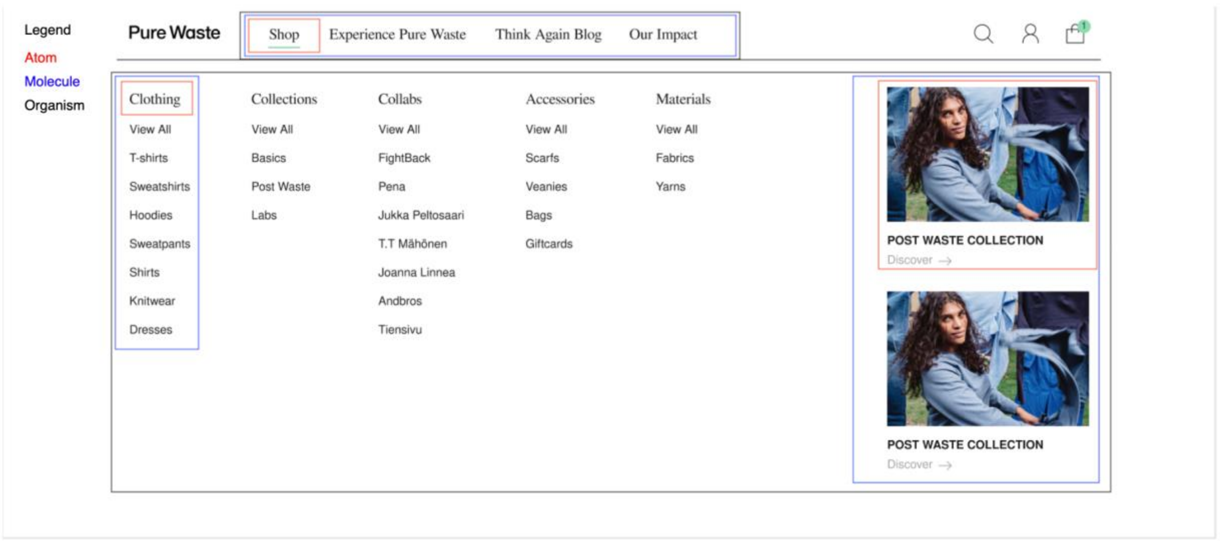


FIGURE 18. Breaking down the design according to Atomic Design principle.

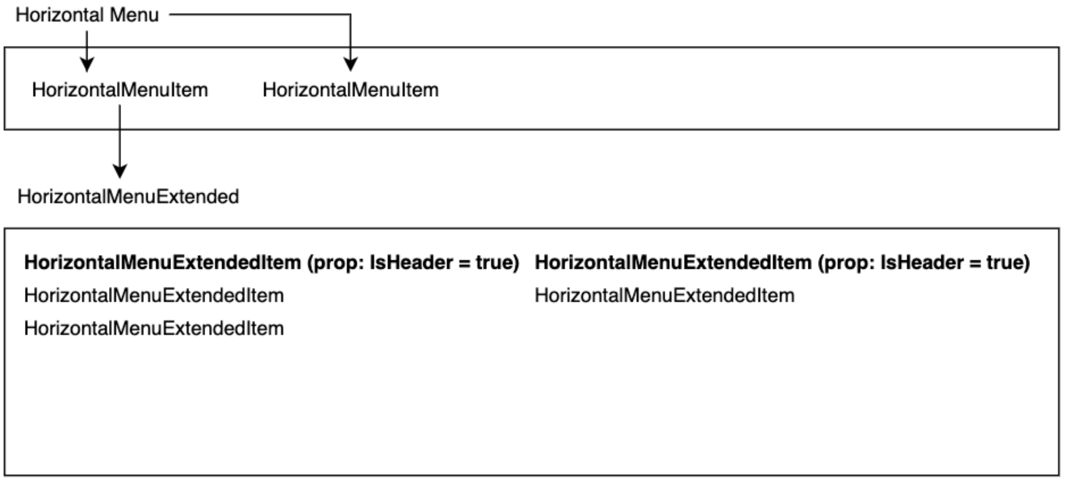

Horizontal Menu
HorizontalMenuItem
HorizontalMenuItem
HorizontalMenuExtended
HorizontalMenuExtendedItem (prop: IsHeader = true)
HorizontalMenuExtendedItem
HorizontalMenuExtendedItem
HorizontalMenuExtendedItem (prop: IsHeader = true)
HorizontalMenuExtendedItem

FIGURE 19. Planning the composition of components for Horizontal Menu

Since Vue and other modern frameworks support conditional rendering on the client-side, the second challenge mentioned in the beginning has been solved by conditionally rendering a different Menu component depending on the breakpoint instead of hiding the components with CSS. A third-party module, *vue-breakpoints,* has been used to perform conditional rendering depending on breakpoints. The wrapper component for the whole menu is the `HorizontalMenu.vue` component that is needed to be called from the `Header.vue` components found in `source/themes/{themeName}/components/core/blocks/Header.vue.`
A HorizontalMenu component was added there inside the `show-at` component provided by *vue-breakpoints* to render the component conditionally:

```
<show-at breakpoint="large">
  <HorizontalMenu />
</show-at>
```

Furthermore, the hamburger icon responsible for opening the sidebar menu was needed to be hidden from the desktop view and only shown on mobile and tab views. So, it was also wrapped inside the `show-at` component:

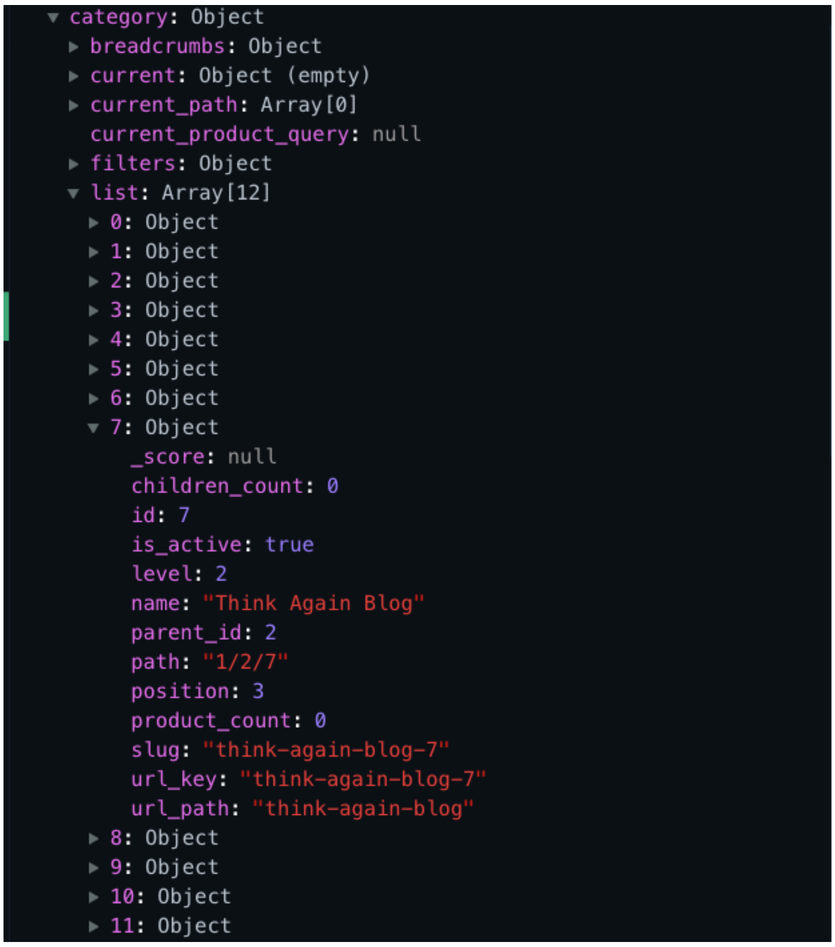
```
<show-at breakpoint="mediumAndBelow">
  <hamburger-icon class="p15 icon pointer" />
```

```
</show-at>
```

At this point, the HorizontalMenu component was created and rendered. The HorizontalMenu and subsequent components in the Horizontal Menu module has a template, logic, and style in the same file as a *Single File Component* similar to most other components in VSF. A Single File Component style composition may seem opposite to the idea of *SoC*, but it is a well-established pattern used within a large scale of Vue based applications where an inherently coupled template, logic and styles are collocated to make components more cohesive and maintainable.[102] The following logic resolves the first challenge of getting data to the component:

```
export default {
  name: 'HorizontalMenu',
  components: {
    HorizontalMenuItem
  },
  data () {
    return {
      baseLevel: 2
    }
  },
  computed: {
    ...mapGetters('category', ['getCategories']),
    getBaseLevelCategories () {
      return this.getCategories.filter((category) => {
        return category.level === this.baseLevel
      })
    }
  },
  methods: {
    ...mapActions('category', ['list'])
  },
  async created () {
    await this.list({level: this.baseLevel})
  }
}
```

So, as shown in the snippet above, when the component was created, it executed the `created` life cycle hook provided by Vue, within the created hooks the `list` function was called with `baseLevel` provided as a parameter. In this case, the `baseLevel` was 2 and declared within the Vue data function. The baseLevel could vary depending on the application. The `list` function does not inherently belong to this component, but it is originated from the category action mentioned earlier. Within Vue methods, the list action was mapped as a function of the component with the `mapActions` function provided by Vuex. The *Async Await* feature from JavaScript was used since Vuex Actions often contain asynchronous operations and the function was needed to be executed upon creation before moving forward with other logic within the component.[103] After the `list` function was executed, the data necessary for rendering the base level of Menu (level 2 in this case) should already be in the state which could be further ensured by checking in the Vuex tab in the Vue Devtools extension as shown in figure 20.



```
▼ category: Object
  ▶ breadcrumbs: Object
  ▶ current: Object (empty)
  ▶ current_path: Array[0]
    current_product_query: null
  ▶ filters: Object
  ▼ list: Array[12]
    ▶ 0: Object
    ▶ 1: Object
    ▶ 2: Object
    ▶ 3: Object
    ▶ 4: Object
    ▶ 5: Object
    ▶ 6: Object
    ▼ 7: Object
        _score: null
        children_count: 0
        id: 7
        is_active: true
        level: 2
        name: "Think Again Blog"
        parent_id: 2
        path: "1/2/7"
        position: 3
        product_count: 0
        slug: "think-again-blog-7"
        url_key: "think-again-blog-7"
        url_path: "think-again-blog"
    ▶ 8: Object
    ▶ 9: Object
    ▶ 10: Object
    ▶ 11: Object
```

FIGURE 20. Category list is loaded after list action is called (from Vue Devtools)

After the data was successfully fetched and loaded in the Vuex state, the component used the `getCategories` function that was mapped with the `mapGetters` function imported from the Vuex library as the final step of getting the data to the component. The same pattern of data fetching was also used when it was needed to fetch category level 2 and 3 data for rendering the HorizontalMenuExtended component.

The third challenge in developing the component was passing around events from one component to another. As per the implementation, `HorizontalMenuExtendedItem` was the component for rendering each level 2 and level 3 category menu. `HorizontalMenuExtended` was the wrapper responsible for rendering all `HorizontalMenuExtendedItem` through a loop. The challenge appeared during the implementation process of the following feature – "*the extended horizontal menu is hidden when any horizontal menu item is clicked and redirected to the designated link".* It is possible to use `v-on` directive provided by Vue for executing JavaScript when a DOM event is triggered – in this case listening to the `click` event and executing logic accordingly seemed like a solution:[104]

```
<div @click="menuLinkClicked"></div>
```

The challenge, however, was executing a function in the parent component when a child component triggers an event. Since, Vue also utilizes the *One-way Data Flow* pattern, which restricts the passing of data from a children to parent component to prevent accidental mutation, the only way to implement the feature was to pass an event.[105] The challenge could be mitigated by using an *Event bus*.

The Event bus is used in Vue applications when a component needs to broadcast an event to the rest of the application, and any other component within the application can subscribe and listen to it.[106] In VSF, a global event bus was already created which could be accessed by components as `this.$bus`. Thus, it was not needed to create a new one. The event bus has a property named `$emit`, that is used for emitting an event to other components, while the `$on` property is used for listening to a particular event. The following code snippets are used for passing an event from the child component (`HorizontalMenuExtendedItem`) to its parent (`HorizontalMenuExtended`):

```
methods: {
  menuLinkClicked () {
    this.$bus.$emit('menu-link-clicked')
  }
}
```

```
async created () {
  this.$bus.$on('menu-link-clicked', this.onLinkClick)
}
```

It was needed to make the components loosely coupled from the rest of the application to make the implementation reusable. The components in the horizontal menu module had that characteristic by design. All the template, logic, and style are self-contained inside the components. Also, the data needed to render the component is stored in the Vuex global store. Thus, the component used as the entry point to the module does not need any data to be passed down as prop.

4.4 Reusing Existing Modules (Fayjus)

The implementation needed the integration of the *Klarna* payment service during the checkout step of the application. Since there was no stable VSF module for Klarna available as of the time of the implementation, it became a blocking issue to continue developing the checkout step in VSF. As a solution to this problem, an existing VSF module named *vsf-external-checkout* was used to execute the checkout step externally within Magento 2. When a user tries to enter the VSF checkout, the module synchronizes the cart and user data with the Magento 2 instance. Later, the module redirects the user to Magento CMS, as shown in figure 21.

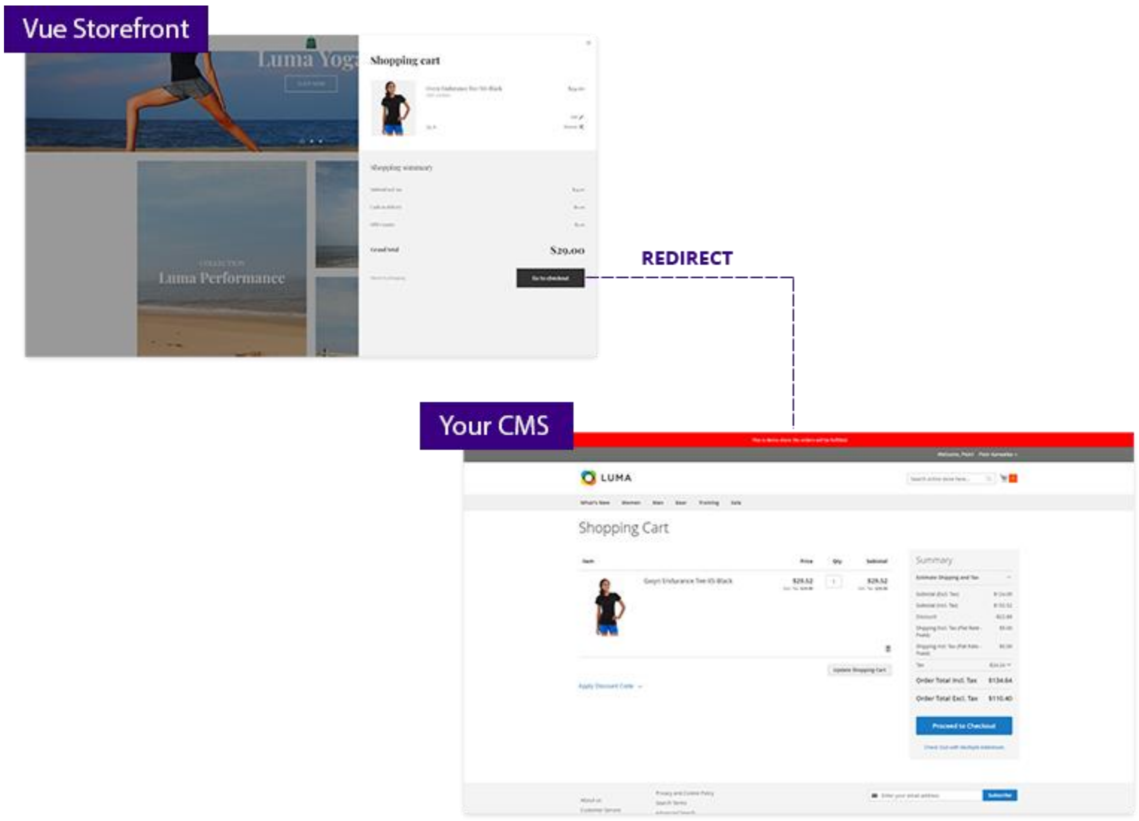


FIGURE 21. VSF external checkout module functionality. [107]

Two different modules were needed to be installed to achieve the external checkout functionality- one from the Magento side and the other from the VSF side. The installation process of the module installed from the VSF side would be documented here to demonstrate the reusability of an existing VSF module. It was not possible to install the *vsf-external-checkout* module through the yarn package manager since the feature was not supported.[107] As part of the manual installation process, the latest release (3.0.0) was downloaded and copied to the `src/module` folder. Next, the configuration was updated in `default.json` and `local.json` from the `config` folder. Due to a bug within the module, it was needed to provide CMS addresses for individual stores explicitly despite the addresses being the same.[108] Also, it was checked that the cart synchronization is enabled:

```
"externalCheckout": {
  "cmsUrl": "https://magento.purewaste-dev.north1.vaimo.net/",
  "stores": {
    "en": {
      "cmsUrl": "https://magento.purewaste-dev.north1.vaimo.net/"
    },
    "fi": {
```

```
    "cmsUrl": "https://magento.purewaste-dev.north1.vaimo.net/"
   }
  }
}
```

Within the `src/module` folder, there is a file named `client.ts` that is responsible for registering all the modules used within the implementation. The external checkout module was also registered there:

```
import { ExternalCheckout } from './external-checkout'

export function registerClientModules () {
  // other extensions
  registerModule(ExternalCheckout)
}
```

The external checkout should have been working at this point, as described in the official documentation of the module.[107] But since the implementation had `multistore` enabled and it was not possible to assign `defaultStorecode` due to a VSF bug,[109] the module was not working in the default store view. However, this issue was resolved by adding a fallback mechanism in the `beforeEach.ts` file. `beforeEach` is a hook provided by vue-router that enables executing the logic before each routing requests are handled. The logic is updated, so if the current store code is missing vue-router redirects the checkout request to the default CMS address:

```
if (storeCode in stores && to.name === storeCode + '-checkout') {
  window.location.replace(stores[storeCode].cmsUrl + '/vue/cart/sync/token/'    + userToken + '/cart/' +
cartToken)
} else if (storeCode in stores && to.name === 'checkout' && stores[storeCode].cmsUrl !== undefined) {
  window.location.replace(stores[storeCode].cmsUrl + '/vue/cart/sync/token/' + userToken + '/cart/' +
cartToken)
} else if (to.name === 'checkout') {
  window.location.replace(cmsUrl + '/vue/cart/sync/token/' + userToken + '/cart/' + cartToken)
} else {
  next()
}
```

Despite manual installation and the issues found within the module, reusing existing modules such as vsf-*external-checkout,* could still be considered a well-documented and easy process since it requires only a couple of steps to integrate a module into the project. To get started with an external module, getting the source code and then registering the module seems to be the only mandatory step. Sometimes additional module-specific configuration within VSF or Magento 2 might also be needed.

INSTALLING NOSTO RECOMMENDATIONS MODULE (DAIR)

Adding Nosto recommendations is another example of reusing existing modules in Vue Storefront.[110] Nosto tagging is a third-party Magento 2 module that allows displaying personalized product recommendations for users.[111] The product department of Vaimo has developed a generic module for implementing Nosto recommendations in VSF – *vsf-nosto-tagging*.

After setting up Nosto from the Magento side, the recommendations have to be enabled from the Nosto administration panel and the domain of VSF application has to be added to the list of "*Other hosts used by the website (one per line)"* in the advanced settings section. The following configuration has to be added to the `config/local.json` in the VSF:

```
"nosto": {
  "id": "test-id",
  "enabled": true,
  "debugEnabled": true
}
```

The module also supports a multistore setup:

```
"nosto": {
  "en":{
    "enabled": true,
    "id": "test-id"
  },
  "fi":{
    "enabled": true,
    "id": "test-id"
  },
  "debugEnabled": true
}
```

The `debugEnabled` option will allow logging Nosto API request results to the console. To add the recommendation block to the Vue component or page, the required components need to be imported and registered:

```
import NostoWrapper from 'src/modules/nosto-tagging-module/components/NostoWrapper.vue'
import VsProductListSlot from 'src/modules/nosto-tagging-
module/components/VsQueryList/VsProductListSlot'
```

Once that is done, a Nosto recommendation can be added from the template:

```
<nosto-wrapper
  :page="'front'"
  :placement-ids="['frontpage-nosto-2']"
  :key="'front1'"
  :slot-element="slotElement"
/>
```

`:placement-ids` is the ID of the recommendation template. The example code above will render the “Most Popular Right Now” block, which is synchronised to the products from the Magento 2 instance (figure 22). The tagging happens even when the user visits product pages from the VSF.

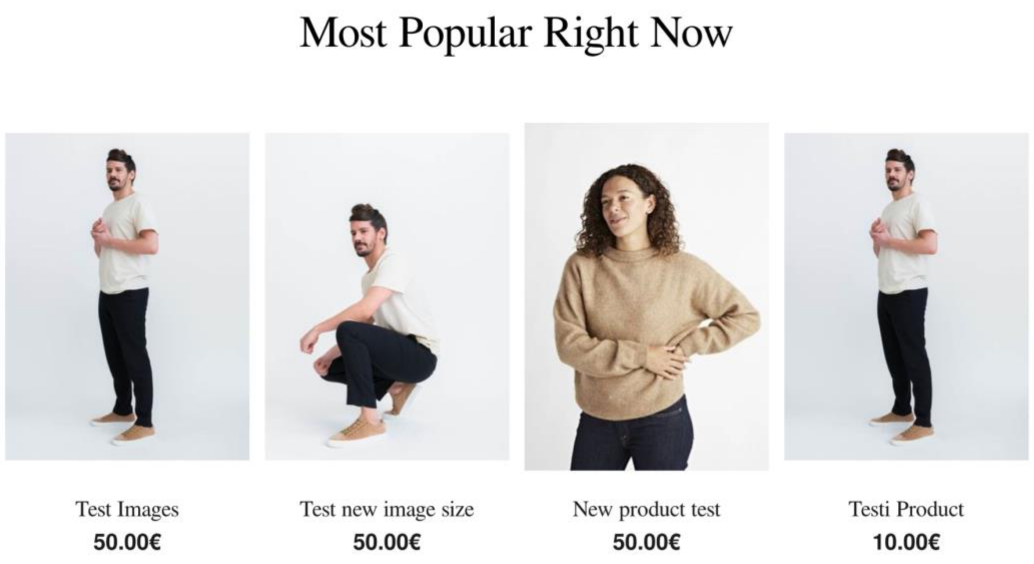


FIGURE 22. Nosto recommendations block in VSF.

4.5 Publishing a Module (Fayjus)

It is possible to create and publish a VSF module both manually and with CLI. The following steps are recommended by the official VSF documentation to create a module manually,

1. All the modules created and used within the project should be under the `src/modules` folder. Thus, the *vsf-external-checkout* module is placed under that directory. The modules created within Vaimo, such as *nosto-tagging-module* could further be organized by putting them in `src/modules/Vaimo`.
2. The next step is to create a file named `index.ts` within the directory. The module should be exported from this file implementing the `StorefrontModule` interface provided by Vaimo. As it could be seen from the `core/lib/modules.ts` file, the interface has the following structure:

```
export type StorefrontModule = (
  options: {
    app: Vue,
    store: Store<RootState>,
    router: VueRouter,
    moduleConfig: any,
    appConfig: any
  }
) => void
```

The purpose of using this interface is to access the store, router, configurations from the newly created module. The content from the `index.ts` file from *nosto-tagging-module* is shown below as a reference,

```
import { isServer } from '@vue-storefront/core/helpers';
import { StorefrontModule } from '@vue-storefront/core/lib/modules';
import { afterRegistration } from './hooks/afterRegistration';
import { module } from './store';
export const KEY = 'nosto-tagging';

export const NostoTagging: StorefrontModule = function ({store,  router, appConfig}) {
  if (!isServer) {
    store.registerModule(KEY, module);
    afterRegistration(appConfig, store, isServer)
  }
};
```

3. The next step is to register the module in the `client.ts` file as it is shown in the *vsf-external-checkout* module installation process.
4. The module could be distributed as a package by introducing a `package.json` file inside the directory. For Vaimo specific modules, a naming convention has been set by the Product Department. As per the convention VSF module should be prefixed with *vue-* (example: `vaimo/vsf-nosto-tagging`).

4.6 CI/CD pipeline with Argo and Kubernetes (Dair)

One of the benefits of headless architecture is the ability to build and deploy storefront, API and the back-end separately from each other. As each service is an independent stand-alone application, it could be developed, maintained and deployed as such.[13] This allows for independent (external) teams, continuous deployment, hassle-free extensibility and unlimited scalability per service. Front-end developers could only build the PWA to deploy their changes, as opposed to building and reinstalling the whole Magento platform. This change results in faster builds, and better continuous deployment, improved SoC. Hosting can be tweaked and scaled per service.[13]

In Vaimo the CI/CD is done with Argo – a container-native workflow engine for Kubernetes (K8S).[112] VSF and VSF-API require Docker by default to run ElasticSearch, Redis and Kibana.[113] Argo CD is used for handling deployments in the GitOps fashion, meaning that Kubernetes cluster "mirrors everything from the git repositories" of the projects.[114]

4.6.1 Build and deploy procedures (Dair)

To initiate a build of VSF with `argo`, the following command needs to be run:

```
argo submit workflow.yaml
```

This will build the master branch by default. It is possible to provide other branches or git revisions as arguments, for example "dev" branch:

```
argo submit -p revision=dev workflow.yaml
```

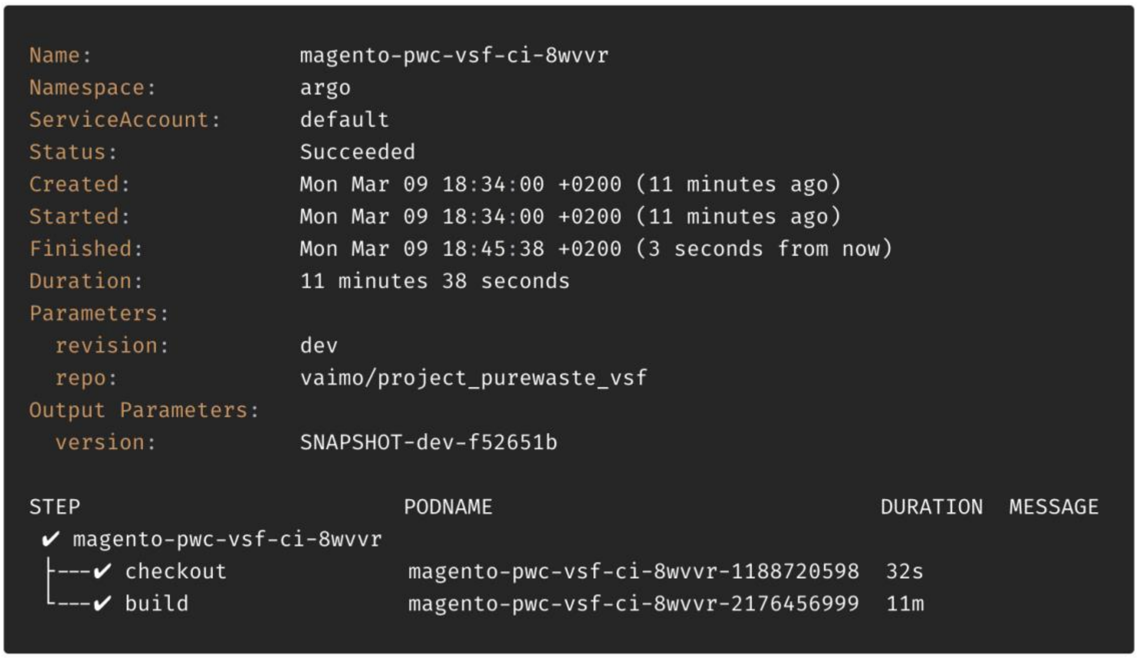
Argo will build the master branch of the VSF repository by default, which can be seen in the `workflow.yaml` – revision argument:

```
spec:
  entrypoint: main
  ttlSecondsAfterFinished: 7200

  arguments:
    parameters:
    - name: repo
      value: vaimo/project_purewaste_vsf
    - name: revision
      value: master
```

Build is done based on the templates defined in `workflow.yaml` (shortened example):

```yaml
spec:
  templates:
  - name: build
    container:
      image: eu.gcr.io/vaimogcr/magento-base-php70:latest
      command: ["/bin/sh", "-c"]
      resources:
        requests:
          cpu: 900m
          memory: 1300Mi
      volumeMounts:
      - name: composer-cache
        mountPath: /var/www/.composer/cache
      - name: npm-cache
        mountPath: /var/www/.npm
```

Once the build has been started, developer can *watch* the progress with the `argo watch` command. When the build is finished, the output parameters, containing the new revision tag for the K8S configuration, will be generated (figure 23).



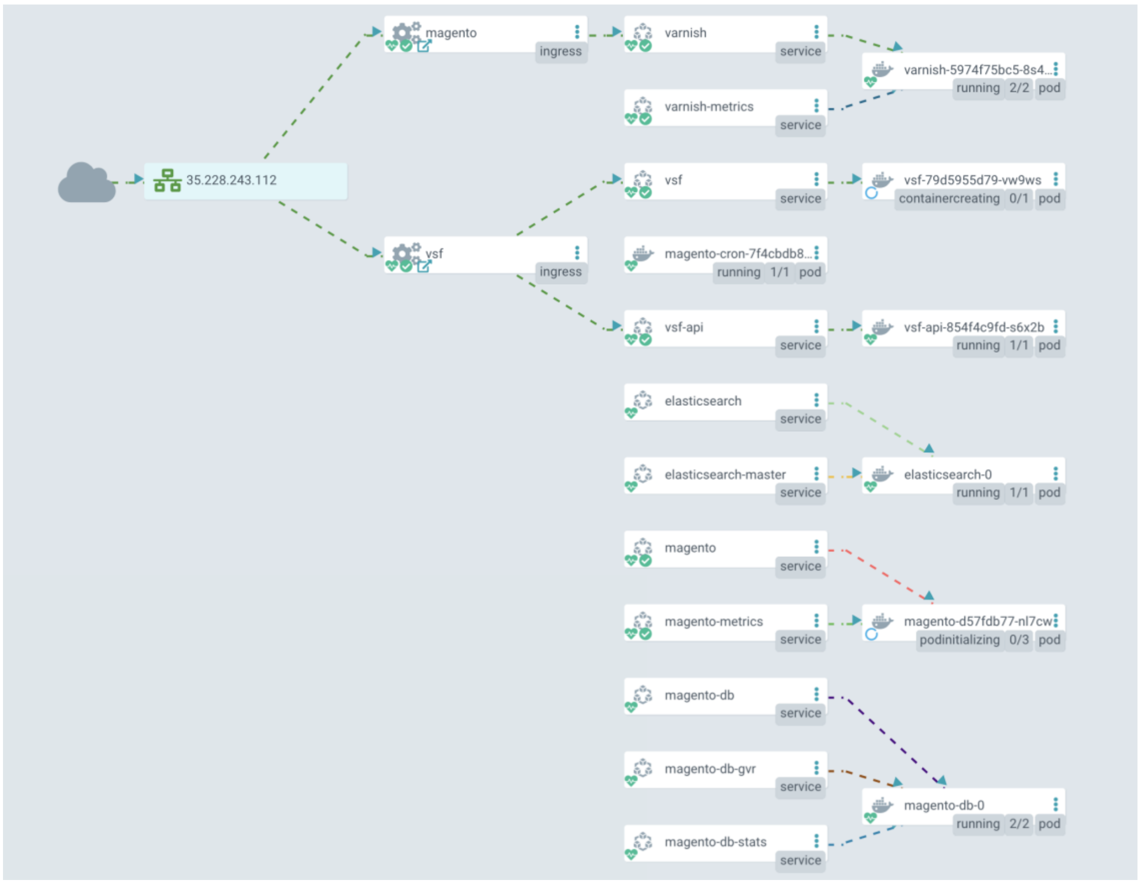
```
Name:                  magento-pwc-vsf-ci-8wvvr
Namespace:             argo
ServiceAccount:        default
Status:                Succeeded
Created:               Mon Mar 09 18:34:00 +0200 (11 minutes ago)
Started:               Mon Mar 09 18:34:00 +0200 (11 minutes ago)
Finished:              Mon Mar 09 18:45:38 +0200 (3 seconds from now)
Duration:              11 minutes 38 seconds
Parameters:
  revision:            dev
  repo:                vaimo/project_purewaste_vsf
Output Parameters:
  version:             SNAPSHOT-dev-f52651b

STEP                             PODNAME                              DURATION  MESSAGE
 ✔ magento-pwc-vsf-ci-8wvvr
 ├---✔ checkout                  magento-pwc-vsf-ci-8wvvr-1188720598  32s
 └---✔ build                     magento-pwc-vsf-ci-8wvvr-2176456999  11m
```

FIGURE 23. Finished build.

`kustomization.yaml`, the configuration file for K8S objects, should be modified for the Argo CD to spot the latest changes. `kustomization.yaml` contains the information about images used within the project and `newTag` represents the current revision of the application. It can be either branch, revision or tag:

```yaml
images:
  - name: magento
    newName: eu.gcr.io/vaimogcr/purewaste
    newTag: SNAPSHOT-feature_PUREW-50_water_saved_to_email-8d6ef26
  - name: vsf-api
    newName: eu.gcr.io/vaimogcr/purewaste-vsf-api
    newTag: 0.0.10
  - name: vsf
    newName: eu.gcr.io/vaimogcr/purewaste-vsf
    newTag: SNAPSHOT-dev-8fdf4d7
```

When the `newTag` is changed, it has to be also synchronized with Argo CD. When everything is up-to-date, every service will be displayed as *synced* from the internal Argo CD web interface. On figure 24, the network graph for the application is displayed. It can be seen that all pods are synchronised, and `vsf` pod is in the `containercreating` state, whereas the `magento` pod is in the `podinitializing` state.



35.228.243.112
magento ingress
vsf ingress
varnish service
varnish-metrics service
vsf service
magento-cron-7f4cbdb8... running 1/1 pod
vsf-api service
elasticsearch service
elasticsearch-master service
magento service
magento-metrics service
magento-db service
magento-db-gvr service
magento-db-stats service
varnish-5974f75bc5-8s4... running 2/2 pod
vsf-79d5955d79-vw9ws containercreating 0/1 pod
vsf-api-854f4c9fd-s6x2b running 1/1 pod
elasticsearch-0 running 1/1 pod
magento-d57fdb77-nl7cw podinitializing 0/3 pod
magento-db-0 running 2/2 pod

FIGURE 24. Argo CD web interface. Network graph of the project.

4.6.2 Improvement of the CI/CD speed with headless architecture (Dair)

Due to the decoupled front-end and lightweight PWA, the time to build the latest front-end changes has been decreased to the average of *9 minutes* with VSF, compared to the average 14 minutes of building the Magento 2 platform. The slower the building time, the more it is likely that the productivity of the development teams, and therefore the productivity of the company suffers.[115] The older projects with a greater number of customizations, integrations and installed modules may sometime take half an hour to build. Headless architecture helps to prevent the increase of the build time, since the logic is split between three independent applications.

4.7 Automated testing (Dair)

Vue Storefront makes use of Jest for unit testing and Cypress for the E2E testing.

A unit is the smallest testable part of the software. The goal of unit testing is to validate that each unit of the software performs as designed. Unit tests usually have one or multiple inputs and a single output. In object-oriented programming, the smallest unit is a method of the class. Unit testing frameworks are used to assist in unit testing.[116]

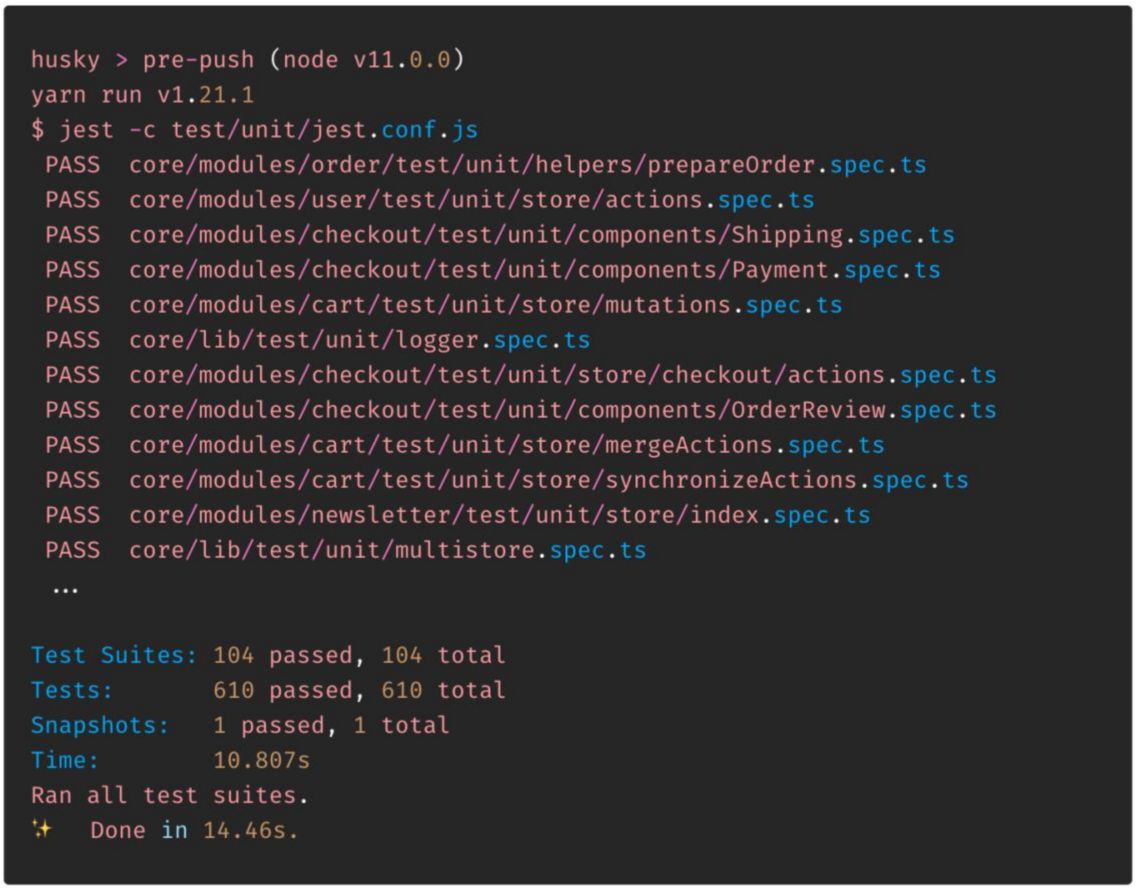
E2E testing ensures that the flow of an application right from the start to the finish is behaving as expected. It helps identifying system dependencies and consistency of data integrity between various system components and systems. The entire application is tested for critical functionalities, such as communicating with the other systems, interfaces, database, network, and other applications.[117]

4.7.1 Unit testing with Jest (Dair)

By default, unit tests are triggered by a pre-commit hook. This hook is configured with Husky, a tool for integrating various actions into git hooks.[118] There are currently only two hooks defined – “pre-commit”, which runs an ESLint command on the files that are staged for the commit, and “pre-push”, which runs all unit tests inside the core folder:

```
const tasks = arr => arr.join(' && ')


module.exports = {
  'hooks': {
    'pre-commit': tasks([
      'lint-staged'
    ]),
    'pre-push': tasks([
      'yarn test:unit'
    ])
  }
}
```

There is a total of 104 test suites and 610 tests for the VSF core files. From the results in figure 25, it can be seen that it takes around 14 seconds to run all of the tests.



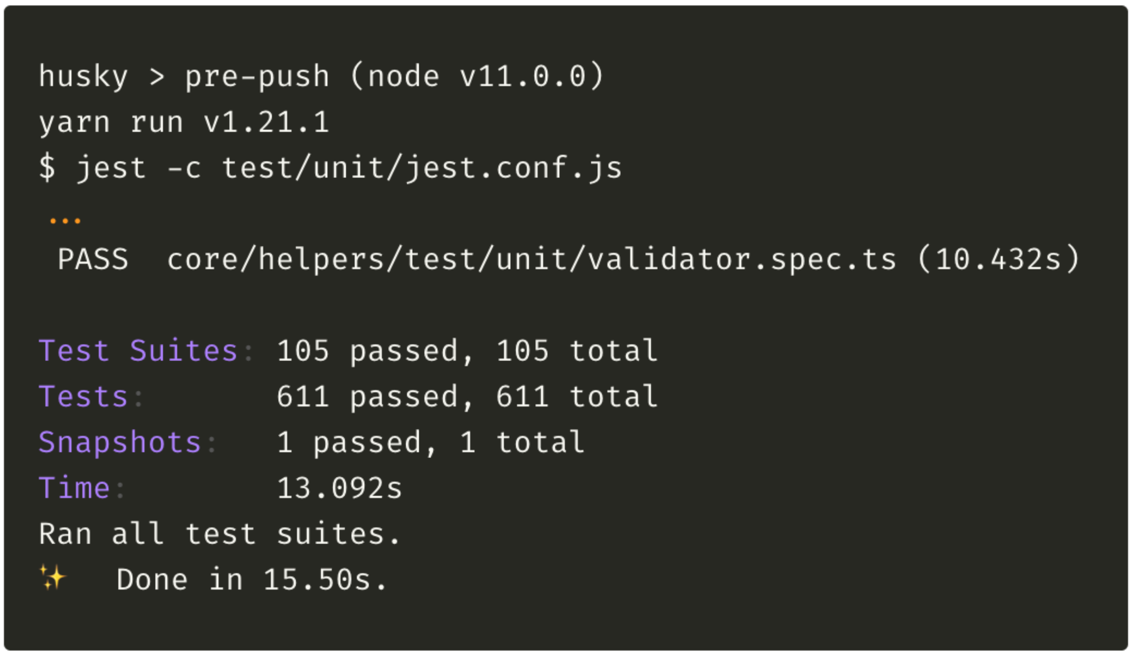
```
husky > pre-push (node v11.0.0)
yarn run v1.21.1
$ jest -c test/unit/jest.conf.js
 PASS  core/modules/order/test/unit/helpers/prepareOrder.spec.ts
 PASS  core/modules/user/test/unit/store/actions.spec.ts
 PASS  core/modules/checkout/test/unit/components/Shipping.spec.ts
 PASS  core/modules/checkout/test/unit/components/Payment.spec.ts
 PASS  core/modules/cart/test/unit/store/mutations.spec.ts
 PASS  core/lib/test/unit/logger.spec.ts
 PASS  core/modules/checkout/test/unit/store/checkout/actions.spec.ts
 PASS  core/modules/checkout/test/unit/components/OrderReview.spec.ts
 PASS  core/modules/cart/test/unit/store/mergeActions.spec.ts
 PASS  core/modules/cart/test/unit/store/synchronizeActions.spec.ts
 PASS  core/modules/newsletter/test/unit/store/index.spec.ts
 PASS  core/lib/test/unit/multistore.spec.ts
  ...

Test Suites: 104 passed, 104 total
Tests:       610 passed, 610 total
Snapshots:   1 passed, 1 total
Time:        10.807s
Ran all test suites.
✨  Done in 14.46s.
```

FIGURE 25. Running unit tests with Jest. Output.

It is possible to either modify existing tests, add new tests for existing modules and components or create tests for new modules and components. Below is the example of a custom test for the VSF core helper – alphanumeric validator:

```
import { unicodeAlpha, unicodeAlphaNum } from '@vue-storefront/core/helpers/validators'

jest.clearAllMocks()

describe('Alphanumeric validator', () => {
  it('Test alphanumeric validators', () => {
    expect(unicodeAlpha('asdf')).toBe(true)
    expect(unicodeAlpha('12')).toBe(false)
    expect(unicodeAlpha('asdf123')).toBe(true)
    expect(unicodeAlphaNum('123')).toBe(true)
    expect(unicodeAlphaNum('asdf123')).toBe(true)
  })
})
```

The test simply evaluates the outputs of two validation functions. After adding this test, the time to run all unit tests has increased to approximately 1 second with the last test taking more than 10 seconds (figure 26). However, the time has moved back to normal on the second run.



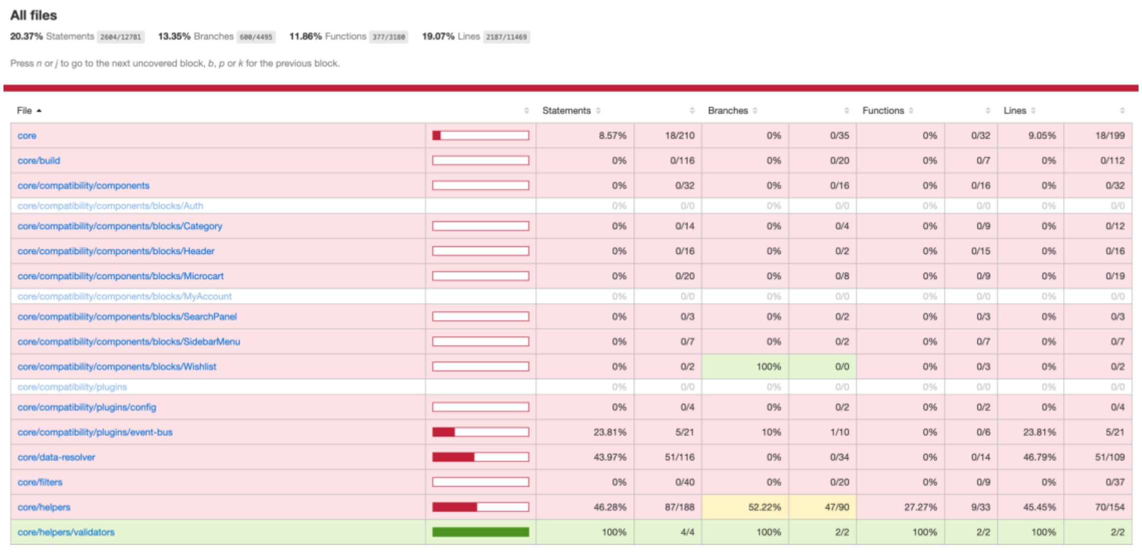
```
husky > pre-push (node v11.0.0)
yarn run v1.21.1
$ jest -c test/unit/jest.conf.js
 ...
 PASS  core/helpers/test/unit/validator.spec.ts (10.432s)

Test Suites: 105 passed, 105 total
Tests:       611 passed, 611 total
Snapshots:   1 passed, 1 total
Time:        13.092s
Ran all test suites.
✨  Done in 15.50s.
```

FIGURE 26. After adding new test.

Jest can also create test coverage reports. Test coverage helps developers to see which parts of the code are not tested. The data is divided into *statement* (has each statement in the program been executed), *function* (has each function or subroutine in the program been called), *branch* (has each decision path, e.g., if-else clause, been tested) and *line* (has each line in the source file been executed) coverage.[119] The report takes into account every JavaScript file under the directories defined in the `test/unit/jest.conf.js`:

```
testMatch: [
  '<rootDir>/src/modules/**/test/unit/**/*.spec.(js|ts)',
  `<rootDir>/src/themes/**/*.spec.(js|ts)`,
  '<rootDir>/core/**/test/unit/**/*.spec.(js|ts)'
]
```

The coverage command will also generate a visual HTML report which can be found in the **coverage/lcov-report** folder (figure 27).[120] As seen from the report, default Vue Storefront unit tests cover 20% of statements, 13% of branches, 12% percent of functions and 19% of code lines from the core files. The unit test for validators, which was written manually, can be seen on the bottom of the list. It has a 100% coverage thanks to its simplicity.



All files

20.37% Statements 2604/12781 **13.35%** Branches 600/4495 **11.86%** Functions 377/3180 **19.07%** Lines 2187/11469

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File ▲	Statements		Branches		Functions		Lines	
core	8.57%	18/210	0%	0/35	0%	0/32	9.05%	18/199
core/build	0%	0/116	0%	0/20	0%	0/7	0%	0/112
core/compatibility/components	0%	0/32	0%	0/16	0%	0/16	0%	0/32
core/compatibility/components/blocks/Auth	0%	0/0	0%	0/0	0%	0/0	0%	0/0
core/compatibility/components/blocks/Category	0%	0/14	0%	0/4	0%	0/9	0%	0/12
core/compatibility/components/blocks/Header	0%	0/16	0%	0/2	0%	0/15	0%	0/16
core/compatibility/components/blocks/Microcart	0%	0/20	0%	0/8	0%	0/9	0%	0/19
core/compatibility/components/blocks/MyAccount	0%	0/0	0%	0/0	0%	0/0	0%	0/0
core/compatibility/components/blocks/SearchPanel	0%	0/3	0%	0/2	0%	0/3	0%	0/3
core/compatibility/components/blocks/SidebarMenu	0%	0/7	0%	0/2	0%	0/7	0%	0/7
core/compatibility/components/blocks/Wishlist	0%	0/2	100%	0/0	0%	0/3	0%	0/2
core/compatibility/plugins	0%	0/0	0%	0/0	0%	0/0	0%	0/0
core/compatibility/plugins/config	0%	0/4	0%	0/2	0%	0/2	0%	0/4
core/compatibility/plugins/event-bus	23.81%	5/21	10%	1/10	0%	0/6	23.81%	5/21
core/data-resolver	43.97%	51/116	0%	0/34	0%	0/14	46.79%	51/109
core/filters	0%	0/40	0%	0/20	0%	0/9	0%	0/37
core/helpers	46.28%	87/188	52.22%	47/90	27.27%	9/33	45.45%	70/154
core/helpers/validators	100%	4/4	100%	2/2	100%	2/2	100%	2/2

FIGURE 27. HTML report for test coverage.

Test coverage gives the overall idea about the test distribution in the project. A low coverage does not require writing unit tests for every component or helper function. For instance, developers may find it hard to write unit tests for presentational components with no or very little logic.[121] Instead, they might find creating snapshot tests more useful. In the snapshot tests the application renders a UI component, takes a snapshot, then compares it to a reference snapshot file stored alongside the test. If the two snapshots do not match, the test will fail, meaning that either the change was unexpected or the reference snapshot needs to be updated to the new version of the component.[122]

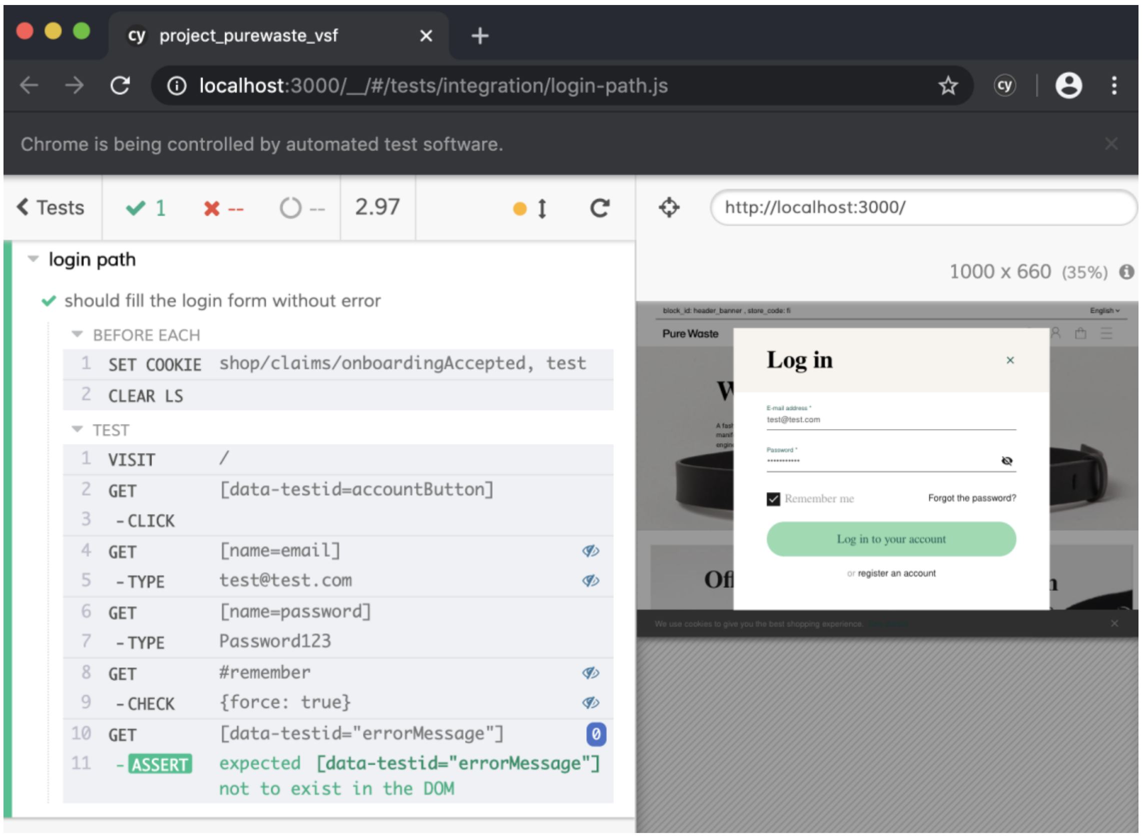
4.7.2 E2E testing with Cypress (Dair)

Cypress is a JavaScript-based framework that runs and evaluates end-to-end tests in the browser. It has features like waiting for the DOM tree generation, real-time reloads and debugging the snapshots taken at any step of the execution. It enables writing generic tests that are not affected by the technology of the web application and it integrates automated testing processes in the development cycles.[123] Cypress takes a snapshot of each step allowing developers to experience the system at different points of test execution.

VSF comes equipped with Cypress and a number of default integration tests for the demo VSF instance. Test scripts are simple JavaScript files with commands to the browser, and conditions for the expected result. Below is the example test script for Cypress:

```
describe('login path', () => {
  it('should fill the login form without error', () => {
    cy.visit('/')
    cy.get('[data-testid=accountButton]').click()
    cy.get('[name=email]').type('test@test.com')
    cy.get('[name=password]').type('Password123')
    cy.get('#remember').check({ force: true })
    cy.get('[data-testid="errorMessage"]').should('not.exist')
  })
})
```

When Cypress test is launched, a browser is opened displaying the test execution. The console window on the left side lists each step in the test and displays information about the commands executed. The actual view of the webpage on the right side of the window renders the actual UI of the website, which end users will see. The *login-path.js* test which was executed in the screenshot has taken 2.97 seconds to run, with every actions resulting in success (figure 28).


cy project_purewaste_vsf

localhost:3000/__/#/tests/integration/login-path.js

Chrome is being controlled by automated test software.

Tests ✓ 1 ✗ -- -- 2.97

http://localhost:3000/

1000 x 660 (35%)

login path

✓ should fill the login form without error

BEFORE EACH

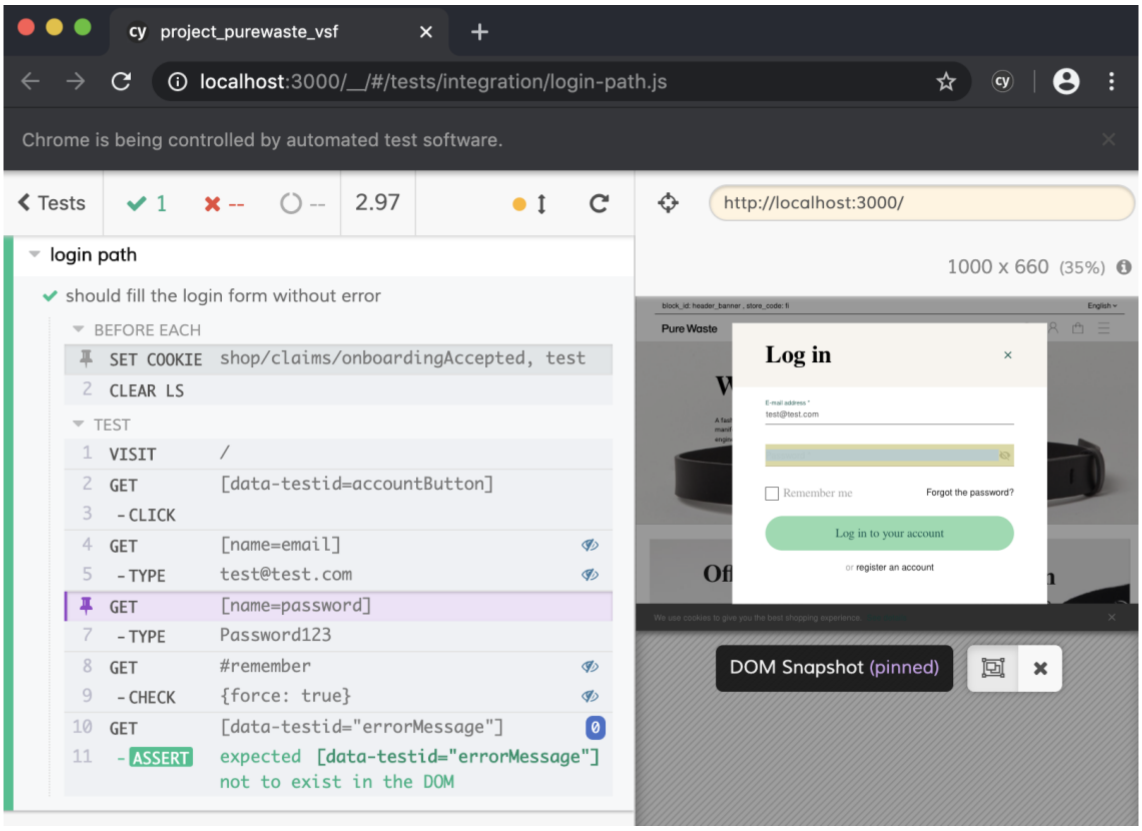
```
SET COOKIE  shop/claims/onboardingAccepted, test
CLEAR LS
```

TEST

```
VISIT       /
GET         [data-testid=accountButton]
 -CLICK
GET         [name=email]
 -TYPE      test@test.com
GET         [name=password]
 -TYPE      Password123
GET         #remember
 -CHECK     {force: true}
GET         [data-testid="errorMessage"]       0
 -ASSERT    expected [data-testid="errorMessage"]
            not to exist in the DOM
```

FIGURE 28. A successful Cypress E2E test

When hovering over the execution steps, the inline frame containing the webpage is updated to the corresponding snapshot, allowing the user to inspect the system at that moment with the familiar debugging tools such as Developer Tools (figure 29).[124] The icon with the crossed eye on the right of the execution log indicates that the element is not visible on the page, and the number indicates the number of elements with the corresponding selector.



```
login path
  ✓ should fill the login form without error
    BEFORE EACH
       SET COOKIE  shop/claims/onboardingAccepted, test
    2  CLEAR LS
    TEST
    1  VISIT       /
    2  GET         [data-testid=accountButton]
    3   -CLICK
    4  GET         [name=email]
    5   -TYPE      test@test.com
       GET         [name=password]
    7   -TYPE      Password123
    8  GET         #remember
    9   -CHECK     {force: true}
    10 GET         [data-testid="errorMessage"]                0
    11  -ASSERT    expected [data-testid="errorMessage"]
                   not to exist in the DOM
```

FIGURE 29. Cypress E2E test snapshot

Cypress is a developer-friendly and robust framework. It helps writing easy and fast E2E tests with a natural-language programming syntax and an interactive output. It can be easily integrated into existing projects and used on a website that is not built with JavaScript. It is easy to setup and install and it does not require an in-depth programming knowledge for QA engineers to write custom test cases.[125] Cypress, in combination with Jest, creates a durable testing foundation for web projects built with the modern web technology stack. It encourages developers to prioritize quality when designing and implementing the software units, it improves testing efficiency with thoroughness in testing and earlier detection of software defects. Automated tests can be reused due to their repetitive nature, which may result in improved security practices since they are usually independent from the test and live project databases.[126]

4.8 Design system maintenance (Dair)

E-commerce projects may face severe changes to the design during the implementation. If there is a need for a design system, then every modification or addition, at first, should be included to the style guide, so the developers can work on it. Component-based architecture enables applying changes only once to modify all the instances of the component in the code. The design system needs to adapt to feedback. It should be constantly iterated on and be able to evolve alongside the products it serves.[127] To create a design system that takes a root and becomes an essential part of the organization's workflow, an increased collaboration between UI/UX engineers and developers is required. Necessary tools and practices for supporting the design systems implementation have to be decided on and utilized.

Storybook creates an environment for centralizing the implementation of every UI component from the style guide individually or within a group (molecules, organisms). Components are maintainable since they are using the same code base for the actual storefront implementation. Components are developed without any back-end implementation and they can be divided into multiple groups forming e.g. atoms, molecules or organisms.

Storybook is served as a separate instance, separate from the actual application. Designers can access the Storybook to test and see the implemented UI components and how they function in different scenarios, whereas QA engineers can evaluate components with the help of installed add-on libraries and give feedback on component-level issues, such as accessibility, during the early stages of development.[127]

4.8.1 Creating a component library with Storybook (Dair)

Storybook has a support for Vue by default. It was installed to the theme of the project with yarn:

```
yarn add @storybook/vue --dev
```

Three add-ons were tested during the investigation of the Storybook: "Source" – to see the source code of the components, "Knobs" – to dynamically interact with component values, and "A11Y" – for the accessibility evaluation.[76] After installing the add-ons, they have to be manually added to `.storybook/main.js`, as follows:

```
module.exports = {
  stories: ['../../../**/*.stories.[tj]s'],
  addons: [
    '@storybook/addon-knobs/register',
    '@storybook/addon-a11y/register',
    '@storybook/addon-storysource'
  ]
}
```

Storybook can then be started:

```
yarn start-storybook
```

The list of components, also known as "stories", on the left panel currently does not have any logical structure in the given implementation (figure 30), but it can be organized to indicate atomic design groups and component types, similarly to the Storefront UI implementation.[128] The left panel is a browser of components with a multi-layered navigation for different component types and a search box. The right panel renders the components and has controls for zooming, color blindness emulation, responsive web tools, and many other, depending on the installed add-ons. On the figure 30 the *Tabs* component is rendered with a test data given below:

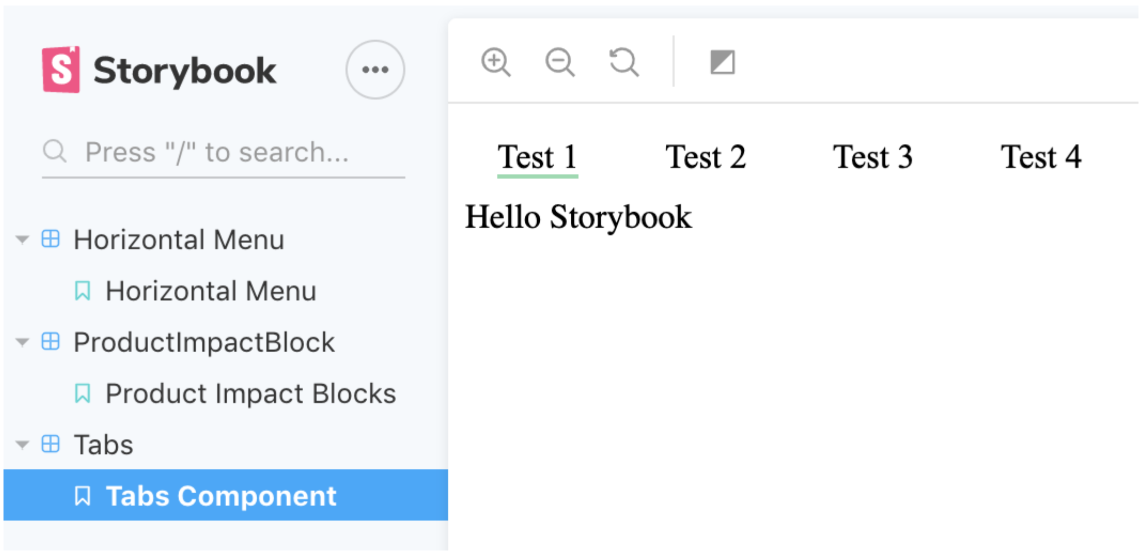
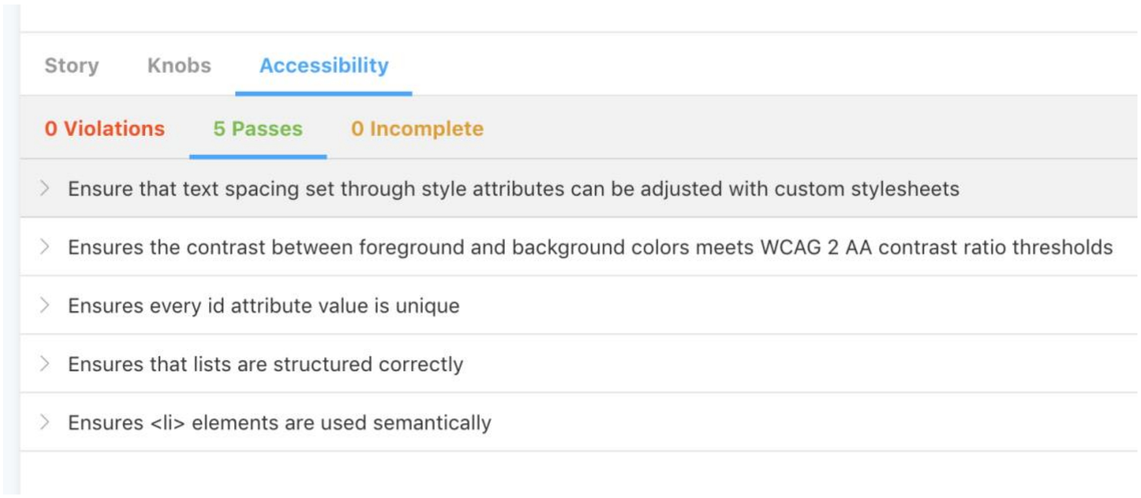


FIGURE 30. Storybook example

The control panel on the bottom of the window (figure 31) displays the tabs for the add-ons installed earlier – “Source”, “Knobs” and “A11Y”[76]. The accessibility add-on “A11Y”[76] gives information about the currently active component’s compliance with accessibility guidelines set by W3C’s Web Content Accessibility Guidelines.



Ensure that text spacing set through style attributes can be adjusted with custom stylesheets

Ensures the contrast between foreground and background colors meets WCAG 2 AA contrast ratio thresholds

Ensures every id attribute value is unique

Ensures that lists are structured correctly

Ensures <li> elements are used semantically

FIGURE 31. Storybook control panel

The *Tabs* component passes all 5 of the accessibility tests generated by “A11Y” add-on.[76] The first test simply checks if the inline text spacing can be adjusted with custom stylesheets, so that it is easier for people with cognitive disabilities to track the new lines after the line break.[129] The second test evaluates the color contrast ratio for users with a low vision.[130] The last three tests are about the semantically correct usage of HTML.[131]

Source code for the example story:

```
import Vue from 'vue';
import Tabs from './Tabs.vue';
import Tab from './Tab.vue';

import { withKnobs, text } from '@storybook/addon-knobs';
import { withA11y } from '@storybook/addon-a11y';

export default {
  title: 'Tabs',
  decorators: [withA11y, withKnobs]
};
```

```
export const TabsComponent = () => ({
  components: { Tabs, Tab },
  props: {
    text: {
      default: text('Text', 'Hello, Storybook')
    }
  },
  template:
    `<tabs>
      <tab name="Test 1" :selected="true">{{text}}</tab>
      <tab name="Test 2">{{text}}</tab>
      <tab name="Test 3">{{text}}</tab>
      <tab name="Test 4">{{text}}</tab>
    </tabs>`
})
```

Storybook comes with Vue Storefront’s de facto component library Storefront UI, which is currently under development.[128] Integrating Storybook into the projects was found to have the following positive effects for developers:[132]

1. It allows developing components in an isolated manner.
2. It encourages components to have a self-sufficient layout and styles. Components are presented individually in the Storybook, and they will not be able to rely on the styling of other components to render properly.

QA and UI/UX engineers may find it easier to test different scenarios on the component level due to the increased control over the components and make necessary modifications to the specification or design. Storybook was found to serve well as a single reference for QA engineers, UI/UX engineers and front-end developers of the current state of the development, and as a good testing playground for creating new pages.

5 BENCHMARKING AND EVALUATION (Fayjus)

Performance is a crucial factor to consider that can impact retention, conversion, and overall experience of an e-commerce application. During this chapter of the thesis, the implementation would be benchmarked along with other e-commerce applications developed inside and outside of Vaimo. There are several models and paid services for measuring a web performance, such as GTMatrix, Webperformance, LoadImpact, Pingdom. However, as the scope of this research is limited to front-end of e-commerce applications, the guidelines from the RAIL (Response, Animation, Idle, Load) performance measuring model would be used since it is a user-centric model that could be executed with free-to-use tools, such as Chrome DevTools, Lighthouse, and WebPageTest.[133] The projects that would be compared include the implementation built with VSF during this thesis, the demo VSF application from Divante, other PWA solutions developed inside and outside of Vaimo and a traditional Magento front-end based application built by Vaimo. In the Table 4, the list of projects that would be used for benchmarking are shown along with related information.

Table 4. List of projects used for benchmarking

Project	Front-end Platform	Back-end Platform	Hosting	Description
Thesis Implementation	VSF	Magento 2	GCP (dev)	The project that was built during this thesis.
Demo VSF	VSF	Magento 2	Storefront Cloud	The official demo website of VSF build and maintained Divante.[134]
Kubota	VSF	-	-	A case study of live project from VSF. [135]
ScandiPWA Demo	ScandiPWA	Magento 2	-	The Official demo website of ScandiPWA.[136]
Deity Falcon Demo	Deity Falcon	Magento 2	-	The Official demo website of Deity Falcon.[137]

Front-commerce Demo	Front-commerce	Magento 2	-	The Official demo website of front-commerce.[138]
Inchoo	PWA Studio	Magento 2	-	The PWA Studio demo website build by Inchoo.[139]
PWA Studio project	PWA Studio	Magento 2	GCP	A PWA studio-based project developed by Vaimo
Traditional M2 project	Magento 2	Magento 2	GCP	A traditional Magento 2 front-end based project by Vaimo

As instructed in the RAIL performance measuring model and in accordance with the earlier research during this thesis, the performance, accessibility, best practices, SEO, and PWA features for the above-mentioned projects have been audited with Lighthouse. [133] The data from the audits would be used for a further analysis (shown in APPENDIX 1).

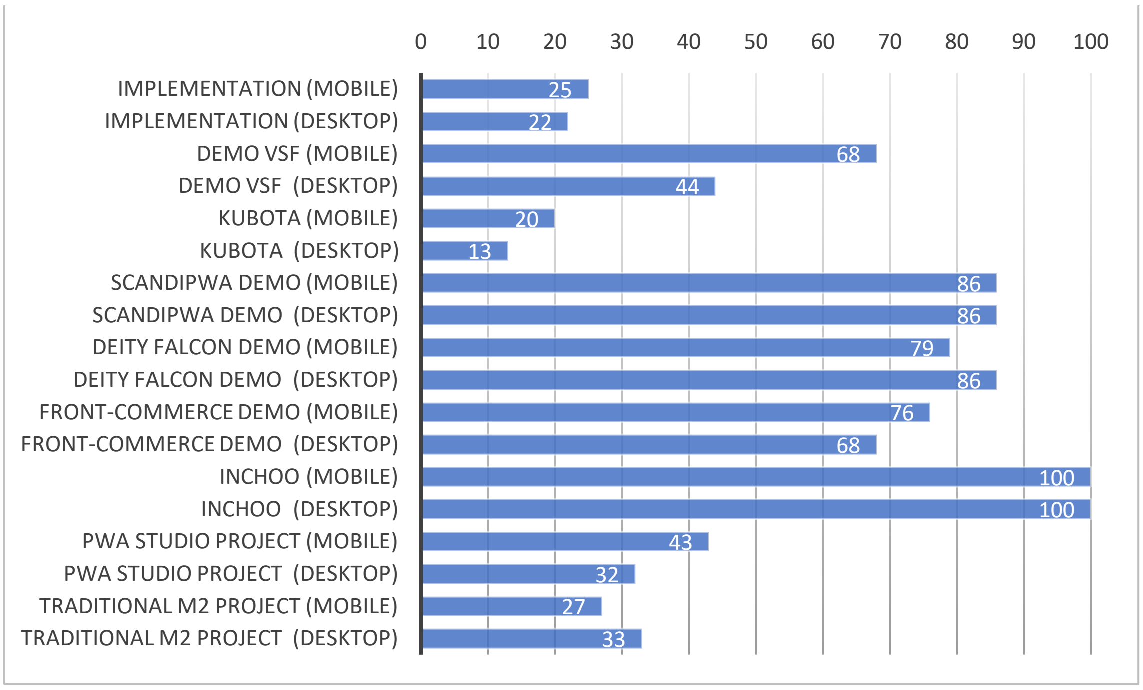


FIGURE 32. Performance benchmarking with Lighthouse

Figure 32 contains the performance scores audited by Chrome Lighthouse. The thesis implementation has the second lowest score in comparison to all the other solutions mentioned. After the initial investigation, the reasoning has been established for this unexpectedly low-performance score. Since the implementation was hosted within a development environment, it was not equipped with a performant server. The first opportunity to increase the score that was suggested by Lighthouse is to reduce the server response time or “time to first byte” (TTFB). From the front-end of the application, there is no potential action that could resolve this issue. However, the following actions could be taken from the server-side for a better TTFB.

1. Optimizing the server’s application logic to prepare pages faster. The server frameworks usually provide recommendations for this.
2. Optimizing server queries for databases or migrating to faster database systems.
3. Upgrading server hardware to have more memory or CPU. [140]

The VSF demo application built and maintained by the creator of the platforms has also less than an average performance score comparing to demo applications provided by other modern front-end based e-commerce PWA which means that the possible score that could be reached by the implementation was also throttled. The performance opportunity that was common in all the VSF based solutions was to minimize the main thread work, which means reducing the time spent parsing, compiling, and executing JS. The PWA Studio project developed by Vaimo is hosted in the live environment. Its performance score resides in the second quartile and also fails to reach the PWA studio potential set by project Inchoo. The last project in the figure is a traditional Magento 2 project developed by Vaimo and hosted in the live environment, which also has a score under the second quartile.

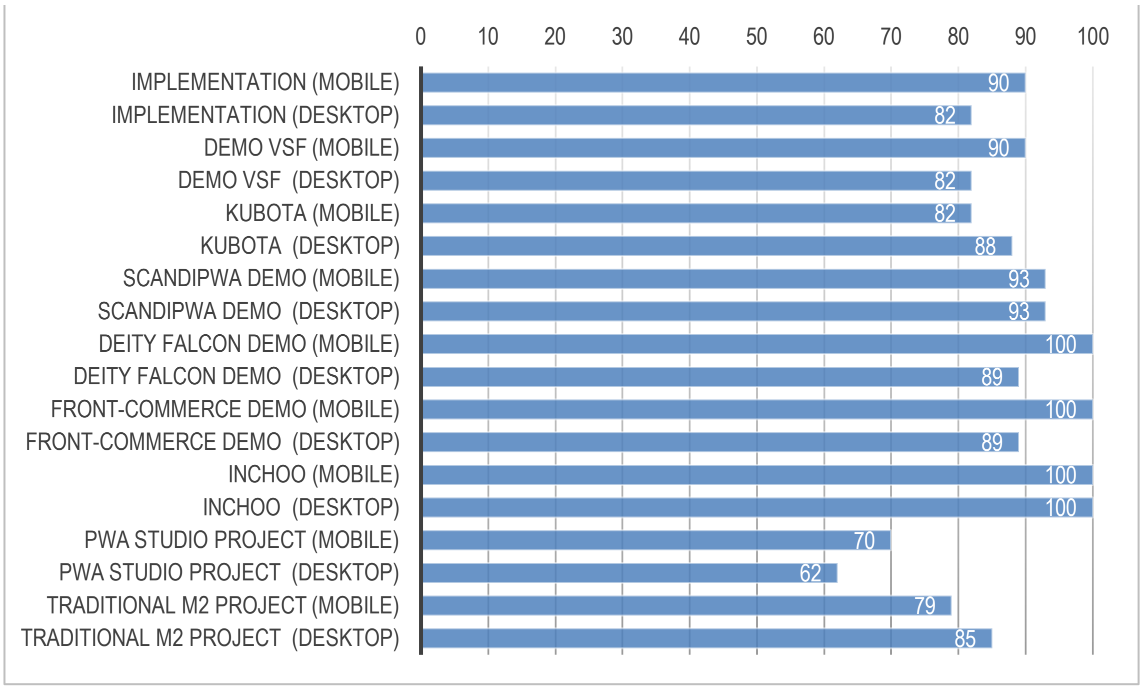


FIGURE 33. Accessibility benchmarking with Lighthouse

VSF has the third-highest accessibility score among all the demo PWAs from figure 33. During the implementation, it was possible to maintain the potential of the demo VSF application in this aspect. Both the PWA studio and traditional Magento 2 project developed by Vaimo have a lower score than the implementation in this category. The following issue, "`[user-scalable="no"]` is used in the `<meta name="viewport">` element or the `[maximum-scale]` attribute is less than 5", is common in all the VSF based solutions. Furthermore, another issue that is specific to the implementation is that "Background and foreground colors do not have a sufficient contrast ratio." If these issues could be addressed, the implementation could reach the highest possible accessibility score.

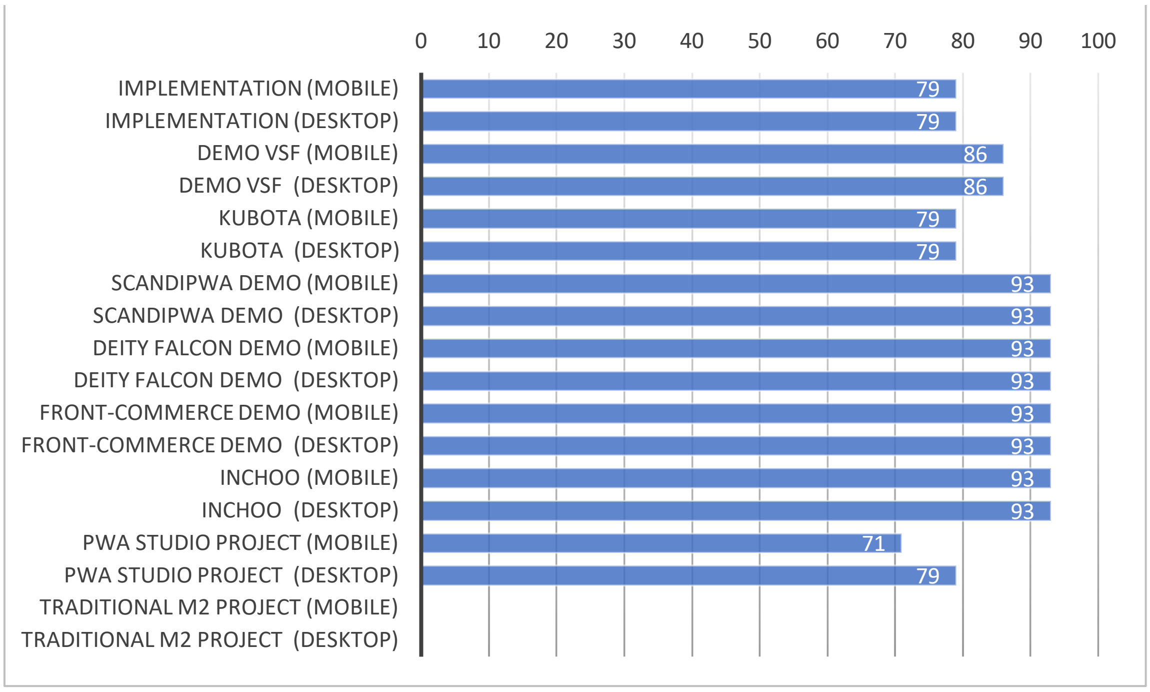


FIGURE 34. Best Practices benchmarking with Lighthouse

The implementation shares a similar score range with the project Kubota and PWA studio project, as observed in figure 34. For all the demo PWAs other than the demo VSF the best practices score is 93. To reach the potential score of demo VSF project, following issues are needed to be resolved within the implementation,

1. Uses `document.write()`
2. Browser errors were logged to the console

The common issue for both Demo VSF and the implementation is that they do not use HTTP/2 for all of their resources.

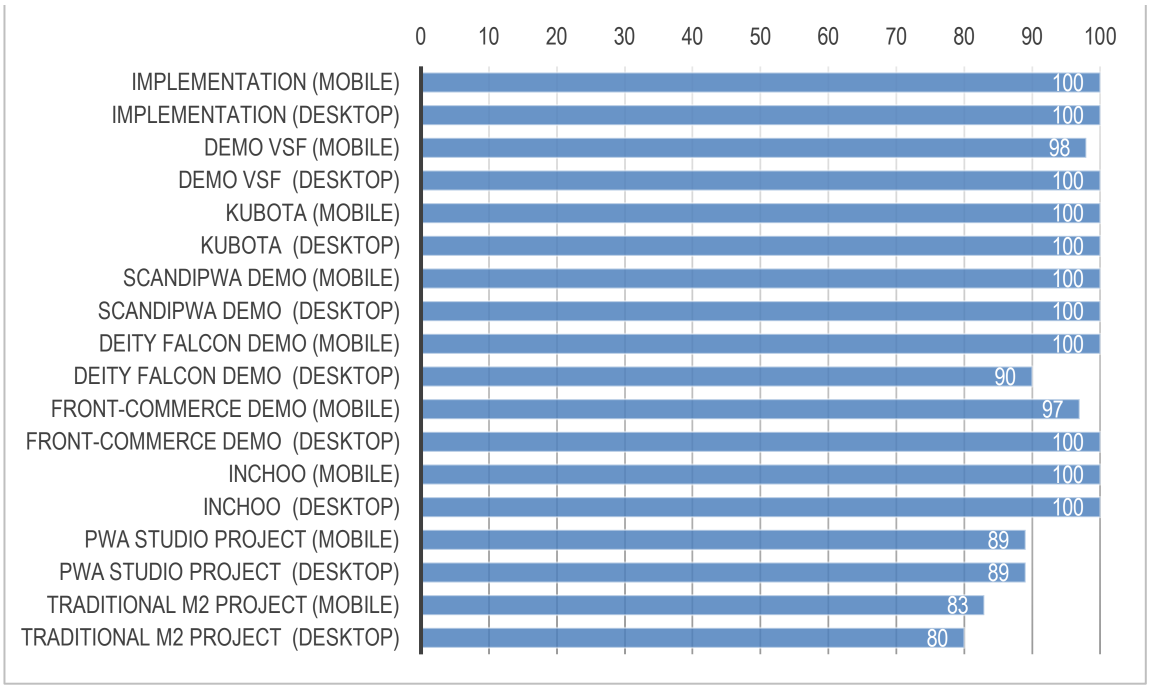


FIGURE 35. SEO benchmarking with Lighthouse

As shown in figure 35, The implementation has achieved the highest possible SEO score in both a mobile and a desktop device. Achieving a perfect score in SEO is a common characteristic for most demo PWA websites from the table. Thus, it could be considered a standard. Although they are not perfect, the SEO scores for other projects from Vaimo also hover in the fourth quartile.

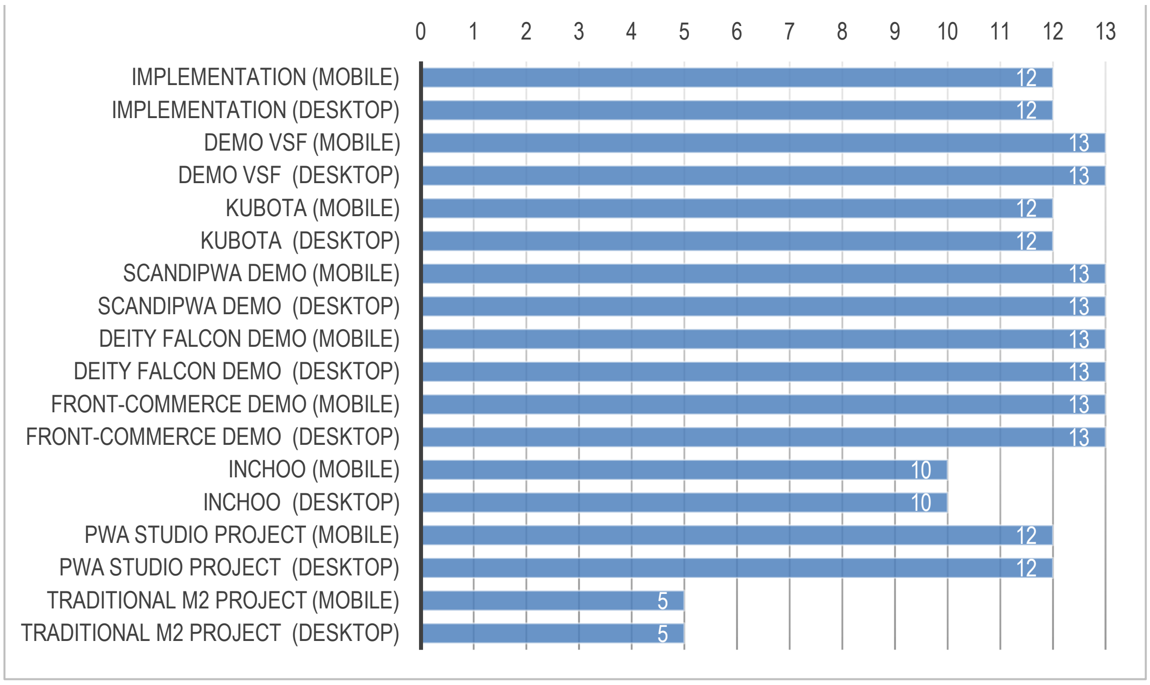


FIGURE 36. PWA benchmarking with Lighthouse

Since most of the platforms mentioned in the benchmarking table market themselves as PWAs, it is essential to examine whether the aspects of PWA are well implemented within them. As seen in figure 36, most of the demo PWA applications have a perfect score in this category, except the project Inchoo. Despite having outstanding scores in all the other sections, the project Inchoo could still not be considered a PWA fully as it did not register a service worker and provide offline browsing support. Both the thesis implementation and the PWA studio project were missing out on one aspect of being a PWA- "Page load is not fast enough on mobile networks", which is a direct consequence of their low TTFB. In the future, more research should be carried out on this issue to understand whether it could be resolved by optimizing a server response or whether a further front-end optimization would be needed.

The speed of the local development has been positively affected by the *Hot reload* feature of the `vue-loader` package.[141src] When editing a component file with hot reload enabled, all instances of the component are swapped in without the page reload. The module preserves the current state of the application. This dramatically improves the development experience for front-end developers. Combined with the Vue Devtools, the Vue framework proves to be fast and convenient for the front-end development on the local instances.

6 FUTURE POSSIBILITIES (Dair)

Vue Storefront proves to be the production-ready solution with a long list of features that keeps growing continually.[142] It has one of the broadest platform support, the most convenient NoSQL/ElasticSearch database back-end and SSR. The community is strong with many active contributors, and the solution is open-source and free for the commercial use.

Nonetheless, like any other PWA solution in the market, the technology is not mature enough to have a total support for every kinds of business needs. The small number of payment providers, incomplete documentation and missing Magento 2 features[143] need time to be developed which makes the Vue Storefront repelling for the clients who favor reliable and well-tested solutions, such as Magento. Poor performance optimization and inability to install VSF as a dependency package are one of many improvement areas for VSF. Continuation of stable development and releases can help to change this situation with time.

In the case of Vaimo and other service providers, the best strategy would be to continue supporting and using the PWA. Developing generic modules and defining the best practices inside the company will help to establish a robust technical baseline for PWA solutions. The benefits for the clients, developers, and development teams discovered in this document can be integrated into the workflow and improved.

ROADMAP FOR THE PROJECT

Since the module *vsf-external-checkout* is a temporary workaround for the unsupported payment options, the project can find an advantage in adapting the Klarna payment module for VSF, when it will be released or implemented.

There were no major API modifications done during the implementation of the project, described in this document. However, if the situation changes or if other projects will have the customizations of API extensions, the *VuePress* can be used to create the API or project documentation.

PWA FEATURES

One of the main advantages of PWA to regular websites is the native support for the device hardware, e.g. microphone or camera, although not as powerful as it is in native applications. Despite that, in typical implementations of e-commerce systems, what PWA offers will be enough. Augmented Reality (AR) is forecasted to be one of the revolutionary methods of how e-commerce retail will be shaped in the near future.[144]

Most popular arguments in favor of native applications include better access to device resources, offline access, smooth UX, and the ability to store user settings (as well as payment information), whereas web applications are best known for the instant access, platform-agnosticism, and the ability to push new content without application updates.[145]

A proof for the closing technology gap in web technologies is also justified by the technologies that enable running non-JavaScript code in the web browser. WebGL and WebAssembly are the most famous examples.

With the rise of AR, features, like virtual trials, can reshape the way users shop online. Augmentations may be used on product pages, social media, and advertising. Developers of the e-commerce experience will not be expected to write AR-specific code since it will be distributed as a library. However, products will require to have 3D models.

CONTINUOUS TESTING

Integrating the automated testing to the CI/CD is known as continuous testing. Specifically, it is the practice in which tests are run as part of the build pipeline so that every check-in and deployment is validated. In VSF, with frameworks like Jest and Cypress, this will ensure the success of every test case and identify that changes can be deployed into production while preserving the high quality of the changes.[146]

DESIGN SYSTEMS IN MULTICHANNEL AND OMNICHANNEL RETAIL

Design systems may help clients to scale their e-commerce solutions to more than a single touchpoint and make each of the consumer interactions unified by the same UX. Even though the channels of the retailers may have no logical or business connections with one another (multichannel), the UI may need to be copied or reused from existing solutions to create a smooth and recognizable user experience for the consumers. Design systems and a component library may be useful for scaling the design among multiple systems.[147] Additionally, the Web Components is considered to be a safe and future-proof technology for the projects that want to reuse the UI components across multiple platforms.[src]

7 CONCLUSION (Fayjus)

The purpose of this thesis was to improve the overall quality and efficiency of developing the front-end of e-commerce services by integrating modern methodologies and workflow. During the thesis, the purpose and the related aims were approached both theoretically and practically. The list of tasks that were carried out during the thesis is provided in APPENDIX 2.

The benefits of technologies, such as PWA, Headless Architecture, Design System, Automated Web Testing and other modern practices, were researched and assessed to the benefits for both clients and development teams. The potential of modern web front-end has been investigated, and a reference architecture has been established to actualize that potential. It was found that most uprising e-commerce front-end platforms have some shared characteristics and elements within their architecture, which makes them both developer-friendly and fast for development. These elements and their impact on the front-end development and workflow have been further investigated before the implementation was attempted.

The implementation reflected the theoretical discoveries of the thesis. It was one of the first headless architecture based PWA solutions developed within Vaimo Finland Oy and the first to use the Vue Storefront platform. Vue Storefront was found to be the most active among all the PWA solutions discovered and compared during the thesis, which are compatible with Magento 2. Since it contained almost every element of the modern web front-end architecture, the development resulted in an easy and straightforward experience, compared to the development with Magento.

As a result of headless architecture and atomic design principles, it was possible to create a reusable front-end component that was not dependent on any other component and that has already been reused within another VSF project. The development process of this component also demonstrates the simplicity of data flow achieved with state management and custom events.

One downside of using emerging platforms, such as VSF, is that the ecosystem revolving around it has not matured enough to provide custom integration solutions in many cases. The failure of Klarna integration within the VSF checkout step is one such scenario that occurred during the implementation. A module that could execute the checkout step externally within Magento was used as a workaround. Although the issue was mitigated by using the module, the process violates both the headless architecture and design system approach. However, the installation of the *vsf-external-checkout* and a *vsf-nosto-tagging* module established the fact that reusing modules within VSF is not a complicated procedure. The module installation and management would have been easier for developers if it could have been installed with the package manager, a feature that VSF does not fully support at the moment. The process of publishing a VSF module has been researched, although no modules were produced during this thesis.

The benefits of headless architecture to increase the overall development speed, as well as the speed of CI/CD operations have been studied. The advantages of integrating automated testing and design systems were listed and tested in practice. The development workflow optimization techniques have been documented and proved to be easy and quick to setup. Future plans and possibilities were analyzed and listed as an inspiration for other implementations and understanding of technology.

During the benchmarking and evaluation process, it was discovered that despite having excellent accessibility, best practices, SEO, and PWA scores, VSF and the solutions built upon it lacks in performance compared to the demo projects from its competitors. Since the platform itself throttles the potential of achieving a high-performance score, it could be highly recommended for future projects to start with an initial performance budget.

Front-end technologies and platforms are evolving rapidly, and it is hard to determine the architecture and practices that would work best in the future. Technologies and concepts that are popular lately – Web Assembly, HTTP/2, Svelte, and Web Components – have the potential to change the way developers think about front-end. This thesis is a snapshot of the capabilities of modern web front-end within the context of e-commerce. The *Implementation* chapter has helped to establish that it is possible to integrate today's front-end methodologies into traditional e-commerce systems and adapt to proven workflow practices that have the potential to amplify satisfaction for both clients and development teams.

REFERENCES

[1] Hosting Tribunal, ‘25 Magento Statistics You Need in 2020 to Boost Your Online Business’ [Web blog post]. Cited 15.2.2020 from https://hostingtribunal.com/blog/magento-statistics

[2] Magezine Team, 2019, ‘Why is the PWA Technology such a Big Deal in 2019?’, *Magezine*, no. 2, p. 14, ‘What technologies does PWA use?’.

[3] ScandiPWA [Web log post], July 19, 2019. ‘History of Progressive Web Apps’, ‘Was Steve Jobs the first to introduce the concept of PWAs?’. Cited 15.2.2020 from https://medium.com/progressivewebapps/history-of-progressive-web-apps-4c912533a531

[4] ScandiPWA [Web log post], July 19, 2019. ‘History of Progressive Web Apps’, ‘Google introduces “Progressive Web Apps”’. Cited 15.2.2020 from https://medium.com/progressivewebapps/history-of-progressive-web-apps-4c912533a531

[5] Tom Steiner, Jeff Posnick, 2019. ‘PWA’, ‘Service workers – Service worker registrations and installability’. Cited 15.2.2020 from https://almanac.httparchive.org/en/2019/pwa#service-worker-registrations-and-installability

[6] Aleksandra Kwiecień [Web log post], June 27, 2019. ‘What PWA solutions are available for eCommerce?’, ‘How to compare PWA solutions? – Technology & integrations. Cited 15.2.2020 from https://divante.com/blog/pwa-solutions-for-ecommerce-comparison

[7] Storm, Alan [answer to Stack Overflow question], October 29, 2009. ‘Why is Magento so slow?’ [closed]. Cited 15.2.2020 from https://stackoverflow.com/questions/1639213/why-is-magento-so-slow

[8] Aleksandra Kwiecień [Web log post], June 27, 2019. ‘What PWA solutions are available for eCommerce?’, ‘How to compare PWA solutions? – Community’. Cited 15.2.2020 from https://divante.com/blog/pwa-solutions-for-ecommerce-comparison

[9] Magezine Team, 2019, ‘Why is the PWA Technology such a Big Deal in 2019?’, *Magezine*, no. 2, p. 14, ‘What technologies does PWA use?’.

[10] Magezine Team, 2019. ‘Who will win the PWA battle? Selected solutions overview’, *Magezine*, no. 2, p. 16, para. 3.

[11] Aleksandra Kwiecień [Web log post], November 27, 2018. ‘Vue Storefront, a powerful PWA frontend for various eCommerce platforms’. Cited 18.12.2019 from https://divante.com/blog/vue-storefront-integrated-platforms/

[12] Adam McCombs & Robert Banh, January 27, 2011. ‘The Definitive Guide to Magento’. Apress, p. 5. ISBN 9781430272281.

[13] Jamie Maria [Web log post], January 9, 2020. ‘Next Level Commerce: Monoliths, Headless and Service Oriented Architectures’. Cited 15.2.2020 from https://dev.to/jamiemarias/next-level-commerce-service-oriented-architectures-5023

[14] Giamir Buoncristiani [Web log post], October 3, 2018. FrontendOps. Cited 17.12.2019 from https://giamir.com/frontendops

[15] Reema Oamkumar [Web log post], March 5, 2018. Software Developer India. Cited 15.2.2020 from https://www.software-developer-india.com/advantages-and-disadvantages-of-magento/

[16] Reitsma, Jisse, 2019. ‘How will Magento extensions fit into PWA?’, Magezine, no. 2, p. 21.

[17] Storm, Alan. (n.d.) ‘Magento for Developers: Part 1 – Introduction to Magento’. Cited 17.12.2019 from https://devdocs.magento.com/guides/m1x/magefordev/mage-for-dev-1.html#2

[18] Sam Saltis [Web log post], November 9, 2019. ‘Headless Commerce Explained in 5 Minutes’. Cited 15.2.2020 from https://www.coredna.com/blogs/headless-commerce

[19] Model–view–controller. (n.d.) In Wikipedia. Cited 15.2.2020 from https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[20] Siraj ul Haq [Web log post], May 2, 2018. ‘Introduction to Monolithic Architecture and MicroServices Architecture’. Cited 15.2.2020 from https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

[21] Sam Dutton, Alan Kent, 2019. ‘E-commerce – First Contentful Paint (FCP)’. Cited 15.2.2020 from https://almanac.httparchive.org/en/2019/ecommerce#first-contentful-paint-fcp

[22] Johnson, Carele [answer to Quora question], June 14, 2017. ‘How long does it take to build an eCommerce website with magento with features like a shopping cart and search engine and managing user accounts?’ Cited 15.2.2020 from https://www.quora.com/How-long-does-it-take-to-build-an-eCommerce-website-with-magento-with-features-like-a-shopping-cart-and-search-engine-and-managing-user-accounts

[23] Spike (software development). (n.d.) In Wikipedia. Cited 15.2.2020 from https://en.wikipedia.org/wiki/Spike_(software_development)

[24] Salvatore Zappalà [Web log post], April 22, 2015. ‘Magento: why complex doesn’t mean good – Complete lack of documentation’. Cited 15.2.2020 from https://medium.com/@salvoadriano/magento-why-complex-doesn-t-mean-good-1f15992202de

[25] Raphaël Benitte, Sacha Greif, Michael Rambeau, 2019. Cited 15.2.2020 from https://stateofjs.com/

[26] Raphaël Benitte, Sacha Greif, Michael Rambeau, 2019. ‘Front End Framworks’. Cited 15.2.2020 from https://2019.stateofjs.com/

[27] Hlebowitsh, Nadia [Web log post], June 10, 2019. ‘2019 Stats on Top JS Frameworks: React, Angular & Vue’. Cited 15.2.2020 from https://www.tecla.io/blog/2019-stats-on-top-js-frameworks-react-angular-and-vue/

[28] Badri, Ajay [Web log post]. (n.d.) Cited 15.2.2020 from https://seilevel.com/requirements/user-stories-technical-stories-agile-development-productmanagement

[29] Hubert Baumeister, Horst Lichter, et al., 2017. ‘Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22-26, 2017, Proceedings’. p. 223. ISBN 978-3-319-57633-6. Cited 15.2.2020 from https://books.google.fi/books?id=RtpCDwAAQBAJ&pg=PA223&lpg=PA223&dq=software+is+not+designed+to+support+agile&source=bl&ots=W_jE3BQypn&sig=ACfU3U3tdIvJBgaKbNCXyEKraLW8YHiPGg&hl=en&sa=X&ved=2ahUKEwjqvPz9977mAhVSxosKHcYWA0AQ6AEwCnoECAkQAQ#v=onepage&q=software%20is%20not%20designed%20to%20support%20agile&f=false

[30] Gash, Dave [Web log post], February, 12, 2019. ‘Measuring Performance’. Cited 15.2.2020 from https://developers.google.com/web/fundamentals/performance/get-started/measuringperf-2

[31] Weicot.com. ‘Architectural diagrams’. Cited 18.12.2019 from http://www.weicot.com/dev/guides/v2.0/architecture/archi_perspectives/arch_diagrams.html

[32] Kyle Vermeulen, April 9, 2015. ‘How do you keep up’. Cited 03.02.2020 from https://www.sitepoint.com/how-do-you-keep-up/

[33] Micah Godbolt, February 2016. ‘Frontend Architecture for Design Systems’ O'Reilly Media, Inc, Ch. 1. ISBN: 9781491926789.

[34] Hosting Tribunal, ‘What percentage of Internet Traffic is Mobile in 2020’ [Web blog post]. Cited 03.2.2020 from https://hostingtribunal.com/blog/mobile-percentage-of-traffic/#gref

[35] ‘Official Mozilla documentation on Web API’. (n.d.). Cited 15.02.2020 from https://developer.mozilla.org/en-US/docs/Web/API

[36] Andy Parker, June 27, 2019. ‘5 Reasons Why UX Matters Now More Than Ever’. Cited 03.02.2020 from https://themanifest.com/web-design/5-reasons-why-ux-matters-now-more-ever

[37] Ann Bednarz, March 24, 2016. ‘The networked world’ Cited 03.02.2020 from https://www.networkworld.com/article/3046457/the-networked-world.html

[38] ‘Web Dev site’. (n.d.). Cited 15.02.2020 from https://web.dev/

[39] Micah Godbolt, February 2016. ‘Frontend Architecture for Design Systems’ O'Reilly Media, Inc, Ch. 3. ISBN: 9781491926789.

[40] Harshal Patil, September 05, 2019. ‘Contemporary Front-end Architectures’. Cited 10.2.2020 from https://blog.webf.zone/contemporary-front-end-architectures-fb5b500b0231

[41] Elgabry, Omar [Web log post], April 1, 2019. 'Component Based Architecture'. Cited 15.2.2020 from https://medium.com/omarelgabrys-blog/component-based-architecture-3c3c23c7e348

[42] Kremic, Ned [Web log post]. (n.d.) 'Horizontal and Vertical User Stories - Slicing the Cake'. Cited 15.2.2020 from https://www.deltamatrix.com/horizontal-and-vertical-user-stories-slicing-the-cake/

[43] Shapiro, Dan. June 16, 2016. 'Understanding Component-Based Architecture'. Cited 15.2.2020 from https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238

[44] Component-based software engineering. (n.d.) In Wikipedia. Cited 15.2.2020 from https://en.wikipedia.org/wiki/Component-based_software_engineering

[45] Afonso, Francisco, September 4, 2019. 'Component-Based Architecture: What Is And Why You Should Care'. Cited 15.2.2020 from https://www.outsystems.com/blog/3-reasons-invest-component-based-architecture.html

[46] Neville-O'Neill, Brian [Web log post], July 8, 2019. 'State management pattern in JavaScript: Sharing data across components'. Cited 15.2.2020 from https://dev.to/bnevilleoneill/state-management-pattern-in-javascript-sharing-data-across-components-2gkj

[47] Biton, Yaron [Web log post], February 27, 2017. 'Who needs state management anyways?'. Cited 15.2.2020 from https://medium.com/@vyaron/who-needs-state-management-anyways-55e6d1c74239

[48] Polevoi, Robert [answer to Quora question], August 11, 2017. 'What does state-management even mean and why does it matter in Front End Web Development with frameworks like React or Vue?'. Cited 15.2.2020 from https://www.quora.com/What-does-state-management-even-mean-and-why-does-it-matter-in-Front-End-Web-Development-with-frameworks-like-React-or-Vue

[49] Crawford, Charlie [Web log post], August 24, 2016. 'What the Flux? An Overview of the React State Management Ecosystem'. Cited 15.2.2020 from https://thenewstack.io/flux-overview-react-state-management-ecosystem/

[50] In-Depth Overview. (n.d.) In Flux docs. Cited 15.2.2020 from https://facebook.github.io/flux/docs/in-depth-overview/

[51] Working with UI Store (Interface state). (n.d.) In Vue Storefront docs. Cited 15.2.2020 from https://docs.vuestorefront.io/guide/core-themes/ui-store.html

[52] 'Routing with PWA Studio'. (n.d.) In Magento PWA Studio docs. Cited 15.2.2020 from https://magento.github.io/pwa-studio/peregrine/routing/

[53] Jordan Irabor, June 19, 2019. 'Router options compared: Vue Router, Voie, Vue-routisan and Vue-route'. Cited 12.02.2020 from https://blog.logrocket.com/router-options-compared/

[54] ‘Official Vue Router documentation’. (n.d.). Cited 15.02.2020 https://router.vuejs.org/

[55] Marco Botto, July 08, 2016. ‘Front-end JavaScript single page application architecture’. Cited 12.02.2020 from https://marcobotto.com/blog/frontend-javascript-single-page-application-architecture/

[56] Sunny Singh, January 20, 2019. ‘The benefits and origins of Server Side Rendering’. Cited 12.02.2020 from https://dev.to/sunnysingh/the-benefits-and-origins-of-server-side-rendering-4doh

[57] Jason Miller, Addy Osmani, February 2019. ‘Rendering on the Web’. Cited 12.02.2020 from https://developers.google.com/web/updates/2019/02/rendering-on-the-web

[58] Paul Lewis, February 12, 2019. ‘Rendering Performance’. Cited 12.02.2020 from https://developers.google.com/web/fundamentals/performance/rendering

[59] Chris Minnick, April 19, 2016. ‘The Real Benefits of the Virtual DOM in React.js’. Cited 12.02.2020 from https://www.accelebrate.com/blog/the-real-benefits-of-the-virtual-dom-in-react-js/

[60] Areknawo, February 14, 2019. ‘DOM performance case study’. Cited 12.02.2020 from https://areknawo.com/dom-performance-case-study/

[61] ‘Official React documentation on reconciliation’. (n.d.). Cited 15.02.2020 https://reactjs.org/docs/reconciliation.html

[62] Zeno Rocha, May 15, 2014. ‘Book of Modern Front-end Tooling’. Cited from http://zenorocha.github.io/book-of-modern-frontend-tooling/

[63] Ashley Nolan, November 12, 2019. ‘The Front-End Tooling Survey 2019 - Results’. Cited 20.02.2020 from https://ashleynolan.co.uk/blog/frontend-tooling-survey-2019-results

[64] MDN contributors, February 24, 2020. ‘MDN Web Docs Glossary: Definitions of Web-related terms - CSS preprocessor’. Cited 20.02.2020 from https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor

[65] Bortz, Robin. (n.d.) ‘Web Performance Testing’. Cited 15.2.2020 from https://www.qualitestgroup.com/white-papers/web-performance-testing/

[66] Accessibility testing. (n.d.) In W3. Cited 15.2.2020 from https://www.w3.org/wiki/Accessibility_testing

[67] William E. Lewis, June, 2017. ‘Software Testing and Continuous Quality Improvement, 3rd Edition’, ‘Chapter 1. A Brief History of Software Testing’. ISBN: 9781351722209. Cited 15.2.2020 from
https://learning.oreilly.com/library/view/software-testing-and/9781351722209/xhtml/ch01.xhtml#ch01

[68] Web Application Testing Complete Guide (How To Test A Website). (n.d.) In Sofware Testing Help. Cited 27.2.2020 from https://www.softwaretestinghelp.com/web-application-testing/

[69] Pittet, Sten [Web log post]. (n.d.) ‘The different types of software testing’. Cited 27.2.2020 from https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing
[70] Zaidman, Vitali. February 18, 2019. ‘An Overview of JavaScript Testing in 2019’. Cited 27.2.2020 from https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2019-264e19514d0a
[71] Raphaël Benitte, Sacha Greif, Michael Rambeau, 2019. ‘Testing. Cited 27.2.2020 from https://2019.stateofjs.com/testing/
[72] Kim Krause Berg [Web log post], January 9, 2019. ‘Website Accessibility & the Law: Why Your Website Must Be Compliant’. Cited 27.2.2020 from https://www.searchenginejournal.com/website-accessibility-law/285199/
[73] What is accessibility? (n.d.) In MDN web docs. Cited 27.2.2020 from https://developer.mozilla.org/en-US/docs/Learn/Accessibility/What_is_accessibility
[74] Assistive Technology. (n.d.) In Yale University – Usability & Web Accessibility. Cited 27.2.2020 from https://usability.yale.edu/web-accessibility/articles/assistive-technology
[75] WCAG 2.1 at a Glance. (n.d.) In W3.org. Cited 27.2.2020 from https://www.w3.org/WAI/standards-guidelines/wcag/glance/
[76] Resources. (n.d.) In The A11Y Project. Cited 27.2.2020 from https://a11yproject.com/resources/
[77] Web Accessibility Audit. (n.d.) In Interactive Accessibility. Cited 27.2.2020 from https://www.interactiveaccessibility.com/services/accessibility-audit
[78] Fanguy, Will [Web log post], June 24, 2019. ‘A comprehensive guide to design systems’. Cited 27.2.2020 from https://www.invisionapp.com/inside-design/guide-to-design-systems/
[79] Thoms, Jess [Web log post], November 21, 2017. ‘Designing at scale: How industry leaders leverage design systems’. Cited 27.2.2020 from https://www.invisionapp.com/inside-design/scale-design-systems/
[80] Rutherford, Zack [Web log post], September 29, 2017. ‘Design Systems vs. Pattern Libraries vs. Style Guides – What’s the Difference?’ Cited 27.2.2020 from https://www.uxpin.com/studio/blog/design-systems-vs-pattern-libraries-vs-style-guides-whats-difference/
[81] Serizer, Inès [Web log post], March 3, 2019. ‘What the heck is a design system?’ Cited 27.2.2020 from https://uxdesign.cc/what-the-heck-is-a-design-system-c89a8ea73b0d
[82] Kahn, Tarif [Web log post], July 11, 2019. ‘Inside a Design System: What You Need to Know and Why’ Cited 27.2.2020 from https://uxplanet.org/inside-a-design-system-what-you-need-to-know-and-why-198ab645d5a3

[83] Frost, Brad, November 28, 2016. ‘Atomic Design’. Cited 27.2.2020 from https://atomicdesign.bradfrost.com/chapter-2/

[84] Build bulletproof UI components faster. (n.d.) In Storybook. Cited 27.2.2020 from https://storybook.js.org/

[85] Build UI components. (n.d.) In Learn Storybook. Cited 27.2.2020 from https://www.learnstorybook.com/design-systems-for-developers/react/en/build/

[86] Shilman, Michael [Web log post], January 14, 2020. ‘Storybook 5.3’. Cited 27.2.2020 from https://medium.com/storybookjs/storybook-5-3-83e114e8797c

[87] Introduction to design systems. (n.d.) In Learn Storybook. Cited 27.2.2020 from https://www.learnstorybook.com/design-systems-for-developers/react/en/introduction/

[88] Rendle, Robin [Web log post], April 3, 2019. ‘What Are Design Tokens?’. Cited 27.2.2020 from https://css-tricks.com/what-are-design-tokens/

[89] Nguyen, Dominic [Web log post], November 25, 2019. ‘Storybook for design’. Cited 27.2.2020 from https://medium.com/storybookjs/storybook-for-design-3ff761c579bf

[90] Agile testing. (n.d.) In Wikipedia. Cited 27.2.2020 from https://en.wikipedia.org/wiki/Agile_testing

[91] Mayeur, Paul [Web log post], October 6, 2016. ‘What Does “Quality Assurance” Mean in an Agile World?’. Cited 27.2.2020 from https://medium.com/appian-engineering/what-does-quality-assurance-mean-in-an-agile-world-d0b5bb78479b

[92] Continuous testing. (n.d.) In Wikipedia. Cited 27.2.2020 from https://en.wikipedia.org/wiki/Continuous_testing

[93] Web Performance Testing. (n.d.) In Qualitest group. Cited 27.2.2020 from https://www.qualitestgroup.com/white-papers/web-performance-testing/

[94] BCG Digital Ventures [Web log post], March 20, 2019. ‘Working in Harmony: Using Design Systems for Better Collaboration’. Cited 27.2.2020 from https://medium.com/bcgdv-engineering/working-in-harmony-using-design-systems-for-better-collaboration-57a5dae53d65

[95] Magezine Team, 2019. ‘Who will win the PWA battle? Selected solutions overview’, *Magezine*, no. 2, p. 20

[96] Aleksandra Kwiecień, June 27, 2019. ‘What PWA solutions are available for eCommerce?’. Cited 15.02.2020 from https://divante.com/blog/pwa-solutions-for-ecommerce-comparison/

[97] ‘Official Vue Storefront documentation on how it connects with backed platforms’. (n.d.). Cited 15.02.2020 from https://docs.vuestorefront.io/guide/general/introduction.html#how-does-it-connect-with-backend-platforms

[98] ‘Official Vue Storefront documentation on how it works’. (n.d.). Cited 15.02.2020 from https://docs.vuestorefront.io/guide/general/introduction.html#how-does-it-work

[99] ‘Official Vue Storefront documentation on building themes’. (n.d.). Cited 20.02.2020 from https://docs.vuestorefront.io/guide/general/introduction.html#building-themes-in-vue-storefront

[100] ‘Official Vuex documentation’. (n.d.). Cited 20.02.2020 from https://vuex.vuejs.org/

[101] ‘Official Vue Storefront documentation on working with core styles’. (n.d.). Cited 15.02.2020 from https://docs.vuestorefront.io/guide/core-themes/stylesheets.html

[102] ‘Official Vue documentation on separation of concern’. (n.d.). Cited 15.02.2020 from https://vuejs.org/v2/guide/single-file-components.html#What-About-Separation-of-Concerns

[103] ‘Official Vue documentation on Actions’. (n.d.). Cited 20.02.2020 from https://vuex.vuejs.org/guide/actions.html

[104] ‘Official Vue documentation on Events’. (n.d.). Cited 20.02.2020 from https://vuejs.org/v2/guide/events.html

[105] ‘Official Vue documentation on One-way data flow’. (n.d.). Cited 20.02.2020 from https://vuejs.org/v2/guide/components-props.html#One-Way-Data-Flow

[106] Joshua Bemenderfer, January 09, 2017. ‘Creating a Global Event Bus with Vue.js’. Cited 22.02.2020 from https://alligator.io/vuejs/global-event-bus/

[107] ‘GitHub repository for Vue Storefront external checkout module’. (n.d.). Cited 02.03.2020 from https://github.com/Vendic/vsf-external-checkout

[108] ‘Issue 12 from Vue Storefront external checkout module’s GitHub Repository’. (n.d.). Cited 02.03.2020 from https://github.com/Vendic/vsf-external-checkout/issues/12

[109] ‘Issue 3881 from Vue Storefront’s GitHub Repository’. (n.d.). Cited 02.03.2020 https://github.com/DivanteLtd/vue-storefront/issues/3881

[110] Onsite Product Recommendations. (n.d.) In Nosto. Cited 12.3.2020 from https://www.nosto.com/products/product-recommendations/

[111] Nosto Magento 2 module. (n.d.) In Github. Cited 12.3.2020 from https://github.com/Nosto/nosto-magento2

[112] Wadher, Pratik [Web log post], August 31, 2017. ‘Introducing Argo — A Container-Native Workflow Engine for Kubernetes’. Cited 12.3.2020 from https://blog.argoproj.io/introducing-argo-a-container-native-workflow-engine-for-kubernetes-55c0b4b76fac

[113] Installing on Linux/MacOS. (n.d.) In Vue Storefront docs. Cited 12.3.2020 from https://docs.vuestorefront.io/guide/installation/linux-mac.html

[114] Ceunen, Bouwe [Web log post], June 5, 2019. ‘CI/CD With Argo On Kubernetes’. Cited 12.3.2020 from https://medium.com/axons/ci-cd-with-argo-on-kubernetes-28c1a99616a9

[115] Anastasov, Marko [Web log post], March 2, 2017. 'What is Proper Continuous Integration?'. Cited 12.3.2020 from https://semaphoreci.com/blog/2017/03/02/what-is-proper-continuous-integration.html

[116] Unit Testing. (n.d.) In Software Testing Fundamentals. Cited 12.3.2020 from http://softwaretestingfundamentals.com/unit-testing/

[117] End-to-End Testing. (n.d.) In Tutorials Point. Cited 12.3.2020 from https://www.tutorialspoint.com/software_testing_dictionary/end_to_end_testing.htm

[118] Husky. (n.d.) in Github. Cited 12.3.2020 from https://github.com/typicode/husky

[119] Singhal, Krishankant [Web log post], February 6, 2019. 'How to read Test Coverage report generated using Jest.' Cited 12.3.2020 from https://medium.com/@krishankantsinghal/how-to-read-test-coverage-report-generated-using-jest-c2d1cb70da8b

[120] Gagliardi, Valentino [Web log post], February 2, 2020. 'Jest Tutorial for Beginners: Getting Started With JavaScript Testing'. Cited 12.3.2020 from https://www.valentinog.com/blog/jest/

[121] Shevlin, Kyle [Web log post], February 23, 2017. 'Renderless Components or How Logic Doesn't Always Need a UI', 'Presentational Components'. Cited 12.3.2020 from https://kyleshevlin.com/renderless-components

[122] Snapshot Testing. (n.d.) In Jest. Cited 12.3.2020 from https://jestjs.io/docs/en/snapshot-testing

[123] Who uses Cypress? (n.d.) In Cypress docs. Cited 12.3.2020 from https://docs.cypress.io/guides/overview/why-cypress.html#Who-uses-Cypress

[124] Our mission. (n.d.) In Cypress docs. Cited 12.3.2020 from https://docs.cypress.io/guides/overview/why-cypress.html#Our-mission

[125] Corker, Seth [Web log post], December 8, 2018. 'Why Cypress is the best way to test'. Cited 12.3.2020 from https://medium.com/front-end-field-guide/why-cypress-is-the-best-way-to-test-64ec0d394bae

[126] Hiren Tanna [Web log post], February 17, 2017. 'Top 10 Benefits of Test Automation'. Cited 12.3.2020 from https://dzone.com/articles/top-10-benefits-of-test-automation

[127] Frost, Brad, November 28, 2016. 'Atomic Design'. Cited 27.2.2020 from https://atomicdesign.bradfrost.com/chapter-5/

[128] StorefrontUI - Customization-first component library. (n.d.) In Storefront-UI Storybook. Cited 12.3.2020 from https://storybook.storefrontui.io/

[129] Inline text spacing must be adjustable with custom stylesheets. (n.d.) In Deque University. Cited 12.3.2020 from https://dequeuniversity.com/rules/axe/3.3/avoid-inline-spacing

[130] Text elements must have sufficient color contrast against the background. (n.d.) In Deque University. Cited 12.3.2020 from https://dequeuniversity.com/rules/axe/3.2/color-contrast

[131] Semantics. (n.d.) In MDN web docs. Cited 12.3.2020 from https://developer.mozilla.org/en-US/docs/Glossary/Semantics

[132] Nedrich, Matt [Web log post], January 24, 2018. 'Should You Use React Storybook?' Cited 12.3.2020 from https://spin.atomicobject.com/2018/01/24/react-storybook/

[133] Meggin Kearney, Addy Osmani, Kayce Basques, Jason Miller, February 12, 2019. 'Measure Performance with the RAIL Model'. Cited 12.02.2020 from https://developers.google.com/web/fundamentals/performance/rail

[134] Divante. (n.d.). 'The demo Vue Storefront Store'. Cited 05.03.2020 from https://demo.vuestorefront.io/

[135] 'Kubota Web store'. Cited 05.03.2020 from https://kubotastore.pl/

[136] ScandiPWA. (n.d.). 'The demo ScandiPWA store'. Cited 05.03.2020 from https://demo.scandipwa.com/

[137] Deity Falcon. (n.d.). 'The demo Deity Falcon store' cited 05.03.2020 from https://demo.deity.io/

[138] Front Commerce. (n.d.). 'The demo Front Commerce store'. Cited 05.03.2020 from https://magento2.demo.front-commerce.com/

[139] Inchoo. (n.d.). 'The demo PWA studio store' cited 05.03.2020 from https://inchoopwa.com/

[140] Web Dev, May 02, 2019. 'Reduce server response times (TTFB)'. Cited 05.03.2020 from https://web.dev/time-to-first-byte/?utm_source=lighthouse&utm_medium=devtools

[141] Hot Reload. (n.d.) In Vue Loader docs. Cited 17.3.2020 from https://vue-loader.vuejs.org/guide/hot-reload.html

[142] Vue Storefront unique features. (n.d.) In Vue Storefront docs. Cited 12.3.2020 from https://docs.vuestorefront.io/guide/basics/feature-list.html#vue-storefront-unique-features

[143] Magezine Team, 2019. 'Who will win the PWA battle? Selected solutions overview', Magezine, no. 2, p. 18.

[144] Rath, Manish [Web log post], August 8, 2019. 'Here are the 3 ways AR is changing the eCommerce industry'. Cited 12.3.2020 from https://medium.com/scapic/here-are-the-3-ways-ar-is-changing-the-ecommerce-industry-9abf6b54a214

[145] Iram, Sivan [Web log post], July 7, 2017. 'Why Web Apps Are The Future Of Augmented Reality'. Cited 12.3.2020 from https://medium.com/arjs/why-web-apps-are-the-future-of-augmented-reality-c503e796a0c5

[146] Auerbach, Adam [Web log post], August 3, 2015. 'Part of the Pipeline: Why Continuous Testing Is Essential'. Cited 12.3.2020 from https://www.techwell.com/techwell-insights/2015/08/part-pipeline-why-continuous-testing-essential

[147] Moffat, Branwell [Web log post]. (n.d.) 'Omnichannel vs. multichannel: What's the difference and who is doing it?' Cited 12.3.2020 from https://www.the-future-of-commerce.com/2017/09/13/omnichannel-vs-multichannel/

[148] Google Chrome Developers, June 18, 2019. 'Web Components: The Secret Ingredient Helping Power The Web'. Cited 17.3.2020 from https://www.youtube.com/watch?v=YBwgkr_Sbx0

APPENDIX

Project	Device	Performance	Accessibility	Best Practices	SEO	PWA
Implementation	Mobile	25	90	79	100	12
Implementation	Desktop	22	82	79	100	12
Demo VSF	Mobile	68	90	86	98	13
Demo VSF	Desktop	44	82	86	100	13
Kubota	Mobile	20	82	79	100	12
Kubota	Desktop	13	88	79	100	12
ScandiPWA Demo	Mobile	86	93	93	100	13
ScandiPWA Demo	Desktop	86	93	93	100	13
Deity Falcon Demo	Mobile	79	100	93	100	13
Deity Falcon Demo	Desktop	86	89	93	90	13
Front-commerce Demo	Mobile	76	100	93	97	13
Front-commerce Demo	Desktop	68	89	93	100	13
Inchoo	Mobile	100	100	93	100	10
Inchoo	Desktop	100	100	93	100	10
PWA Studio project	Mobile	43	70	71	89	12
PWA Studio project	Desktop	32	62	79	89	12
Traditional M2 project	Mobile	27	79	-	83	5
Traditional M2 project	Desktop	33	85	-	80	5

Task	Scope
Investigating and documenting the drawbacks of existing solutions	Theory
Investigating and documenting the potentials of modern front-end – Methodologies, architectures and tools	Theory
Investigating and documenting the potentials of modern front-end – Workflow Optimization	Theory
Comparing available e-commerce platforms based on modern front-end	Theory
Installing VSF and integrating with a Magento 2 instance	Implementation
Creating a new VSF theme	Implementation
Understanding VSF structure and architecture	Implementation
Developing custom reusable components in VSF (HorizontalMenu)	Implementation
Publishing components	Implementation
Reusing existing modules (VSF External Checkout)	Implementation
Reusing existing modules (Nosto)	Implementation
Documenting the CI/CD process	Implementation
Writing automated front-end tests	Implementation
Establishing component maintenance procedure	Implementation
Benchmarking the application and comparing with existing solutions	Implementation
Investigating current limitations & future possibilities	Theory