



VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Thang Nguyen

JAMSTACK: A MODERN SOLUTION FOR E- COMMERCE

School of Technology
2022

CONTENTS

1	INTRODUCTION	7
1.1	Traditional Website.....	7
1.2	Jamstack.....	8
2	REQUIREMENTS, SPECIFICATIONS, AND DESIGN	10
2.1	Requirements.....	10
2.2	Specifications	10
2.3	Design.....	11
2.3.1	Backend design	12
2.3.2	Frontend design	13
3	TECHNOLOGIES.....	17
3.1	Introduction	17
3.2	NextJS.....	17
3.2.1	NextJS vs ReactJS.....	17
3.2.2	Features.....	17
3.2.3	Trends.....	18
3.3	Headless CMS.....	19
3.3.1	Introduction	19
3.3.2	Headless CMS vs Traditional CMS.....	19
3.3.3	Sanity.....	20
4	IMPLEMENTATION.....	22
4.1	Environment Setup	22
4.2	Headless CMS.....	22
4.3	Frontend.....	24
5	TESTING AND RESULTS	27
5.1	Testing with Cypress	27
5.2	Results in Browser.....	27
6	CONCLUSIONS	30
	REFERENCES	31
	APPENDIX 1. THE FOLDER STRUCTURE OF SANITY	32

APPENDIX 2. THE FOLDER STRUCTURE OF NEXTJS 33

LIST OF FIGURES

Figure 1. User shopping activity diagram	11
Figure 2. Jamstack workflow	12
Figure 3. Database relationship	12
Figure 4. The browser sends requests and displays the data	13
Figure 5. Wireframe of Home page	13
Figure 6. Wireframe of the Products page	14
Figure 7. Wireframe of Product Detail page	14
Figure 8. Users send requests to CMS	15
Figure 9. Wireframe of the Authentication page	15
Figure 10. Wireframe of the Cart page	16
Figure 11. Download numbers of React and NextJS on NPM Trends	19
Figure 12. User Interface of Sanity dashboard	24
Figure 13. Vendor information is connected to the product via type reference	24
Figure 14. Test results in terminal	27
Figure 15. Home page	28
Figure 16. Login successfully	28
Figure 17. Checkout with PayPal testing account	29

LIST OF CODE SNIPPETS

Code snippet 1. Product item schema	23
Code snippet 2. Page initialization in <code>_app.tsx</code>	25
Code snippet 3. The Layout component contains Header and Footer	25
Code snippet 4. Adding AWS Amplify to Profile page	26
Code snippet 5. Fetching with the SSG method	26
Code snippet 6. Fetching with the SSR method	26

LIST OF APPENDICES

APPENDIX 1. The folder structure of Sanity

APPENDIX 2. The folder structure of NextJS

LIST OF ABBREVIATIONS

API	Application Programming Interface
CDN	Content Delivery Network
CMS	Content Management System
CSS	Cascading Style Sheets
DOM	Document Object Model
GROQ	Graph-Relational Object Queries
HTML	HyperText Markup Language
ISR	Incremental Static Regeneration
PHP	Hypertext Preprocessor
SEO	Search Engine Optimization
SPA	Single Page Application
SSG	Static Site Generation
SSR	Server-Side Rendering
SWR	Stale While Revalidate
UI	User Interface
URL	Uniform Resource Locator

1 INTRODUCTION

The main purpose of this thesis was to introduce a new web development architecture Jamstack, exploring its benefits and comparing it to traditional approaches. In this context, a website was built with NextJS, Tailwinds, and Sanity to fulfill the requirements specified in e-commerce.

However, there were some possible limitations to this study. Firstly, it is a subjective opinion of the author, a fact remains that many people are still using traditional architecture and WordPress. Secondly, the approach is relatively new, thus, there are not many documents available on the internet. Lastly, there is a massive number of frameworks and libraries that may confuse beginners in the first place.

1.1 Traditional Website

Previously, we needed a folder of HTML files and CSS for the interface. HTML is the bone structure of the layout and CSS describes the attributes of HTML elements on the screen such as sizes and colors. Together, they determine how websites are displayed by browsers to the users. Then we need to set up a backend with PHP or .NET, creating a CMS which allows our customers to input their content. It will be saved into one or more databases, for example, Apache or MySQL, we can fetch records from here and display the content on our website. This was how websites were built in a traditional way which is complex.

Later, due to the high demand for online shopping, WordPress was released. It is open-source which helps to build and deploy a website within minutes. On the market, there are a variety of themes to choose from for WordPress with different functionalities, it does not require much knowledge to code a WordPress site since everything is already built-in. Until today, it is still one of the most popular frameworks with a contribution of 39.5% of all the websites in 2021, according to

a study. However, there are some limitations in WordPress that we need to overcome.

- **Slow speed:** There are some factors affecting the speed of a WordPress site such as bad themes, and lots of scripts running in the background due to the high number of plugins and size (Cortez, n.d.).
- **Large size:** WordPress is a huge framework, in version 5.5.1, the size of its core is 30.8MB, which leads to a total of 47.7MB. This is one of the main reasons why WordPress is slow (Morey, 2018).
- **Vulnerability:** Heavily relying on themes and plugins, WordPress sites can be attacked easily if those themes and plugins contain viruses. Since it is open-source, anyone can contribute to the core functionalities leading to security issues. In addition, with its high popularity, it is likely to be the primary target of hackers. According to measurements by Wordfence, more than 86 billion blocked password attacks during the first 6 months of 2021 (Dewhurst and Chamberland, 2021, p2).

1.2 Jamstack

Jamstack is an architecture designed for the modern website, it is not a specific programming language or tool. Jamstack is a new approach to development to achieve faster, more secure, and better-scale websites, JAM stands for JavaScript, API, and Markup language.

The main difference between Jamstack and other technologies is that it does not need a server. Instead of using databases, our website is connected to services using APIs and served as static files. This approach has several advantages making it one of the best ways to build websites these days.

One of the advantages is speed. Instead of rendering page contents on the server at request time, all the pages are built on CDN and ready to serve. It

makes Jamstack websites much faster than ones built in traditional methods (Wlosik, 2021). Research from Google has shown that 53% of users left the site if it does not load within three seconds (Kirkpatrick, 2016). Meanwhile, since Jamstack pages are statically built, the content is displayed very quickly.

While WordPress sites use pre-built themes, Jamstack has full control of the content and HTML structure. It is easy to add page titles, descriptions, and alt texts, helping crawler bots work more effectively. It results in the site's index being boosted naturally with less effort. In addition, page speed contributes largely to SEO as Google has recognized desktop speed as a key factor to rank websites (Southern, 2022).

Jamstack sites are safe due to their designs which have fewer attack possibilities since most potential connections to web applications or database servers are eliminated by serving pre-rendered static HTML. Meanwhile, dynamic functions in a WordPress setup are vulnerable to hackers.

Because the website is primarily static content, it can be easily deployed as files from CDN servers in multiple locations. This means that the website can be quickly and inexpensively published from new web locations, providing scalability options for businesses expanding into new markets or anticipating increased traffic from various locations.

However, it is important to note that WordPress sites do not require any coding because they are supported by a large number of themes and plugins, web development knowledge is needed to work with Jamstack.

2 REQUIREMENTS, SPECIFICATIONS, AND DESIGN

2.1 Requirements

To achieve all the requirements of an e-commerce website, the project needs the following features in the frontend

- Layout
- Login/register page
- Profile page
- List page
- Product detail page
- Cart/checkout page
- Multi-language button

The backend features include CMS.

2.2 Specifications

The specifications for the frontend are as follows:

- The layout provides a better visual experience to users with interactive elements allowing them to navigate through the website. The layout is also responsive which means it displays well on both desktop and smaller devices.
- The login/register page helps users to create an account and log in, to be more secure because the registration process comes with email verification. It is possible to change the password or get a new one if the users forget their password.
- The profile page comes after users have logged in, they can manage their favorite items or view their purchase history.
- The list page shows all the products with pagination and filters.

- The detail page provides product information, and functionalities that allow users to purchase, add or remove items to their favorite.
- The cart page lets customers increase or decrease the number of products, or delete them. The checkout section comes with multiple payment methods such as PayPal and Visa cards.
- The multi-language button to control the language of the website, available in English (by default), Finnish and Swedish.

The specifications for the backend include:

- CMS is a dashboard where the clients input and manage data such as images and content through UI, which is served later on the frontend side.

2.3 Design

The objective of this chapter is to illustrate users' shopping process and the workflow of Jamstack.

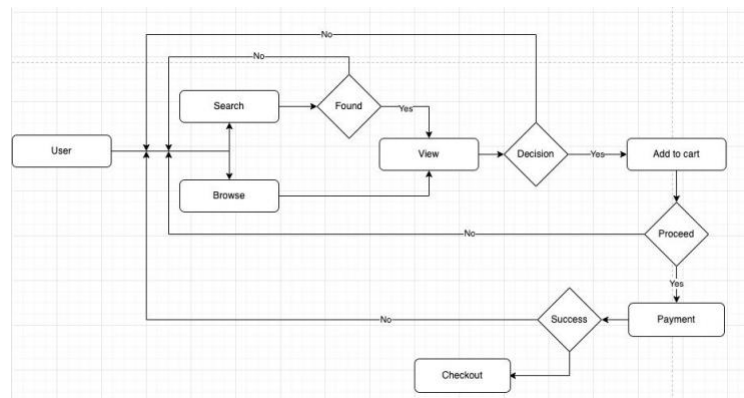


Figure 1. User shopping activity diagram

As can be seen in Figure 1, when users open the application, they see the home page where they can browse or search for products and view product details. After choosing desired items, users might add them to the cart and manage quantities. If they are not satisfied with the current items, they can delete them and start

again, otherwise, they may proceed to checkout. Then they can cancel the order or make a payment, and with a successful checkout, users are redirected to a success page and receive a notification email.

2.3.1 Backend design

Figure 2 illustrates the interaction between the backend and frontend, Sanity takes care of the CMS part, and all the content information of the project is managed here. It is possible to add brands and products in multiple languages, then these contents are served to the frontend as APIs.

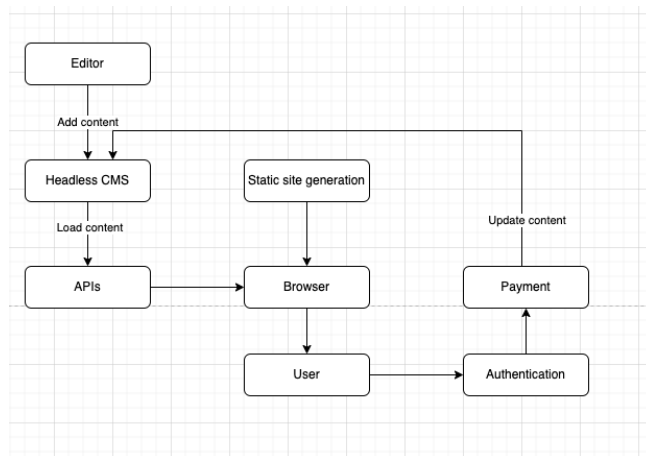


Figure 2. Jamstack workflow

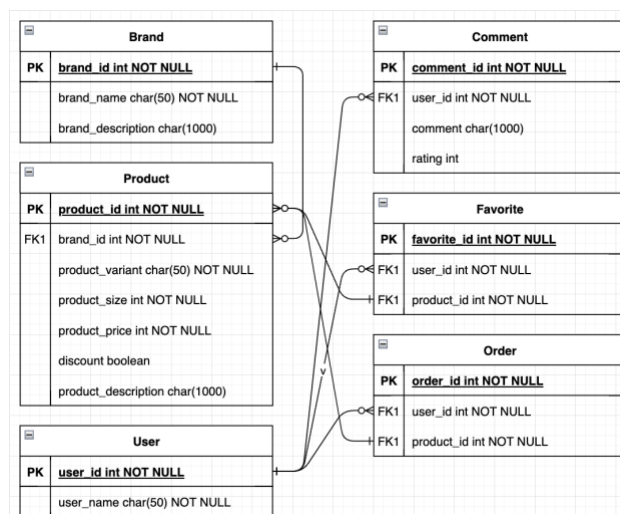


Figure 3. Database relationship

As seen in Figure 3, the Brand field connects to the Product field using the ID, so from one brand, the information of all the products belonging to that brand can be retrieved. Similarly, the User field connects to the Comment, Favorite, and Order, and users can write a review, manage their favorite products, and view their order history.

2.3.2 Frontend design

When users visit the application, depending on the page, the specific requests are sent to the CMS to get information and display it on the browser as shown in Figure 4.

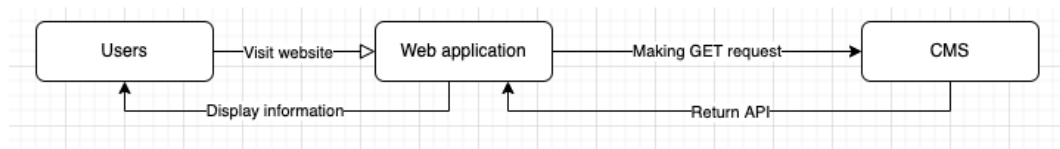


Figure 4. The browser sends requests and displays the data

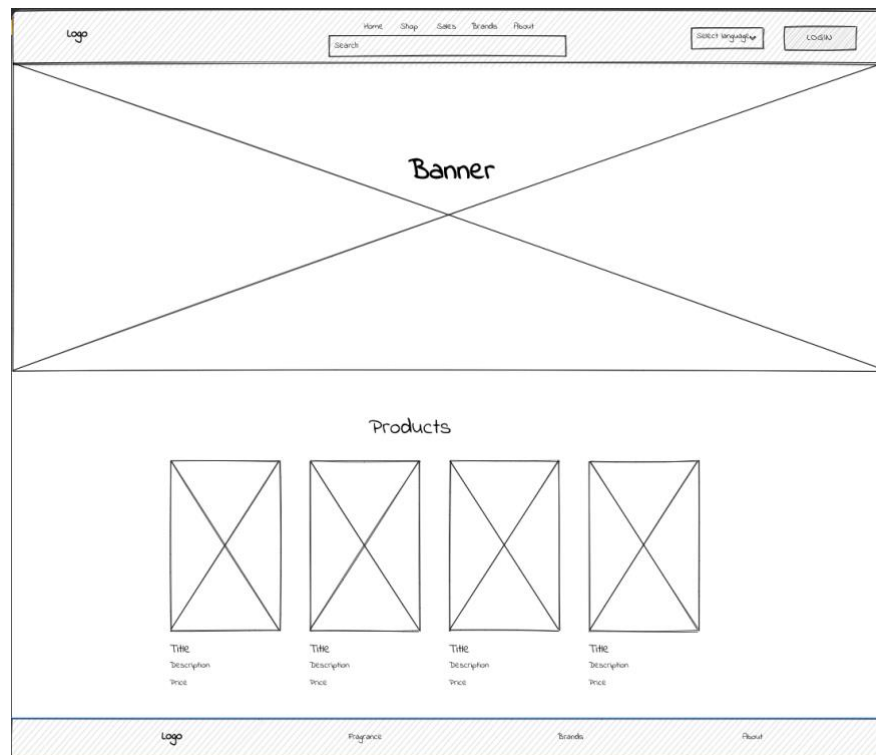


Figure 5. Wireframe of Home page

Figure 5 describes the layout of the Home page, users can navigate to other pages with links on the header section. Below is the product section which displays a list of product items based on the requests. To be specific, in the application, three requests for discount, new and best-selling products are dispatched.

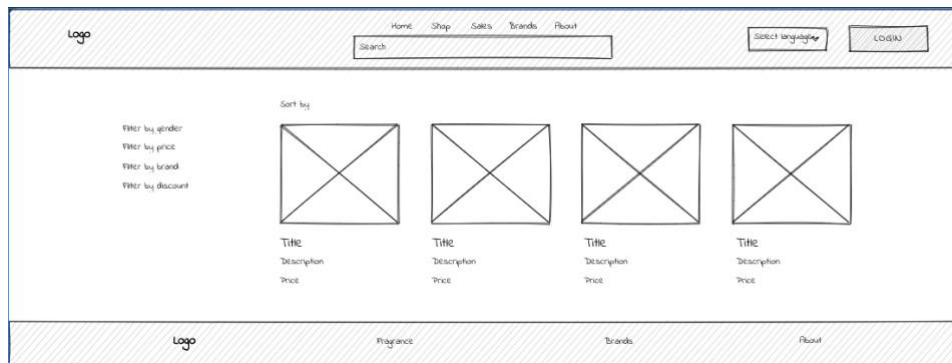


Figure 6. Wireframe of the Products page

As shown in Figure 6, users can view all the products fetched from the CMS. They can use sort and filter functions to display the items more precisely according to their needs. The request with corresponding parameters will be sent to get the desired products.

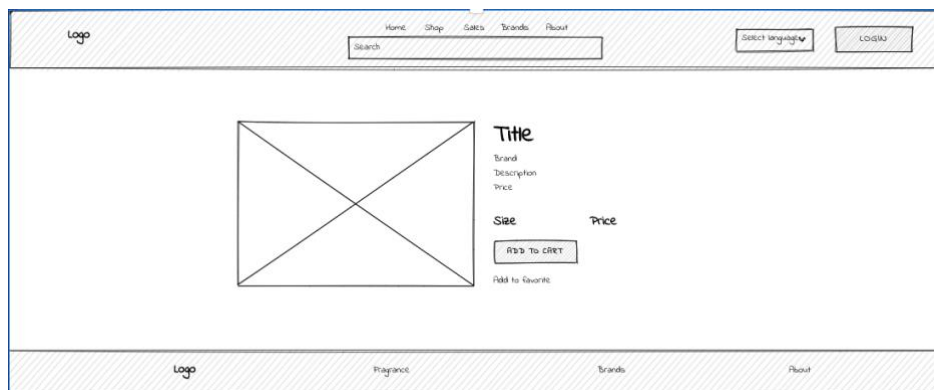


Figure 7. Wireframe of Product Detail page

From the product list, users can choose to view the detail of each product which is given in Figure 7, where users may add the item to their favorite list or the cart.

Amplify then verify users' status, if they are logged in, a request is sent to the CMS to save the favorite item or the cart to the database as in Figure 8.

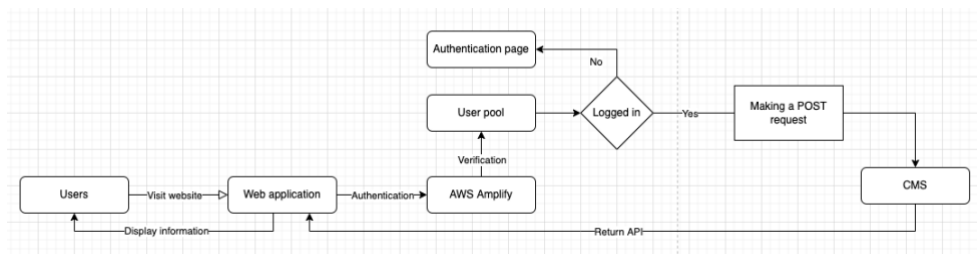


Figure 8. Users send requests to CMS

Figure 9 illustrates the layout of the Authentication page where users are redirected if they are not logged in. From there, users can sign in or create a new account.

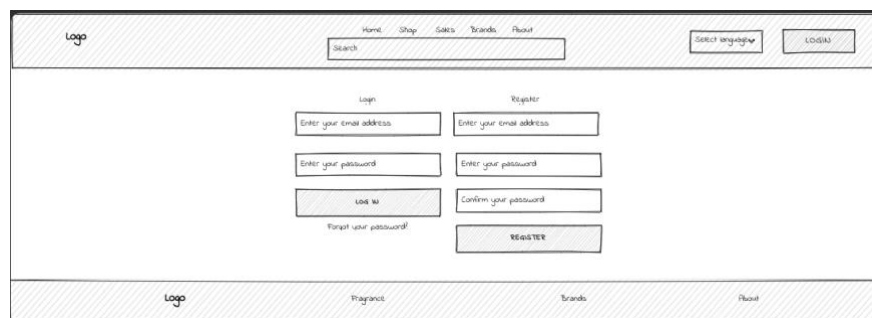


Figure 9. Wireframe of the Authentication page

The Cart page is one of the most important parts of an e-commerce website that allows users to manage their cart such as updating the quantity or deleting unwanted items as described in Figure 10. Then users can proceed to the checkout section which is handled by PayPal. It is available with different payment methods and supports testing accounts for real checkout on the website.

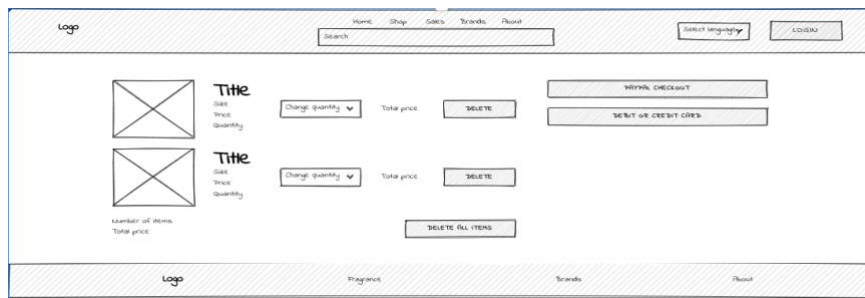


Figure 10. Wireframe of the Cart page

3 TECHNOLOGIES

3.1 Introduction

We need at least a Static Site Generator and a Headless CMS to start the development. There are a lot of choices such as Jekyll, Gatsby, and Nuxt for the first technology, and Strapi, Ghost, and Netlify for the latter one, NextJS, and Sanity.io were selected. Along with those, Tailwinds and React Query were chosen for styling and state management.

3.2 NextJS

NextJS is a framework built on top of React, providing all the features needed for building extremely fast static websites. Although fairly new technology, Next is widely used by large enterprises such as Netflix, Uber, and Starbucks. It works well with Headless CMSs or e-commerce platforms to deliver exceptional performance and SEO benefits.

3.2.1 NextJS vs ReactJS

The main difference between NextJS and React frameworks is the ability to build static websites with NextJS, meaning, Next provides React structures plus extra functionalities.

3.2.2 Features

The main reason why a lot of companies choose Next is the data fetching method. While React has only CSR, Next offers three additional ways to fetch data which are SSR, SSG, and ISR (Gamela and Rebelo, 2021). In CSR, HTML is generated by JS in the browser, so if JS is not enabled, users only see a blank page. The same happens when a Google bot crawls the website, HTML structures cannot be examined, which leads to bad SEO results.

SSR is a popular technique that renders SPA on the server and sends it to the client. This makes it possible to serve dynamic components as static HTML markup.

In SSG, the website is compiled and rendered at the build time, the result is a collection of static files that include the HTML file itself as well as assets such as JavaScript and CSS. The difference between those two is that SSG does not consume an API request, the data is static and only changes at build time, while SSR is dynamic.

ISR is a mix of SSG and SSR, it allows us to use SSG without rebuilding the whole site. With revalidate, the number of pages is chosen to generate and the stale time. Having less build time, the performance is extremely fast.

Generally, SSG is the fastest method, however, data will not be updated until the site is built again. In addition, rebuilding thousands of pages can be laborious, therefore, it is more suitable for blog sites, the category page. In contrast, if there are plenty of items and the information needs to be fresh on every render, for instance, on e-commerce websites, SSR is the best choice.

If we need SSR and SEO, Next is a better choice, especially when we are building blogs, news, or e-commerce websites. For applications that are heavily focused on the client-side, we can go with React.

3.2.3 Trends

Today, people are moving towards Next due to its wonderful features. According to NPM Trends, a site that records download statistics of JS frameworks, there has been a rise in Next since 2019. It is increasing rapidly and at the time of writing this thesis, the download numbers of Next are almost 13 times more than React's as shown in Figure 3.

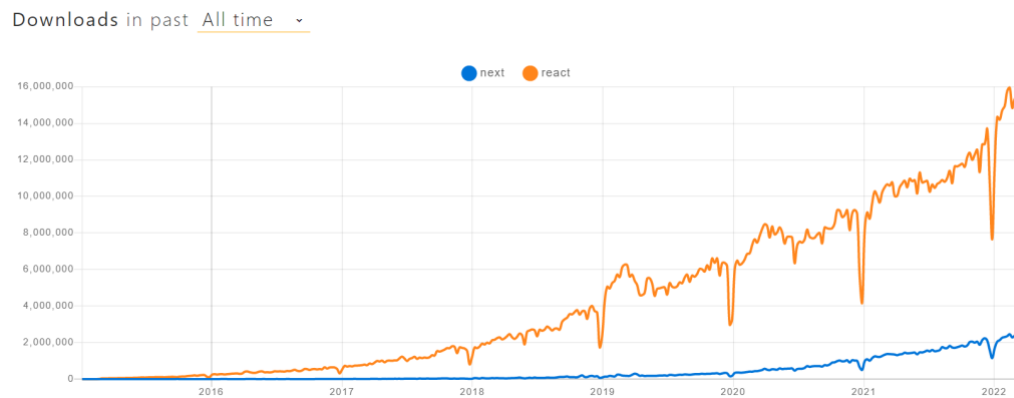


Figure 11. Download numbers of React and NextJS on NPM Trends

3.3 Headless CMS

3.3.1 Introduction

Headless CMS is a form of CMS that has a backend only, it provides accessibility through Rest API or GraphQL to be displayed to different devices. The term "headless" refers to the idea of cutting the frontend where the content is displayed off the backend where the content is stored (Czapla, 2022).

Headless CMS is designed to be frontend agnostic and API-driven. Developers can distribute content freely to their target audience using the frameworks and technologies of their choosing.

3.3.2 Headless CMS vs Traditional CMS

There are some advantages of Headless CMS over traditional CMS:

- **Flexibility:** The front and backend are completely independent of one another, organisations can freely choose any technologies that best fit their needs.
- **Cross-Platform Support:** A single content may be simply presented on a variety of web and mobile devices.

- **Development experience:** Headless CMS allows developers to pick programming languages of choice, which they are most familiar with. In addition, being API-driven, the interaction with the frontend side is done with ease, and developers can focus on how content is presented.
- **More secure:** Headless CMS is likely to be more secure because it simply gives content via API or GraphQL, whereas traditional CMS allows user content editors via some interface that is attached to the site, which is more vulnerable to attackers.

3.3.3 Sanity

There are nearly one hundred Headless CMS frameworks, the most common are Strapi, Ghost, and Netlify. In this project, Sanity.io was chosen for some reasons, one of which is stability. While most Headless CMSs require no coding, everything interacts via UI, Sanity needs to implement with codes. However, it is quite stable since codes are more accurate and transparent than the UI itself. It allows developers to control and debug functionalities easily.

Sanity also comes with already made functions for specific projects such as blogging and e-commerce. It is not necessary to do everything from the beginning, we can extend their examples with our new features.

The content editing feature works perfectly as others, in addition, every time users create or edit the content, it is saved to the version history, allowing users to undo or redo the exact version they want.

Graph-Relational Object Queries is a declarative language for querying collections of mostly schema-free JSON objects. It is the best feature of Sanity that makes it unique among other Headless CMSs. GROQ allows developers to do the same work as in GraphQL with less effort.

The main purpose of these query languages is to develop faster and more predictable APIs. To illustrate, there are two API endpoints for books and authors, in which, a book has one or more authors, and an author may write several books. It is not recommended to store all authors' data in books and vice versa, instead, we store the ID of related items. To display the information of a book with its author on the front, we need to make complex functions in the REST API which may include nested loops. GraphQL helps to do this efficiently. In addition, it allows us to get the exact data instead of getting the whole object, making the API cleaner and easier for debugging.

4 IMPLEMENTATION

4.1 Environment Setup

The programming was done in JavaScript with Visual Studio Code on macOS, it is also compatible with Linux or Windows. Sanity was used to build the backend, which needs Node.js 12.22.0 or later to run, and Sanity CLI was installed to create the CMS in the terminal.

In the frontend part, React and React DOM are required to run NextJS, including other third-party libraries:

- AWS Amplify handles Authentication such as registration and login.
- React PayPal makes it easy to integrate with PayPal checkout.
- Tailwinds takes care of the website layout.

4.2 Headless CMS

After having Sanity CLI installed on the computer, the following command was used to install Sanity in this project.

```
npx sanity@cli init
```

There are several options with already made codes such as Blog and E-commerce. However, the product structure of this application is different because it has a variety of sizes, so the blank project was chosen for easier customization.

When Sanity CLI finished running, the project backend was created with the structure as shown in APPENDIX 1.

The most important thing in Sanity is the schema that allows us to build the structure of contents and manage their relationship. To be more specific, there are brands, brand details, product details, and users.

```
fields: [  
  {
```

```

    name: 'title',
    title: 'Title',
    type: 'string',
  },
  {
    name: 'slug',
    title: 'Slug',
    type: 'slug',
    options: {
      source: 'title',
      maxLength: 96,
    },
  },
  ...
  {
    name: 'vendor',
    title: 'Vendor',
    type: 'reference',
    to: { type: 'vendor' },
  },
  {
    name: 'images',
    title: 'Images',
    type: 'array',
    of: [
      {
        type: 'image',
        options: {
          hotspot: true,
        },
      },
    ],
  },
]

```

Code snippet 1. Product item schema

Code snippet 1 is the schema code of product items. There are three attributes: name, title, and type. The name is the ID of the unique field, the title is displayed on the UI and the type is the data type of the field. The schema types of Sanity include slug, array, block, string, and number, in which, the slug indicates the URL on the browser while the rest are the content of the field. Furthermore, the type of reference is used to define a relationship between two data. This code snippet connects the product item to its vendor (brand).

After configuring all the schemas, running the backend to open a dashboard as in Figure 4, in which information such as brand, product, and product details can be added by content writers.

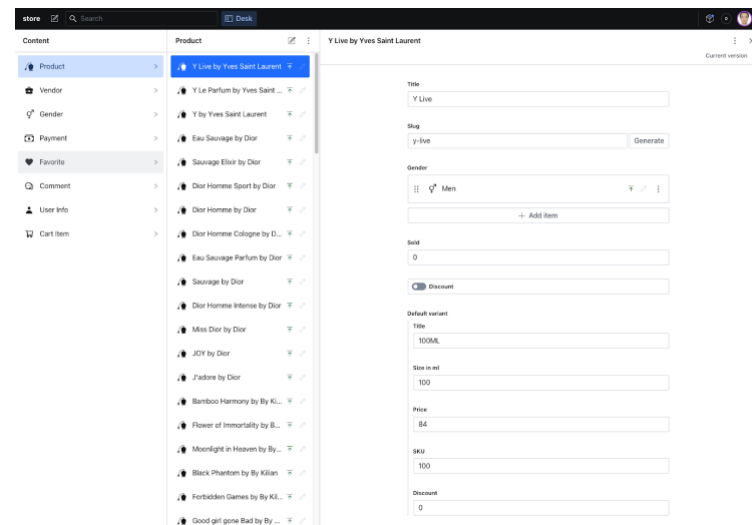


Figure 12. User Interface of Sanity dashboard

Sanity provides a playground to test queries and APIs given in Figure 5, those queries will be used in the frontend part to display data from the CMS.

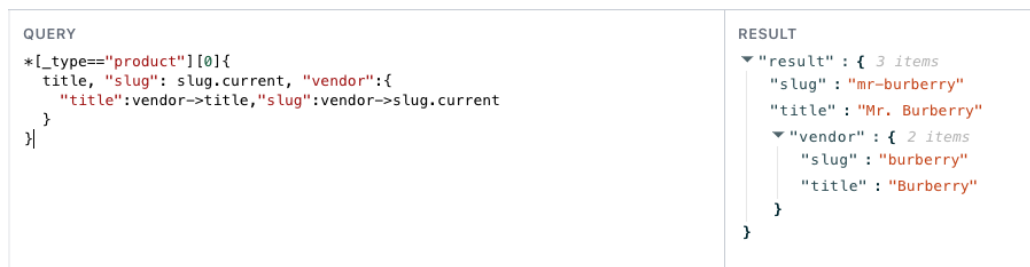


Figure 13. Vendor information is connected to the product via type reference

4.3 Frontend

The fastest way to install NextJS with Typescript for frontend is to use

```
npx create-next-app app-name @typescript
```

The APPENDIX 2 displays NextJS project structure, in which, the most important file is `_app.tsx` located in `pages` folder. It is the root file of the whole project; all the configurations and libraries will be placed here in the Code snippet 2.

```

<QueryClientProvider client={queryClient}>
  <ReactQueryDevtools initialIsOpen={false} />

```



```

<Hydrate state={pageProps.dehydratedState}>
  <CartProvider>
    <Layout>
      <NextNProgress
        color='#29D'
        startPosition={0.3}
        height={5}
        showOnShallow={false}
      />
      <Component {...pageProps} />
    </Layout>
  </CartProvider>
</Hydrate>
</QueryClientProvider>

```

Code snippet 2. Page initialization in _app.tsx

NextJS uses the App component to control page initialization. In this code block, two libraries are used, in which React Query helps to manage and cache server APIs while NextNProgress shows the loading bar when navigating between pages.

```

const Layout: React.FC = ({ children }) => {
  return (
    <div className='main'>
      <Header />
      {children}
      <Footer />
    </div>
  )
}

```

Code snippet 3. The Layout component contains Header and Footer

The layout is a component created to reuse shared components. In Code snippet 3 above, Header and Footer will be displayed on every page, it is not advisable to duplicate them to all the files.

```

<Authenticator variation='default'>
  {{{ signOut, user }: AmplifyType) => (
    ...

```

```
    )}
  </Authenticator>
```

Code snippet 4. Adding AWS Amplify to Profile page

After installing Amplify, login and registration functions are ready to use as illustrated in the Code snippet 4. Anything put in the middle of Authenticator component is called a private route, which is hidden when the user is not authenticated and only displayed after the user has logged in.

```
export const getStaticProps: GetStaticProps = async () => {
  const queryClient = new QueryClient()
  await queryClient.prefetchQuery('all_brands', getAllBrands)
  return { props: { dehydratedState: dehydrate(queryClient) } }
}
```

Code snippet 5. Fetching with the SSG method

The Code snippet 5 describes the SSG fetching method, in which the data is fetched only once during build-time and served as HTML later. It does not consume API requests and only changes if NextJS is built again. This method is suitable for data that does not change often, in this case, product brands are likely to stay the same.

```
export const getServerSideProps: GetServerSideProps = async ({ query }) => {
  await queryClient.prefetchQuery(['product_detail: ' + slug], () =>
    getProductDetail(brand_slug, slug),
  )
  return {
    props: {
      dehydratedState: dehydrate(queryClient),
    },
  }
}
```

Code snippet 6. Fetching with the SSR method

On the other hand, the price, and the quantity of the product change frequently, therefore, it is better to use SSR as in the Code snippet 6.

5 TESTING AND RESULTS

5.1 Testing with Cypress

Testing plays an essential role in software development since it ensures the quality of the product before delivering it to customers. Cypress was used in this project for end-to-end testing. Because it operates in browsers, it allows accurate testing, whereas other tools, such as Jest and Selenium, run outside of the browser.

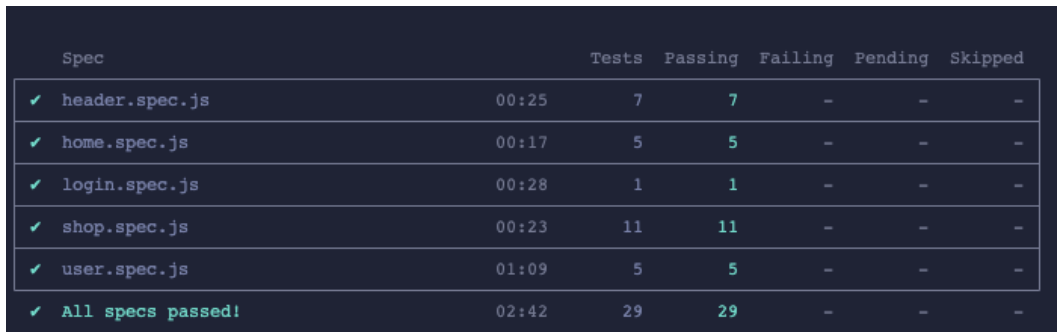
The test cases were written based on the functionalities of the website.

The navigation links on the Header component function properly.

The Home page displays the correct components.

The Shop page displays only the products that match the filter.

Users can log in to manage their favorite items and make purchases.



Spec	Tests	Passing	Failing	Pending	Skipped
✓ header.spec.js	00:25	7	7	-	-
✓ home.spec.js	00:17	5	5	-	-
✓ login.spec.js	00:28	1	1	-	-
✓ shop.spec.js	00:23	11	11	-	-
✓ user.spec.js	01:09	5	5	-	-
✓ All specs passed!	02:42	29	29	-	-

Figure 14. Test results in terminal

When the testing process is finished, a result is exported to the terminal as in Figure 6.

5.2 Results in Browser

Finally, the application built with Jamstack technologies was successfully constructed. Figure 7 shows the Home page, allowing users to view products in

different sections, selecting them to view their details. On the top of the page, the Header displays navigation that links to other pages.

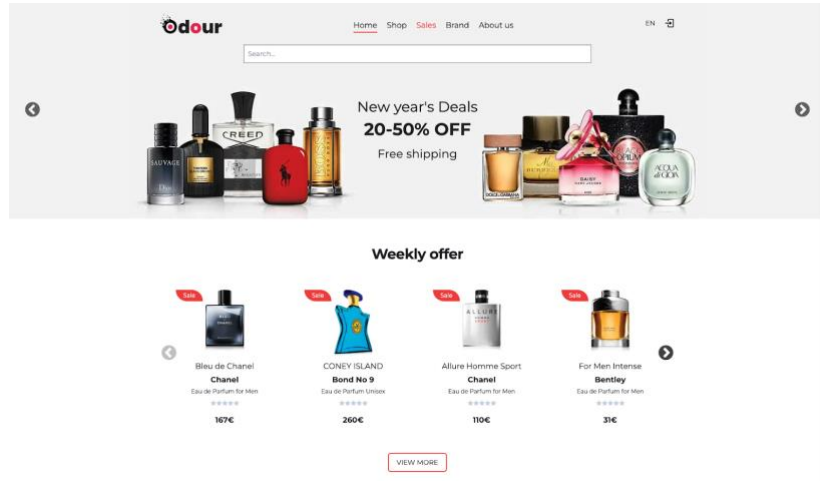


Figure 15. Home page

When customers log in, they can manage their favorite products and view their previous purchases as seen in Figure 8.

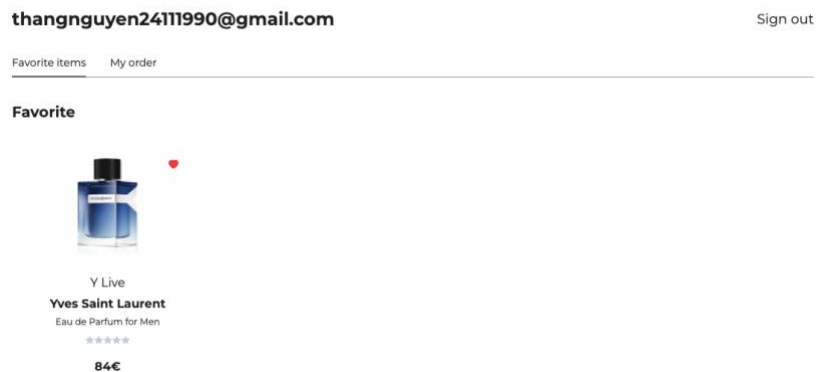


Figure 16. Login successfully

Figure 9 demonstrates the checkout section using the PayPal testing account. If the payment is successful, users are redirected to the Success page and the order information is saved into the CMS.

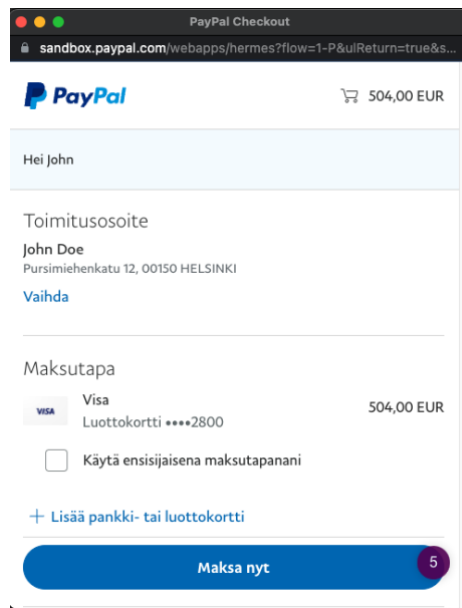


Figure 17. Checkout with PayPal testing account

6 CONCLUSIONS

This thesis fulfills all the requirements needed for an e-commerce website, including viewing products, registration and login, checkout, and reviewing order history. It is also good in SEO and loads extremely fast which increases user experience, resulting in better sales. Generally, the website works as expected since it meets all the specifications.

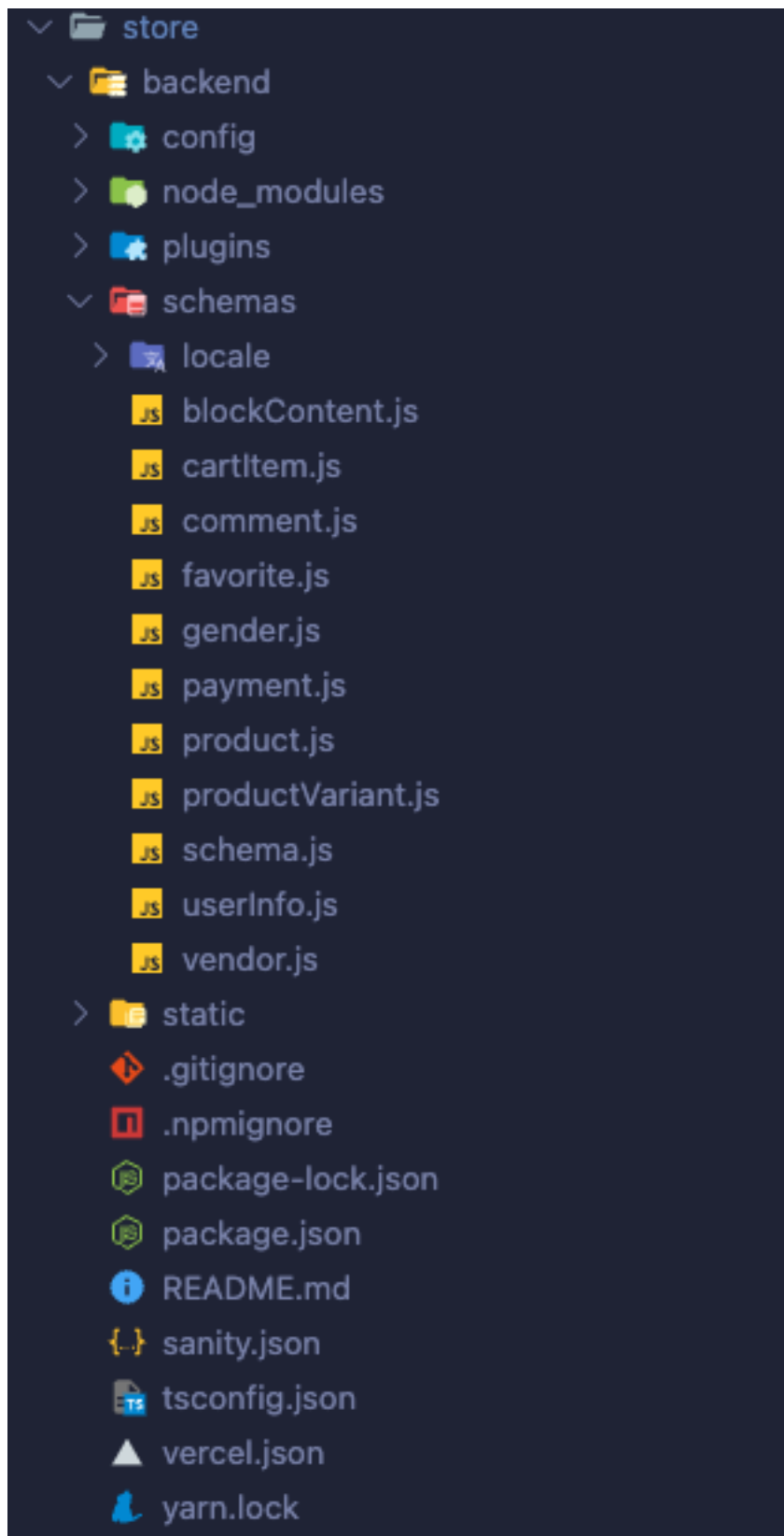
Two problems occurred during the process. Due to a large number of frameworks and libraries, choosing one to use was quite difficult. It requires a lot of research to pick the most suitable ones. In addition, the stack is relatively new, and searching for information and solutions on the internet was hard.

Overall, Jamstack is a good solution that provides many benefits to web development. It remains to be seen how well it takes on in the future. Jamstack is more suitable for frontend or frontend focused full-stack developers and SaaS applications such as news and e-commerce. Though the technology is still in the early phases of adaptation and there is no evidence that it will be the future of web development. Based on this experience, it can be said that Jamstack plays an important role in web development.

REFERENCES

- Cortez, K, n.d. The disadvantages of WordPress. Accessed 28.1.2022.
<https://doctorlogic.com/blog/disadvantages-of-wordpress.html>.
- Czapla, A, 2022. Headless CMS vs Traditional CMS. Accessed 5.2.202.
<https://blog.vuestorefront.io/headless-vs-traditional-cms>.
- Dewhurst, R, and Chamberland, C, 2021. WordPress Security Report. Accessed 28.1.2022.
- Gamela, A and Rebelo, G, 2021. Next.js vs React: What are the differences? Accessed 11.2.2022. <https://www.imaginarycloud.com/blog/next-js-vs-react>.
- Kirkpatrick, D, 2016. Google: 53% of mobile users abandon sites that take over 3 seconds to load. Accessed 2.2.2022.
<https://www.marketingdive.com/news/google-53-of-mobile-users-abandon-sites-that-take-over-3-seconds-to-load>.
- Morey, R, 2018. How Much Disk Space Does a WordPress Website Need? Accessed 28.2.2022. <https://wp-rocket.me/blog/how-much-disk-space-wordpress-needs>.
- Southern, M, 2022. Page Speed and SEO. Accessed 28.2.2022.
<https://backlinko.com/hub/seo/pagespeed>.
- Wlosik, M, 2021. Jamstack for eCommerce at Scale. Accessed 10.2.2022.
<https://www.layer0.co/post/jamstack-ecommerce-at-scale>.

APPENDIX 1. THE FOLDER STRUCTURE OF SANITY



APPENDIX 2. THE FOLDER STRUCTURE OF NEXTJS

