Bang Nguyen

Improving web development process of MERN stack

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

10 May 2021

Author Title Number of Pages Date	Bang Nguyen Improving web development process of MERN stack 52 pages 10 May 2021
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Mobile Solutions
Instructors	Kari Salo, Senior Lecturer

The objective of this thesis was to discuss and improve the web development process of the MERN stack. The web application should have the Admin dashboard for content management and the client-side for normal users, which is similar to the common CMS such as WordPress, Drupal, or Joomla.

The frontend will be built using React; in this thesis, the author decided to use Next.js, which is the React framework that provides server-side rendering, routing, and tooling. Besides Next.js, the application also implemented Apollo to manage data such as GraphQL mutation, queries, and data caching.

For the backend, the author used KeystoneJS is a headless CMS framework to build server applications that connect with a database. KeystoneJS is based on Express for Node.js and uses MongoDB as its default database. Unlike WordPress, where the website can be set up by non-technical users, KeystoneJS requires more technical knowledge to develop a website, and the developer can customize everything to build a specialized application system and APIs

Keywords	React, Nextjs, GraphQL, Nodejs, Apollo, KeystoneJS, CMS, MERN, LAMP, JavaScript, PHP

Contents

List of Abbreviations

SPA	Single page application
MPA	Multi-pages application
JSON	JavaScript Object Notation
MVVM	Model-View-ViewModel
DOM	Document Object Model
VDOM	Virtual DOM
HTML	HyperText Markup Language
CRUD	Create-Read-Update-Delete
API	Application Programming Interface
MERN	MongoDB, Express, React, Nodejs
CMS	Content management system
PHP	Hypertext Preprocessor
API	Application programming interface
SEO	Search Engine Optimization
UX	User Experience
UI	User Interface
GUI	Graphical User Interface

1 Introduction

Often, startup founders think that their product will uniquely provide the best solution for their target market’s problems and undoubtedly turn into profit. This thinking has two assumptions: value hypothesis and growth hypothesis. Therefore, to validate these two hypotheses, the startup needs to provide a version of its product, which can demonstrate the idea and core values of the business – minimum viable product (MVP) [1].

In the last decade, developers used server-rendering technology to build their web applications (Multi-pages application). Businesses often used WordPress, Drupal, or other CMS platforms to build their website. The main advantage of using those CMS was the building speed with minimal coding which was provided by a lot of pre-built templates, themes, and plugins. However, that was also its biggest drawback since their website depended on those plugins, so the upgrade and maintenance become tremendously tedious.

Over the years, with the growth in complexity level of the web application, the pre-made templates have become insufficient. Because of that, programmers begin to use a stack of various technologies in web development, and JavaScript becomes the main language for frontend and backend. One of the most popular and robust modern stacks is the MERN stack. [2]

In this thesis, the author tried to provide a modern way to build a web application using the MERN stack. For more specific, the author wanted to develop a single-page web application that has an Admin dashboard for content management and the UI for normal users. Instead of using the common CMS such as WordPress, Drupal, or Joomla, the author wanted to use the MERN stack while still have the benefits of the traditional CMS. For the scope of the thesis, the author mainly focused on discussing the common CMS, the changes in user behavior, and the benefits and drawbacks of the traditional and modern web development design pattern, language, and development stack. Besides the technical discussion, the author provided a summary of setting up the GraphQL server with Keystone and some frontend demonstrations to prove the benefit of this new improvement.

From there, the author will provide his solution, a different approach to use MERN stack in web development, which solves the drawbacks but still keep benefits of modern single page application. He uses Next.js for the frontend development that provides SEO support but still keeps the performance speed. For the backend development, he uses KeystoneJS, which is an Express headless CMS framework. KeystoneJS will take care of most of the heavy works in the backend development and provide an Admin dashboard for content management and GraphQL API for data fetching and mutation from the frontend.

In the next section, the author will explain all the technical definitions and programming concepts used in the project.

2 Multi Pages Application and Single Page Application

At this moment, there are millions of web applications that help companies deliver their goods and services, connect with customers, and increase user engagement. As they provide higher scalability, improved flexibility, and simplified maintenance compared to desktop solutions, now they are much more popular.

In this section, the author will have an overview of the two main streams of developing web applications: Multi-page application and Single page application.

2.1 Multi-pages application (MPA)

At the beginning of developing a web application, the web application was designed based on client/server pattern, in which the browser from client only designed to display the data and information is received from the server and send the user data back. The biggest advantage of the client/server approach is, in fact, the client does not depend on the user’s specific operation system; hence, the web application has become a popular cross-platform service. [3] See Figure 1 below.

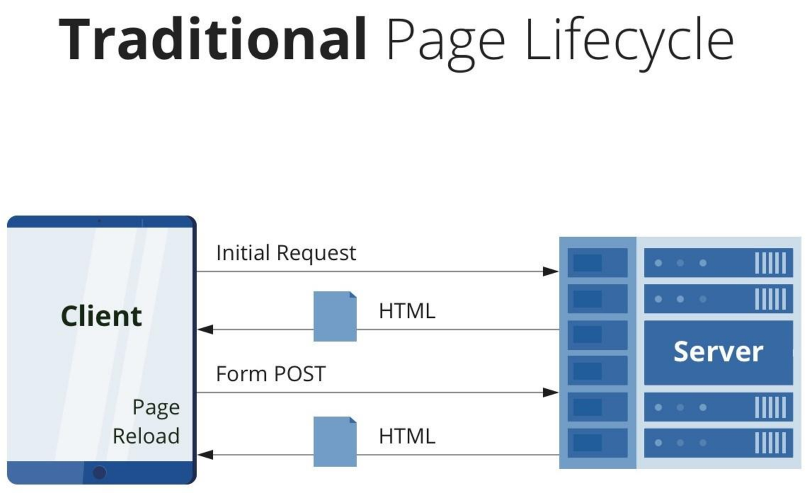


Figure 1. MPA lifecycle [3]

Moreover, the MPAs are more optimized for Search Engine Optimization (SEO). Since the MPAs are server-side rendering applications, search engines such as Google, Bing, and Ecosia could easily get metadata and keywords from them. Furthermore, there were many data analytics tools for the MPAs to help get an insight into the customer's behavior, system functioning, etc. The examples of the MPAs are most e-commerce websites such as Amazon, eBay, and other e-learning webs like Udemy. [3]

However, the downside of the client/server pattern is the loading time. According to the analysis of Michael s. Mikowski, when his book was published, roughly thirty-five-million person minutes were wasted waiting for website pages to finish loaded globally [4]. One of the reasons for the slowness is because the server has to do all the works from data operations such as writing to the database, calculations, transform the data to a readable format, render the static page, and then serve the final result to clients. Figure 1 illustrates another flaw of the MPA, which is its life cycle. Because the MPA focuses on serving the static webpage to the client, the application needs to reload every time it receives a request from the client, even though only a small part of it needs to update, thus, increase the workload on the server.

Besides the slowness, sometimes the process might have some faults because of the Internet connection, which cause the missing of some component and functionalities, or worst, failing to load the entire web page. This sluggish, uncertain, and clunky user experience (UX) is unacceptable; therefore, the developer has been trying to create a new approach that can solve all these problems. One of the popular approaches in modern web development is Single Page Application. [4]

2.2 Single page application (SPA)

“Today's consumers demand a fast, engaging, and secure online shopping environment when searching for a product online. We see a direct relationship between online revenues and site performance and therefore, we have to ensure our site performs well and loads fast," said Michael Cooper, Vice President and General Manager, HomeDepot.com [5]

With the continual advancements in technology, the web application has undergone significant change. In the early 90s, the improvement of hardware technology had made the computer become stronger and faster, thus, people started having the idea about the hybrid between the desktop application and the server/client approach. Hence, the Single page application was born.

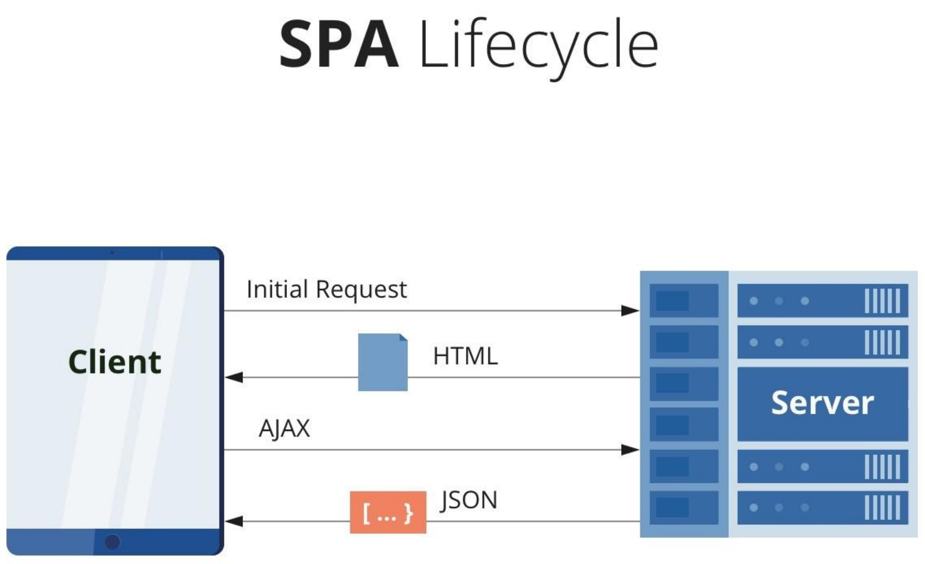


Figure 2. SPA Lifecycle [3]

In contrast to the traditional approach, where the server serves the whole static web page to be displayed on the client, the SPA lifecycle illustrated in figure 2 shows that the client only needs the data (commonly in JSON format) from the server to render the webpage in the browser. It does not require any content reloads; in fact, the loaded content is based on the user’s interaction. Therefore, users do not need to wait for too long to access the web page since the data is loaded automatically. The client in this approach does more than displaying the pre-build web page; moreover, it performs most of the processing and transforming the data fetched from the server into the web view to be rendered.

One of the main advantages of SPA is the content load speed, since the UI is rendered from the client, the lifecycle of the data transfer between server and browser is minimal. With the correct setup, the client only gets the data for the current view, which further enhances loading speed several times faster than traditional MPA. [3]

Moreover, developing SPA is easier, thus, it costs less time, money, and resources to build and release a good quality SPA. Lastly, the SPA also could work offline since most of the data can be cached in the browser and since they do not need to get the data from the server, the user can still interact with the application to a certain extent and the application can synchronize with the server once the connection is restored. [3] See Figure 3 below.

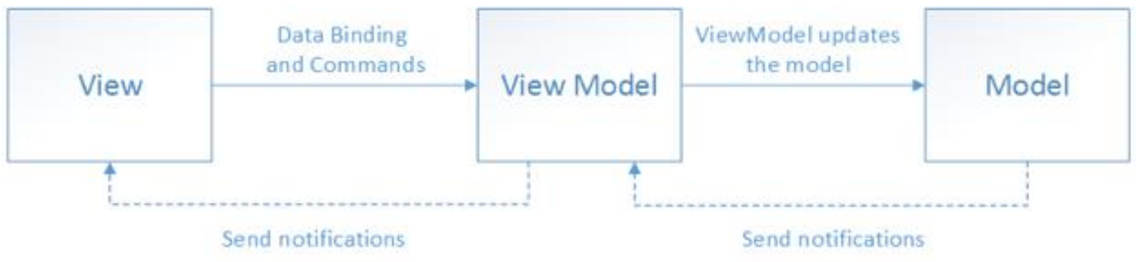


Figure 3. The MVVM pattern [6]

With the advancement of the technology, the SPA becomes more complex in size and scope, which raises potential issues about maintenance and the cost of UI modifications and testing frontend. To resolve these issues, back in the early 2000s, Martin Fowler had published an article that introduces a pattern call the Presentation Model (PM), in which a view was created by an abstraction called PM. Later, John Gossman, one of the architects of WPF and Microsoft Silverlight, has developed Model View View-Model (MVVM) pattern, which was similar to PM presented by Fowler. This pattern helps separate the business logic and the application's UI, having opened a door to the new era for the web application. [6]

3 JavaScript and Hypertext Preprocessor (PHP)

The community often assumes that JavaScript is for the frontend development while PHP is for the backend. However, that might be true in the past, but with the advancement of technology, that statement is only partially correct. In this section, the author will point out PHP and JavaScript main differences, their strong points, and use cases.

3.1 PHP: Hypertext Preprocessor

PHP is the recursive acronym for Hypertext Preprocessor. PHP is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. According to the W3Tech survey, 79% of all websites use PHP [7]. Among the most popular ones, there are Facebook, Wikipedia, and WordPress. PHP was designed by Rasmus Lerdorf in 1994 and had been maintaining and improving by The PHP Development Team. They claimed that PHP is fast, flexible, and pragmatic; PHP powers everything from a simple blog to the most popular websites in the world. [8] That might be correct in the past, but nowadays, the popularity of PHP had been declined over time.

3.2 JS: JavaScript

JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object orientation, and first-class functions. According to Statista, 69 % of developers worldwide use JavaScript, and 5 % more plan to adopt this language. The report shows that it is the most popular programming language in the world as of late 2019 [9]. Along with HTML and CSS, JavaScript has become one of the core languages in web development. JavaScript was initially designed as a frontend development language. However, the introduction of Node.js in 2019, which is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser, had enabled JavaScript ability to develop both frontend and backend. [10]

3.3 Differences between JavaScript and PHP

In this section, the author compared the characteristics of JavaScript and PHP in different aspects: [11]

3.3.1 Performance and speed

JavaScript:

JavaScript is characterized by an event-driven, single-threaded, non-blocking I/O execution model. JavaScript is asynchronous, which means it can run through the entire code simultaneously without waiting for some functions to be executed. Node.js is sped up even further by the V8 engine, constant server connection, and callback functions. Therefore, JavaScript had become the best solution for modern low latency applications such as streaming platforms.

PHP

PHP is characterized by a multi-threaded, blocking I/O execution model. PHP is synchronous, which means the second line of code in PHP cannot be executed until the first one is, which makes it much slower than JavaScript.

3.3.2 Extensibility

JavaScript

JavaScript can be combined with HTML, XML, and Ajax. For web development, some of the most popular front-end JS technologies are Vue, Angular, React, and the upcoming Svelte. For the server-side, JavaScript uses Node.js and often using Express as the framework to develop the application's backend.

Node.js uses Node Package Manager (NPM), which is the largest software registry for package management.

PHP

PHP can only be combined with HTML. The greatest advantage of PHP is the availability of CMSs like WordPress or Drupal. These solutions may greatly facilitate and even make web development cheaper. PHP can also be extended with any LAMP stack technology and such server solutions as MySQL or PostgreSQL.

There are two package managers for PHP – PEAR, and Composer. PEAR (PHP Extension and Application Repository) is a structured library of open-source PHP code. Composer is a dependency management tool for PHP.

3.3.3 Universality

JavaScript is cross-platform, and so is PHP. Both PHP and JavaScript are primarily aimed at developing web applications, even though both can be used for mobile app development.

JavaScript

JavaScript can be used for both frontend and backend development. Combining with other frameworks and libraries, users can develop web applications and mobile solutions using only JavaScript.

PHP

Unlike JavaScript, PHP is a backend development language. PHP cannot be used to develop a web application separately, it is often used within a stack, one of the most popular stacks is the LAMP stack, which stands for Linux, Apache, MySQL, and PHP/Perl/Python. The author explained the LAMP stack in more detail in section 4.2.

3.3.4 Popularity

JavaScript

Most JS frameworks are open source, but not JavaScript itself.

JavaScript is the most popular language on GitHub with over a 20% share of pull requests [12]. JavaScript is used by Facebook, Netflix, LinkedIn, Trello, Uber, Airbnb, Instagram, eBay, NASA, and Medium.

PHP

PHP code is open source, which makes it more flexible and customizable.

PHP takes only eighth place with about 5 % of pull requests [12]. The most beloved advantages of PHP are a large community, open-source, and simple deployment. PHP is used by Facebook, Lyft, Wikipedia, Slack, Tumblr, and 9 GAG.

3.3.5 Debug and Maintainability

Web development is a challenging endeavor, and this challenge is only increasing as the web matures. What might have been considered a passable website ten years ago would seem starkly outdated to the user of today, and modern businesses are diverting more and more resources towards their online presence. The complexity of both front-end and back-end web development has risen to meet these standards, and the software writers of today are responsible for managing this complexity. Moreover, although various frameworks and tools have emerged to make this process easier, much of this cognitive load still falls squarely on developers. [13]

JavaScript

With the increase in popularity between the new generation of developers, JavaScript had improved itself with rapid changes and new features. However, that is also one of the main disadvantages of JavaScript when it comes to maintainability.

Across ES5, ES6, and ES7, JavaScript kept adding more features and syntax such as const, let, destructuring, concise object assignment, and symbols to name a few, this brought both good and bad issues. With these new features, JavaScript code becomes more secure, less bug, and improve readability. However, it also introduced some problems in its maintainability.

One of the maintainability criteria is consistent in writing code, however, JavaScript had provided many ways to do the same task, which makes the JavaScript code inconsistent, opinionated, and confused to others. To fix this problem, the JavaScript developer team uses some tools such as Eslint and Prettier, and set up rules to ensure consistency and improve the readability of the codebase.

The rapid development of JavaScript and its added new syntaxes had made some of them were not available or not working properly in some browsers such as Internet Explorer or Safari. Because of that, the developer had to use Babel to compile the latest JavaScript syntax to older JavaScript syntaxes.

Another problem in JavaScript maintainability is the type system in JavaScript. Firstly, JavaScript uses dynamic type, which means it allows the developer to re-assign different data type to the existed variable. This often leads to the bug where the function not working correctly since it receives the wrong data type and other developers have to waste a lot of time debugging the error. To prevent this, JavaScript had introduced the “const” keyword, in short, if a variable was defined using “const”, it cannot change its value later.

With the release of TypeScript in 2012, which is designed and maintained by Microsoft to allow the developer to be able to write typed JavaScript. TypeScript has improved JavaScript maintainability and provided solutions to many JavaScript flaws such as type system, which make JavaScript code become clear and concise custom type definition, no need for Babel to compile latest JavaScript syntax.

PHP

PHP is a powerful language; however, one of its drawbacks is maintainability.

“PHP will remain popular but its growth will slow, as people get nervous about its maintainability and security stories”

Tim Bray – 2008 [14]

According to Rick Grehan, debugging a PHP application is often more difficult than debugging an application in some other programming language such as JavaScript. There are two reasons, he added, in his article on InfoWorld. [15]

First, a PHP application rarely consists of only PHP code. The application is invariably a mixture of PHP, HTML, CSS, and JavaScript. HTML, CSS, and JavaScript run in the browser, PHP executes in the Web application server, and SQL executes in the database server. In PHP, however, there is a tendency to mix HTML and language syntax inside HTML files, which quickly results in poorly maintainable code where views and business logic are not separated. As a result, it is hard to extend PHP applications with new functionality and manage applications with a large codebase.

Second, a PHP Web application does not have a continuous, linear flow of execution. Its execution is a series of HTTP request/response transactions. Each transaction is independent of all other preceding and following request/response pairs. The problem arises when the developer starts a debug session on file X.PHP, when X.PHP sends its HTTP response, that PHP code is completed. A subsequent HTTP request in the same user session might trigger the execution of file Y.PHP. Consequently, a single user session effectively passes through multiple debug sessions.

Similar to JavaScript, PHP is also a dynamically typed language, which gets the same problem as JavaScript. Furthermore, there are some other reasons which make PHP lose its popularity such as the lack of namespaces, most of the PHP codebase was in bad shape, and mixing PHP code with HTML content. [16]

3.3.6 Suitable application

JavaScript

JavaScript has dedicated server hosting, which makes it perfectly suitable for large projects. It can be used to develop both the frontend and backend of almost any software application type, including 3D games, AR/VR solutions, and IoT products.

PHP

PHP is a general-purpose programming language, it is primarily used for developing dynamic web pages. Considering the availability of PHP-based content management systems such as Moodle and WordPress, PHP is the best solution for blogs, learning management systems, and e-commerce websites.

4 Content Management System

In this section, the author first gives an overview of the Content Management System (CMS) web such as WordPress, Joomla, and Drupal.

Content Management System (CMS) is a server program where the web content was stored in the database, and the web page would be built on demand when the client browser requested it. Moreover, CMS is a dynamic system, which means it could be extended and added new features. There are many CMS available in the market such as Joomla, Drupal, and WordPress. All those three CMS are free and have a lot of features and pre-made plugins, extensions, and templates to help create a website. Therefore, CMS was a preferred option for people who want to have easy content changes, simplify the management of a large amount of content, and be able to create a highly interactive website in a short amount of time like an online store, portal, and blog without technical knowledge. [17]

However, as the application grows, it requires updating and maintenance for those plugins and extensions, thus, the maintenance fee increased exponentially. Because of that, as illustrated in Figure 4, the interest for the traditional CMS had declined over time, especially since WordPress had dropped around 75 percentages of its users' interest since its peak in August 2014. See Figure 4 below.

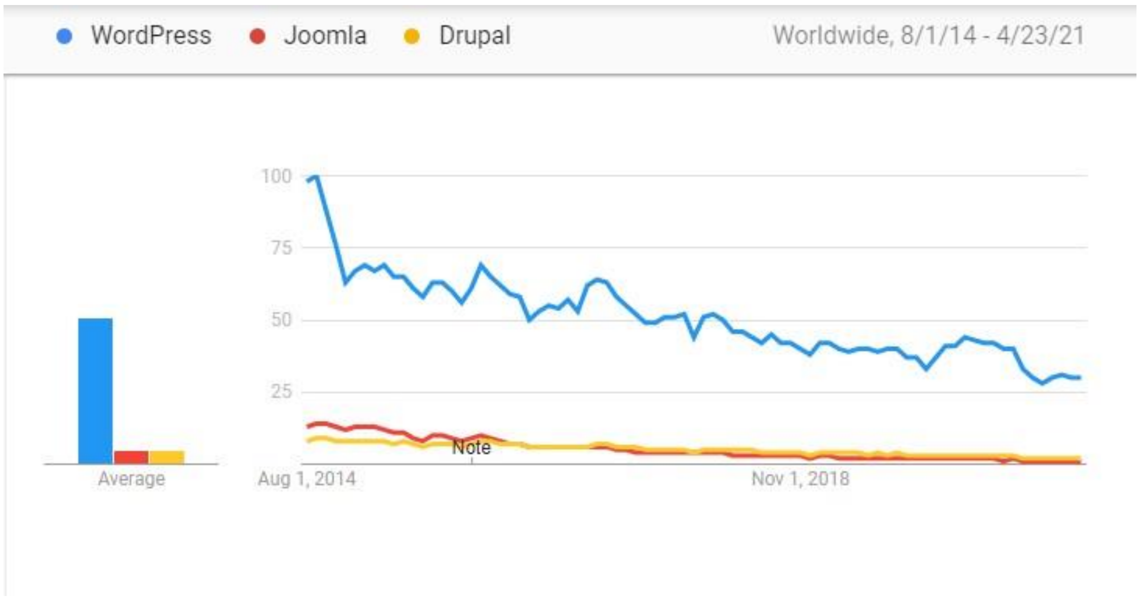


Figure 4. Interest over time from August 2014 till April 2021 (source: Google Trend)

Secondly, user behavior had changed over the years, based on the studies conducted by Akamai in 2006 and 2009 [5], consumers had become more impatient when pages take longer than two seconds to load. The study in 2009 pointed out that almost fifty percent of visitors expected a visual of web page content within two seconds compared to four-second loading time in 2006.

Furthermore, Figure 5 had pointed out that shoppers often become distracted when they were waiting for a page to load; they started to browse other online stores, which results in losing shopper loyalty, especially high-spending shoppers. Since these CMSs were designed as MPA, using server-side rendering architecture to display a web page, it needs to build a whole web page again whenever the user navigates to a new route. Together with an enormous number of plugins and extensions, plus server-side rendering, which ultimately prolongs the loading time and requires more waiting time when it becomes an unbearable user experience. In his article Single Page Application: Trend or Future, Vesic, and his co-author had agreed that SPA is a modern approach to application development. [18] Since the MERN stack is used to develop SPA, it inherits all SPA strong points and has proven to be a better application built by traditional CMS in terms of speed and performance. See Figure 5 below.

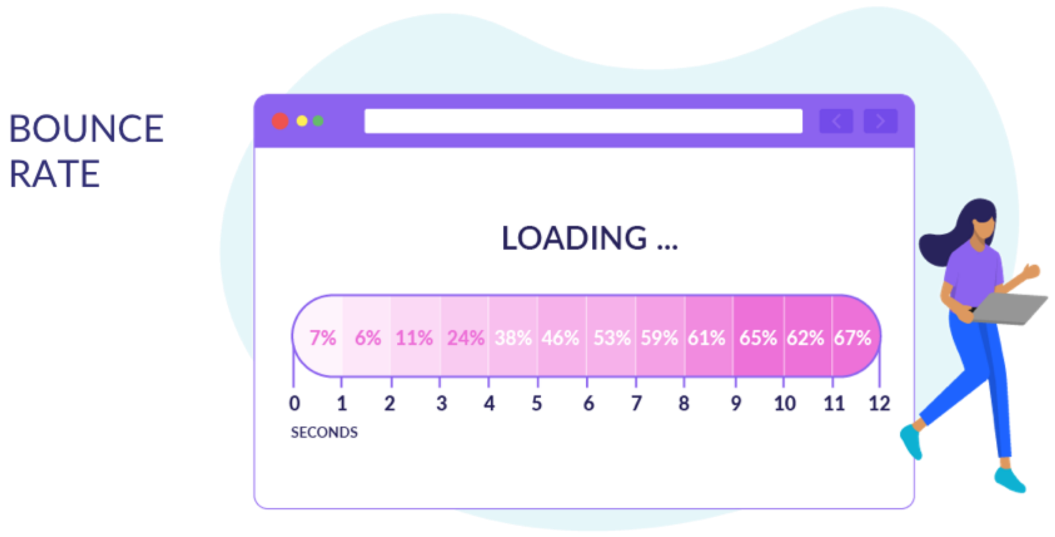


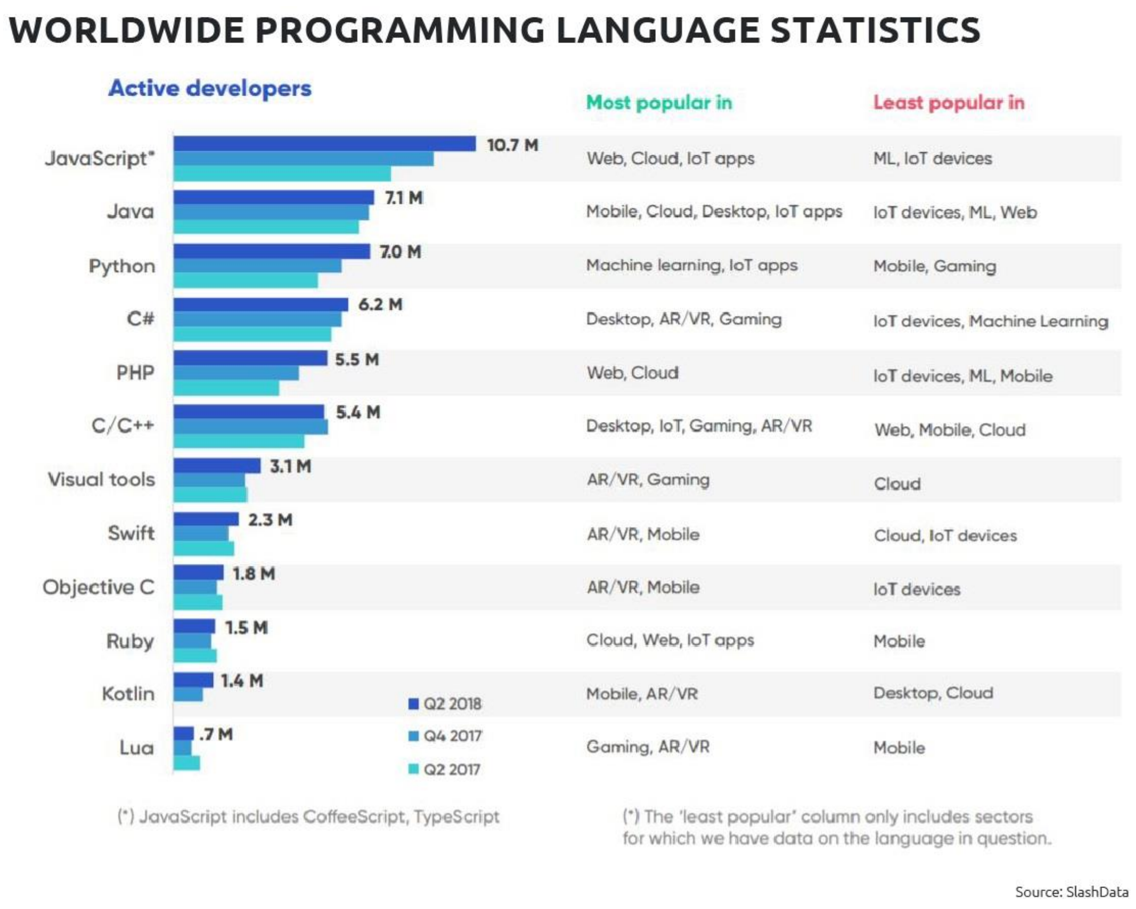
Figure 5. Bounce rate by web page loading time [19]

Finally, the interest in the PHP language had been declined, in contrast to the rise of JavaScript as shown in Figure 6. Since JavaScript is a full-stack development language, it is more universal than PHP. Thus, MERN could provide faster development and quickly deploy products to market as the challenge of understanding different languages to develop both frontend and backend is nonexistent. Therefore, MERN helps cut down the development cost and have better efficiency. The reason for the shift was driven by these factors:

- The presence of Node.js enables JavaScript to be able to programing server had increased demand for JavaScript developers since they can develop both frontend and backend.

- JavaScript provides better solutions for business

- The growing popularity of data-intensive real-time IoT devices and application that developed using JavaScript.



	Active developers	Most popular in	Least popular in
JavaScript*	10.7 M	Web, Cloud, IoT apps	ML, IoT devices
Java	7.1 M	Mobile, Cloud, Desktop, IoT apps	IoT devices, ML, Web
Python	7.0 M	Machine learning, IoT apps	Mobile, Gaming
C#	6.2 M	Desktop, AR/VR, Gaming	IoT devices, Machine Learning
PHP	5.5 M	Web, Cloud	IoT devices, ML, Mobile
C/C++	5.4 M	Desktop, IoT, Gaming, AR/VR	Web, Mobile, Cloud
Visual tools	3.1 M	AR/VR, Gaming	Cloud
Swift	2.3 M	AR/VR, Mobile	Cloud, IoT devices
Objective C	1.8 M	AR/VR, Mobile	IoT devices
Ruby	1.5 M	Cloud, Web, IoT apps	Mobile
Kotlin	1.4 M	Mobile, AR/VR	Desktop, Cloud
Lua	.7 M	Gaming, AR/VR	Mobile

(*) JavaScript includes CoffeeScript, TypeScript

(*) The 'least popular' column only includes sectors for which we have data on the language in question.

Figure 6. Worldwide programming language statistic [20]

In this section, the author had shown the decline of the traditional CMS and the increase in popularity of JavaScript. Since the traditional CMS targets non-technical users and help them build their website, the MERN stack aims to provide a complete set of tech stacks to the developer for building a full-stack web application. In the next section, the author will try to compare the MERN stack and the traditional LAMP stack.

5 LAMP and MERN

A stack is a combination of technologies such as frameworks, libraries, databases, and web servers that are used to develop a web application. In the second section, the author had introduced two main streams of developing web applications: Multi-page application and Single-page application, which also represented traditional development and modern development respectively. In this section, the author will compare once again two representatives of server-side rendering, which is a LAMP stack, and client-side rendering, which is a MERN stack.

5.1 LAMP

LAMP is an acronym of the following components:

- Linux: Linux is the LAMP stack’s foundation layer, and the rest of the three layers run on top of this layer. Linux was introduced in 1991. It is an open-source operating system. Many industries use Linux due to its flexibility.

- Apache: Apache Server is the second layer of the LAMP stack. Apache HTTP web server came into existence in 1995. Apache is also free and open-source HTTP web server software.

- MySQL: MySQL is the third layer of the LAMP stack, a relational database system used to store the application data. MySQL database is also an open-source database. MySQL works well in every project from small to large.

- PHP: PHP is the fourth and final layer of the LAMP stack, and it resides on the top of all the layers. Websites and web applications run on the PHP layer.

Each represents an essential layer of the stack as illustrated in Figure 7. LAMP delivers a strong platform for developing and hosting large, database-driven, and dynamic website.

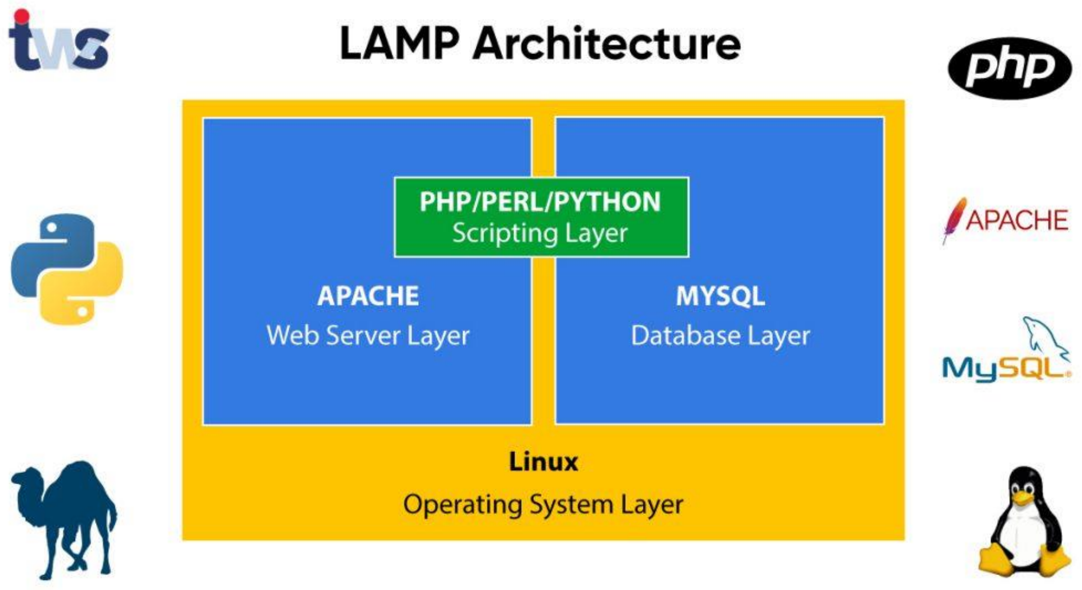


Figure 7. LAMP stack architecture [21]

Pros of LAMP:

LAMP has the biggest and oldest community, therefore, the LAMP stack was used by hundreds of thousands of companies and well maintained and supported. Being Linux based, you will find help for any topic in the large open-source community. MySQL is a very reliable and scalable solution. PHP is in version 7 and is also supported by a mature and big community.

All four components of the LAMP stack are open source software, so they are free and available for download, thus, reducing the cost of development.

Since the LAMP stack is server-side rendering, developers have full control of the server and decide which versions and software to install, so they do not have to depend on the client's browser.

LAMP can easily customize the stack and interchange the components with other open source software. Therefore, there are some other alternatives to the LAMP stack:

- **LEMP** (Linux, NGINX, MySQL/MariaDB, PHP/Perl/Python)

- **LAPP** (Linux, Apache, PostgreSQL, PHP)
- **LEAP** (Linux, Eucalyptus, AppScale, Python)
- **LLMP** (Linux, Lighttpd, MySQL/MariaDB, PHP/Perl/Python)
- **WAMP** (Windows, Apache, MySQL/MariaDB, PHP/Perl/Python)
- **WIMP** (Windows, Internet Information Services, MySQL/MariaDB, PHP/Perl/Python)
- **MAMP** (Mac OS x, Apache, MySQL/MariaDB, PHP/Perl/Python)

Cons of LAMP:

One of the disadvantages of the LAMP stack is being one of the biggest and oldest stacks. There was a lot of legacy codebases in the past when the programming world was not mature and people often not following best practice. Those codebases were a mess and not many developers want to debug or maintain them.

Starting with PHP is easy, but mastering it is hard. This is also true for security in these PHP apps. Some would also describe it as a scripting language instead of a real programming language because it is not strongly typed and not pre-compiled.

Performance speed is also one of the LAMP application's flaws. Since it uses server side rendering, the server has to do all the works from data operations such as writing to the database, calculations, transform the data to a readable format, render the static page, and then serve the final result to clients.

5.2 MERN

As mentioned above in the introduction, MERN stands for MongoDB, Express, React, and Node. MERN is a popular JavaScript stack used for the process of development. It is a combination of Open Sources written in the single most popular programing language, JavaScript. These components help provide end-to-end framework support to developers.

- **MongoDB**: is a NoSQL document-oriented database. Data is stored in flexible documents with a JSON (JavaScript Object Notation)-based query language. The content, size, and number of fields in the documents can differ from one to the next. The data structure is dynamic, flexible, and easy to scale.

- **Express.js**: It is a fast and minimalist web framework used for Node.js. The Express framework is designed for building robust web applications and APIs. It is known for its fast speed and minimalist structure, with many features available as plugins.

- **React**: It is a front-end JavaScript library used to build user interfaces. React was created by a software engineer at Facebook, and was later open-sourced. Instead of relying on templates to automate the creation of repetitive HTML or DOM (Document Object Model) elements, React uses a full-featured programming language (JavaScript) to construct repetitive or conditional DOM elements. The author will explain in more detail about React library in section 6.1.

- **Node.js**: Node.js was initially built for Google Chrome, and later open-sourced by Google in 2008. It is built on Chrome’s V8 JavaScript engine. It is designed to build scalable network applications and can execute JavaScript code outside of a browser. See Figure 8 below.

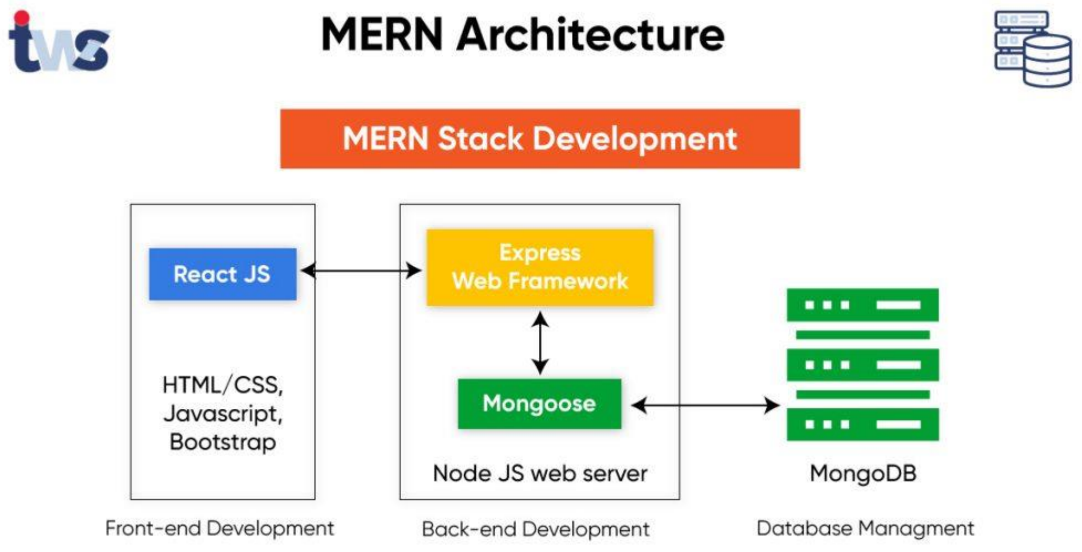


Figure 8. MERN stack architecture [21]

Pros of MERN

One of the most benefits when using the MERN stack is that it uses the only JavaScript for a complete web development cycle from frontend to backend. Since developers use JavaScript, it brings all the benefits of JavaScript to the application, which were explained in section 3.2.

The web application developed by MERN also takes advantage of SPA, where the author had described above in section 2.2.

In the MERN stack, the developer separates the web application into three components: the backend, the frontend, and the database system. These three components connect to others using APIs, (often REST APIs), which are based on HTTP and imitate web communication styles, making them very advantageous to use in the MERN stack. With the REST APIs, developers have the possibility, for example, of developing new front-ends and developing new applications that can be linked to the same system very easily. [22]

Finally, the MERN stack uses React library to develop the frontend. React.js is based on the single-page application (SPA) which involves having a web application accessed from a single web page. This avoids loading a new page with each action and makes for a greatly streamlined user experience. For an e-commerce site, this type of fluid and optimized experience is essential for staying in the race against the competition.

Cons of MERN

One of the disadvantages of the MERN stack is JavaScript. Since the whole MERN stack uses JavaScript, it inherits both the remarkable advantages and the drawbacks of JavaScript.

React.js is a library and it was not matured yet. Because of that, the syntax and best practices for programming React keep on changing. Furthermore, React gives developers more freedom to coding structure and styling. For example, Angular forces developers to use TypeScript, but in React, TypeScript is optional. Thus, it makes the codebase more and more inconsistent as time goes on.

6 Development of the application

In this section, the author will give an overview of the main tools for web application development based on the MERN stack. The target web application will be a single-page application that has an Admin dashboard for content management and the UI for normal users. The author used React and Next.js for the frontend development, KeystoneJS, and GraphQL for the backend development.

One of the advantages of CMS is that the setup and development speed for a basic web store. CMS provides all the necessary config, dashboard, and connection with the database, in contrast to the MERN stack where developers need to build everything from scratch. However, the MERN stack has some benefits over CMS such as SPA design, modern technologies, and better performance and maintainability. Because of that, the author had chosen Next.js as the main framework for the frontend development and KeystoneJS for the backend development. The author had given more detail about key technologies that were used in the development of the web store.

6.1 React Library

With the rise of the SPA and MVVM pattern, the web development community has advanced to a new era. However, soon after that, people realize a new obstacle in the render client user interface with complex structure and keep changing over time. The application then because slow and sluggish because of the scaling capacity of JavaScript when the datasets become more dynamic and enormous. Therefore, the engineers at Facebook have created a new library to solve their challenges; it must be maintainable, at the same time, scalable. Hence, React was released in 2013 and introduced some original ideas in web advancement. However, these advances were welcomed by the web development community at that time. [23]

In the traditional architecture, people apply two-way-data binding for the view part, the view needs to listen to the change from the model, and at the same time, handle the interaction with the user to update the view. When the application grows, new models and views will be added to the application, thus, the connection between views and their models becomes more and more complicated. In many cases, an update on a model or view might affect others without the developer’s awareness. The difficulty in maintaining and tracking error keep in-creasing as the application growth, it prevents the development of new features and development when releasing [23].

Furthermore, when displaying the view on the Browser, it creates a Document Object Model (DOM) of that application. The DOM represents the application’s view in a tree structure and demonstrates the structure to the layout in HyperText Markup Language (HTML) tag. In SPA, the developer uses JavaScript to create and modify the DOM structure, however, this approach has two issues. The first one is the difficulty in maintaining and debugging as discussed above. The second one is the speed that DOM takes to update its structure because DOM mutation is not optimized for performance speed. [24]

React was designed to deal with a single set of problems, which is displaying complex and ever-changing data to the user interface in a large-scale application. The View in MVVM must be light to rendered and updated, at the same time, not increasing the strain to the machine, the code must be declarative, easy to debug, and maintain. In order to achieve those results, React team has introduced different concepts regarding web development workflows and best practices [23].

Virtual DOM

In React document, they define Virtual DOM (VDOM) is a programming concept where a copy of a real DOM is stored in memory and synced together, it’s implemented by libraries in JavaScript on top of browser APIs. Since virtual DOM is not a specific technology, in the scope of the thesis, the author only considers its definition in React world. Thus, the term “virtual DOM” is often called as React element and they are synced by the ReactDOM library. In React 16, “fibers” (React Fiber) are implemented into the React engine to empower the rendering of the virtual DOM. Hence, it might be considered as a part of the “virtual DOM” which holds additional information about the component tree. [25]

6.2 GraphQL

According to the official document, GraphQL is a query language and a server-side runtime for executing queries. GraphQL is not tied to any specific database or storage engine and is based on the type of system that defines the application data. [26]

GraphQL minimizes the amount of data that is transferred across the wire by being selective about the data depending on the client application’s needs. The GraphQL server only exposes a single endpoint and responds with precisely the data a client requested. Thus, a mobile client can fetch less information, because it may not be needed on a small screen compared to the larger screen for the web application. [27]

Secondly, another useful feature of GraphQL is that it makes it simple to fetch all required data with one single request. The structure of the GraphQL server makes it possible to declaratively fetch data as it only exposes a single endpoint. Thus, it can be used to connect the backend application and fulfill each client's requirements without dedicating a separate API for each client.

Furthermore, GraphQL allows the client to inquire about the fields, types, and supported queries. This feature allows GraphQL to perform query autocompletion and provide schema documentation. With this feature, the client can know exactly what operations are supported and the fields, which are required, and what sort of results can be returned.

From that bulk information, the client can then carefully choose only the relevant parts. [28]

Moreover, GraphQL requires a type system, which allows for syntactic query validation and the server responds with appropriate error messages. Therefore, if the schema on the server defies that it is expecting a query parameter to be an integer, a different data type will result in a validation error.

However, GraphQL still has some flaws when comparing to other traditional REST APIs, one of them is caching. Since GraphQL can provide different query data through the same endpoint, hence, implement caching in GraphQL is more complex. In this thesis, to overcome this disadvantage, the author had used Appolo Client, which stores the results of its GraphQL queries in a normalized, in-memory cache. This enables your client to respond to future queries for the same data without sending unnecessary network requests. [29]

6.3 Next.js

Next.js is a React framework for developing SPA. It was built on top of React library to help and further enhance the performance of both the product and the development team. Next.js also provides a lot of advantages such as better performance, better user experience, and rapid feature development as shown in Figures 9 and 10.

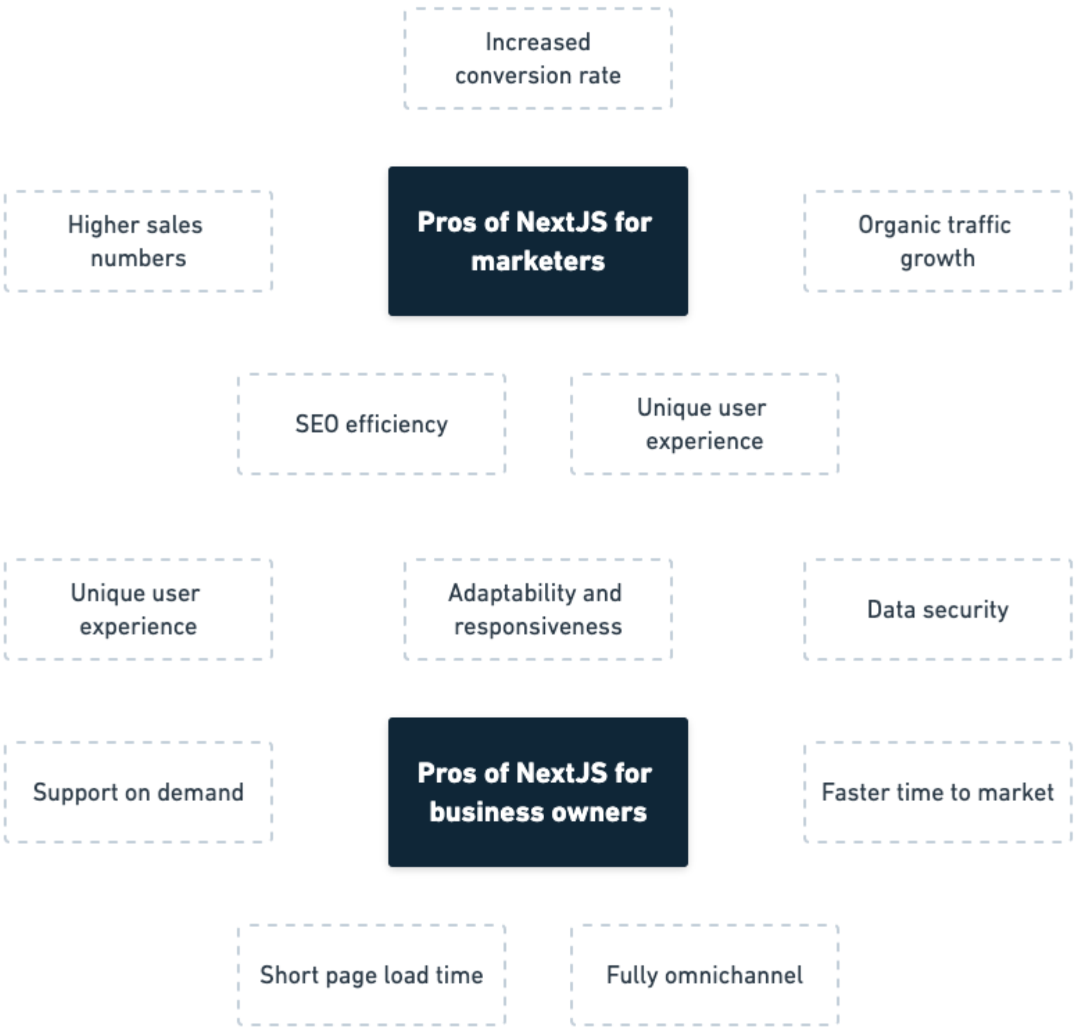


Figure 9. Pros of Next.js for the business side [30]

As the author mentioned in section 2, MPA and SPA, one of the disadvantages for the SPA was the SEO, however, Next.js had overcome this flaw by providing pre-render every view by default. [31] This means that instead of having client-side JavaScript do all the render work once the user navigates to the view, Next.js generates HTML with minimal JavaScript code necessary for that view and only runs to make it fully interactive when the browser loads it. Next.js provides two ways of pre-rendering:

- Static Generation: The HTML is generated at build time and will be reused on each request. They can be cached by CDN without extra configuration to improve performance.

- Server-side Rendering: The HTML is generated on each request, similar to WordPress and other traditional web applications.

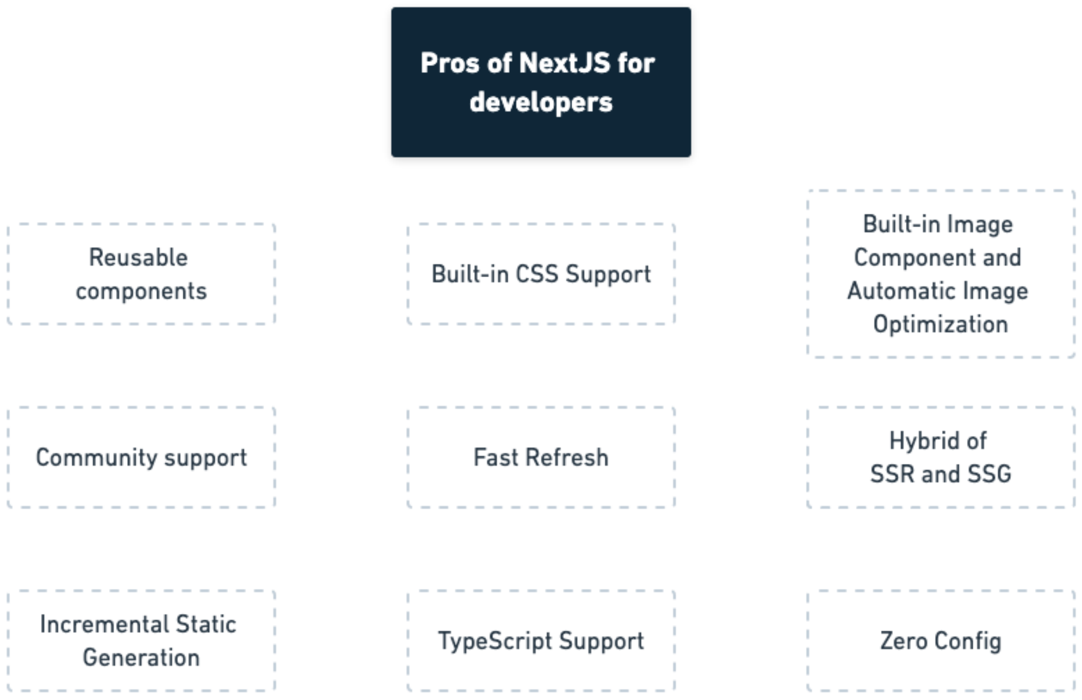


Figure 10. Pros of Next.js for developers [30]

Moreover, Next.js also allows developers to create a flexible “hybrid” app by mixing both normal client-side rendering and the pre-rendering depended on use cases.

6.4 KeystoneJS

In this thesis project, the author used KeystoneJS for building the backend. KeystoneJS is a headless CMS framework, which is a content management system to build server applications that connect with a database. KeystoneJS is based on Express for Node.js and uses MongoDB as its default database. Unlike WordPress, where non-technocal users can be set up the website, KeystoneJS requires more technical knowledge to develop a website, and the developer can customize everything to build a specialized application system and APIs. [32]

Some key features of KeystoneJS include:

- Developer oriented
- Auto-generated Admin UI based on data models
- Lightweight, simple to set up, and full feature GraphQL
- Flexibility and customizable: the developer can either use the provided template or start from scratch and make use of KeystoneJS tools without the need to write everything.
- Extendable and compatible with other services: Despite being a framework, KeystoneJS can integrate with other JavaScript packages and other third-party services such as Amazon S3, Cloudinary, and Mandrill.

The application technology infrastructure

In this section, the author will explain the technology infrastructure of the application, what is the role of each, and how they interact with each other. A summary of the application's infrastructure is illustrated in table 1 below:

Table 1. An overview of the online store infrastructure

Frontend		**Backend**
React	Apollo Client	KeystoneJS
• Next.js for server-side rendering, routing, and tooling • Styled Components to styling and to use js in CSS but still keep the CSS syntax • Apollo React hooks for interacting with Apollo client • Jest & React Testing Library for testing	• Providing GraphQL API • Performing GraphQL mutation • Fetching data from database vie GraphQL Queries • Caching GraphQL Data	• Providing admin interface to manage data • Providing GraphQL CRUD APIs for the database • Defining Schema and Data relationship • Performing authentication, checking roles and permission

		• Sending email • Charging card with Stripe

From table 1, the author had shown his idea to develop the application with three main components: React, Appolo Client, and KeystoneJS. React was in charge of rendering the view. Apollo Client helped manage both local and remote state data by providing GraphQL query hooks mutation. Lastly, KeystoneJS was responsible for defining data models, their roles and relationship, and handling CRUD operations, and building an admin interface to manage data.

Next, the author will give an overview of the development process, from setting up the development environment, structuring of the application, developing main features, and testing.

6.5 Setting up the development environment

To have a better development process, the author used some developer tools to test and debug the application during the development.

Tool name	**Definition & Usage**
Node.js	• Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine [10] • Node.js allows the developer to write JavaScript outside the browser • Node.js is designed to build scalable network applications

MongoDB	• MongoDB is a document database, which means it stores data in JSON-like documents [34] • It is a powerful query language, which allows the developer to filter and sort by any field, no matter how nested it may be within a document.
MongoDB Compass	• MongoDB is a Graphical User Interface for MongoDB • It supports developer to have a clearer view about document structure, thus, have a better decision when working with data-base
Visual Studio Code	• Visual Studio Code is a lightweight but powerful source code ed-itor • It combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debug-ging (https://code.visualstudio.com/)
React Developer Tools	• React Developer Tools is a Chrome DevTools extension for the open-source React JavaScript library. • It allows you to inspect the React component hierarchies
Apollo Client Devtools	• It is a GraphQL debugging tool for Apollo Client • It has a built-in GraphQL console, a query watcher, a mutation inspector, and a cache inspector

Table 2. An overview of development tools was used in this application

In table 2, the author had described some of the required tools for setting up the devel-opment environment and their uses. There are still other tools such as npm, Git, Lodash library, Eslint, and Prettier. However, for the sake of simplicity, the author decided to not list all the dependencies but only some main ones here. A more detailed explanation will be given in the following sections.

In this project, the frontend was written in JavaScript, using React as a base, the author used the Next.js framework to build the application. Listing 1 below presents the list of the frontend dependencies.

```
"dependencies": {
    "@apollo/client": "^3.3.8",
    "@apollo/link-error": "^2.0.0-beta.3",
    "@apollo/react-ssr": "^4.0.0",
    "@stripe/react-stripe-js": "^1.1.2",
    "@stripe/stripe-js": "^1.11.0",
    "apollo-upload-client": "^14.1.3",
    "babel-core": "^7.0.0-bridge.0",
    "babel-plugin-styled-components": "^1.12.0",
    "date-fns": "^2.16.1",
    "downshift": "^6.0.12",
    "enzyme": "^3.11.0",
    "enzyme-adapter-react-16": "^1.15.5",
    "graphql": "^15.4.0",
    "graphql-tag": "^2.11.0",
    "graphql-upload": "^11.0.0",
    "lodash.debounce": "^4.0.8",
    "next": "^10.0.5",
    "next-with-apollo": "^5.1.1",
    "nprogress": "^0.2.0",
    "prop-types": "^15.7.2",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-transition-group": "^4.4.1",
    "styled-components": "^5.2.1",
    "waait": "^1.0.5"
  },
  "devDependencies": {
    "@apollo/react-testing": "^4.0.0",
    "@babel/core": "^7.12.10",
    "@babel/preset-env": "^7.12.11",
    "@testing-library/jest-dom": "^5.11.8",
    "@testing-library/react": "^11.2.3",
    "@testing-library/user-event": "^12.6.0",
    "babel-eslint": "^10.1.0",
    "babel-jest": "^26.6.3",
    "casual": "^1.6.2",
    "eslint": "^7.17.0",
    "eslint-config-airbnb": "^18.2.1",
    "eslint-config-prettier": "^7.1.0",
    "eslint-config-wesbos": "^1.0.1",
    "eslint-plugin-html": "^6.1.1",
    "eslint-plugin-import": "^2.22.1",
    "eslint-plugin-jsx-a11y": "^6.4.1",
    "eslint-plugin-prettier": "^3.3.1",
    "eslint-plugin-react": "^7.22.0",
    "eslint-plugin-react-hooks": "^4.2.0",
    "jest": "^26.6.3",
    "prettier": "^2.2.1",
    "react-test-renderer": "^17.0.1"
  },
```

Listing 1. List of frontend dependencies

For the backend, besides using KeystoneJS as the main dependency, the application also used some other packages as its dependencies.

```
"dependencies": {
    "@keystone-next/admin-ui": "^8.0.1",
    "@keystone-next/auth": "^14.0.0",
    "@keystone-next/cloudinary": "^2.0.9",
    "@keystone-next/fields": "^4.1.1",
    "@keystone-next/keystone": "^9.3.0",
    "@keystone-next/types": "^12.0.0",
    "@keystonejs/server-side-graphql-client": "^1.1.2",
    "@types/nodemailer": "^6.4.0",
    "dotenv": "^8.2.0",
    "next": "^10.0.5",
    "nodemailer": "^6.4.17",
    "react": "^16.14.0",
    "react-dom": "^16.14.0",
    "stripe": "^8.130.0"
  },
  "devDependencies": {
    "@typescript-eslint/eslint-plugin": "^4.9.0",
    "@typescript-eslint/parser": "^4.9.0",
    "babel-eslint": "^10.1.0",
    "eslint": "^7.14.0",
    "eslint-config-airbnb": "^18.2.1",
    "eslint-config-airbnb-typescript": "^12.0.0",
    "eslint-config-prettier": "^6.15.0",
    "eslint-config-wesbos": "^2.0.0-beta.4",
    "eslint-plugin-html": "^6.1.1",
    "eslint-plugin-import": "^2.22.1",
    "eslint-plugin-jsx-a11y": "^6.4.1",
    "eslint-plugin-prettier": "^3.1.4",
    "eslint-plugin-react": "^7.21.5",
    "eslint-plugin-react-hooks": "^4.2.0",
    "prettier": "^2.2.1",
    "typescript": "^4.1.2"
  }
```

Listing 2. List of backend dependencies

In listing 1 and listing 2, besides the main dependencies that had been explained in the above sections, the application still has other important dependencies such as:

- Babel: It is a toolchain that is used to convert ECMAScript 2015+ JavaScript features into a backward-compatible version of JavaScript for browsers that haven not supported them yet. Babel will ensure that the application can be open on different browser versions. [35]
- Dotenv: It is the module that loads environment variables from a .env file into process.env. Storing configuration in the environment separate from the codebase.
- Stripe: The author used Stripe for the web application payment system. The Stripe Node library provides convenient access to the Stripe API for integrating with Stripe payment service
- Jest: It is a JavaScript test runner; the author used Jest for creating, running, and structuring tests.

- React Testing Library: It is a very lightweight solution for testing React components. In addition, it provides light utility functions on top of react-dom and react-dom/test-utils, in a way that encourages better testing practices.
- ESLint: It is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code. ESLint helps prevent potential error in development.
- Prettier: Prettier is an opinionated code formatter. ESLint enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary

6.6 GraphQL server development

The main difference between the normal MERN stack and the author’s method was in the server-side development. In the MERN stack development, the developers need to manually set up the server configuration, query to the database and define API endpoints. However, in the author’s method, it used KeystoneJS to set up the server. The author only needed to set up some simple configurations such as authentication, server connection, database, and schema. Furthermore, the author used GraphQL for querying the database, which makes the server setup, became simpler.

6.6.1 Creating Schema

In Keystone, they use **list** keyword property to represent the data model, or in the author case, the GraphQL schema. For example, the product schema would be defined in the **Product.ts** file and then be imported and put into the Keystone configuration file. The Product.ts was illustrated in listing 3 below:

```
import { integer, select, text, relationship } from '@keystone-next/fields';
import { list } from '@keystone-next/keystone/schema';


export const Product = list({
  access: {
    create: // Create permission,
    read: // Read permission,
    update: // Update permission,
    delete: // Delete permission,
  },
  fields: {
    name: text({ isRequired: true }),
    description: text({
      ui: {
        displayMode: 'textarea',
      },
    }),
    photo: relationship({
      ref: 'ProductImage.product',
      ui: {
        displayMode: 'cards',
        cardFields: ['image', 'altText'],
        inlineCreate: { fields: ['image', 'altText'] },
        inlineEdit: { fields: ['image', 'altText'] },
      },
    }),
    status: select({
      options: [
        { label: 'Draft', value: 'DRAFT' },
        { label: 'Available', value: 'AVAILABLE' },
        { label: 'Unavailable', value: 'UNAVAILABLE' },
      ],
      defaultValue: 'DRAFT',
      ui: {
        displayMode: 'segmented-control',
        createView: { fieldMode: 'hidden' },
      },
    }),
    price: integer(),
  },
});
```

Listing 3. Product schema file

In the Product.ts file, the author first needs to import the data type such as integer, text, select, and relationship. After that, the author used the **list()** function to create a schema definition, which included two main options: access and fields.

- Access option defines access control rules for the schema. These rules determine which of the CRUD (create, read, update, delete) operations users are allowed to perform.
- Fields option defines the names, types, and configuration of the fields in the list. This configuration option takes an object with field names as keys and configured field types as values.

```
allProducts() {
  id
  name
  price
  description
  photo {
    id
    image {
      publicUrlTransformed
    }
  }
}
```

Listing 4. The GraphQL query to get data of all products

Besides the normal type such as integer or text, there was a special type that was the **relationship**. The relationship type represents a relationship between two schemas. In the Product.ts file, with this setup, the GraphQL allProducts query could get data from both Product and ProductImage schema at once. For example, the fetching query in listing 4 could get data from all products, which included id, name, price, description, and photo (including photo id and its URL). See listing 4 above.

6.6.2 Authentication setup

The next step after defining the GraphQL schema was to set up the authentication, user roles, and permission for the web application. The main functionality of the Keystone authentication system was to provide a GraphQL query to authenticate a user and then begin the admin session. Furthermore, it also prepared a pre-build sign-in page in the Admin UI. Listing 5 below was an example of the authenticate function.

```
import { createAuth } from '@keystone-next/auth';


export const { withAuth } = createAuth({
  listKey: 'User',
  identityField: 'email',
  secretField: 'password',
  initFirstItem: {
    fields: ['name', 'email', 'password'],
  },
  passwordResetLink: {
    sendToken: async ({ itemId, identity, token, context }) => { /* ... */ },
    tokensValidForMins: 60,
  },
});
```

Listing 5. The authentication function

The **createAuth** function from Keystone’s auth package allows the author to configure the authentication requirements and export **withAuth** wrapper that is used later in the Keystone system configuration.

Firstly, the **listKey** option selected the schema that would be used to authenticate, in the example, the “User” data was chosen to validate the authentication. Next, the author used email and password for the identity field and secret field respectively. These two fields were used to identify and validate the signed-in user authentication.

Secondly, Keystone provided an optional feature, which allowed sign up the first user into the system via the **initFirstItem** option. If this option is enabled and there are no users in the system, the Admin UI will present a form to create an initial user in the system.

Finally, the author needed to set up the reset password feature. Fortunately, Keystone also provided support with the **passwordResetLink** option. This option added support for sending password reset links to users. The mutation sendUserPasswordResetLink allows you to send a reset token to a user via the sendToken function. The author provided a list of fields that were added into GraphQL authentication API in listing 6 below.

```
# Signin API
Mutation {
  authenticateUserWithPassword(email: String!, pass-
word: String!): UserAuthenti-cationWithPasswordResult!
}

Query {
  authenticatedItem: AuthenticatedItem
}

union AuthenticatedItem = User

union UserAuthenticationWithPasswordResult = UserAuthenticationWithPass-
wordSuccess | UserAuthenticationWithPasswordFailure

type UserAuthenticationWithPasswordSuccess {
  sessionToken: String!
  item: User!
}

type UserAuthenticationWithPasswordFailure {
  code: PasswordAuthErrorCode!
  message: String!
}

enum PasswordAuthErrorCode {
  FAILURE
  IDENTITY_NOT_FOUND
  SECRET_NOT_SET
  MULTIPLE_IDENTITY_MATCHES
  SECRET_MISMATCH
}
```

```
# Create first user
Mutation {
  createInitialUser(data: CreateInitialUserInput!): UserAuthentication-
WithPass-wordSuccess!
}
input CreateInitialUserInput {
  name: String
  email: String
  password: String
}

# Reset password
Mutation {
  sendUserPasswordResetLink(email: String!): SendUserPasswordResetLinkRe-
sult
  redeemUserPasswordResetToken(email: String!, token: String!, pass-
word: String!): RedeemUserPasswordResetTokenResult
}

Query {
  validateUserPasswordResetToken(email: String!, token: String!): Vali-
dateUserPasswordResetTokenResult
}
```

Listing 6. GraphQL authenticate API generated by Keystone

6.6.3 KeystoneJS main configuration

When starting the server, the KeystoneJS looks for the **keystone.ts** file, which contained all the server configurations. Listing 7 below illustrated the Keystone configuration file.

```
import { config, createSchema } from '@keystone-next/keystone/schema';
import { withItemData, statelessSessions } from '@keystone-next/key-
stone/session';

import { withAuth } from './auth';
import { Role } from './schemas/Role';
import { OrderItem } from './schemas/OrderItem';
import { Order } from './schemas/Order';
import { CartItem } from './schemas/CartItem';
import { ProductImage } from './schemas/ProductImage';
import { Product } from './schemas/Product';
import { User } from './schemas/User';

const databaseURL = process.env.DATABASE_URL
const sessionConfig = {
  maxAge: 60 * 60 * 24 * 360,
  secret: process.env.COOKIE_SECRET,
};

export default withAuth(
  config({
    server: {
      cors: {
        origin: [process.env.FRONTEND_URL],
        credentials: true,
      },
    },
    db: {
      adapter: 'mongoose',
      url: databaseURL,
      async onConnect(keystone) {
        console.log('Connected to the database!');
      },
    },
    lists: createSchema({
      User,
      Product,
      ProductImage,
      CartItem,
      OrderItem,
      Order,
      Role,
    }),
    ui: {
      isAccessAllowed: ({ session }) =>
        !!session?.data,
    },
    session: withItemData(statelessSessions(sessionConfig), {
      User: `id name email role { ${permissionsList.join(' ')} }`,
    }),
  })
);
```

Listing 7. KeystoneJS configuration file

In the previous Authentication setup section, the author had created a function **withAuth,** which is used to authenticate the user. In this section, the author used the **withAuth** function to wrap the main Keystone configuration, so that Keystone would modify the **config** to inject extra fields, extra GraphQL queries and mutations, and custom Admin UI functionality into the system.

In listing 7, five main options were used in the Keystone configuration file, which was: server, db, lists, ui, and session. There are more options for a more complex application, however, for the sake of simplicity, the author decided to use these five options in this thesis.

Firstly, the **server** option is used to configure the Express web server. In this application, the author configured cors middleware so that the server will accept a connection request from the frontend URL.

Secondly, for the database configuration, the application used MongoDB as its database, the author used “mongoose” as an adapter and then put the database URL in the “url” option.

After that, the author started to set up a data model by putting GraphQL schemas in the method **createSchema**. Keystone refers to data model or schema as “lists”, this was where the author used the Product schema that was defined in section 6.6.1. Besides the Product schema, there were some other schemas such as User, Role, and Order.

Finally, the author configured the **ui** option to control the Admin UI, for this application, any user that signed in was able to see the Dashboard. To determine the sign-in state of the user, the **session** option was configured to store the session data, which was a user’s id, in a cookie.

6.7 Client-side React development

In the application client-side development, the author used Next.js, which is a React framework that provides support to server-side rendering and further enhances the performance of the React application. Furthermore, the author also used Apollo Client for the application state management, which allowed the author to manage both local and remote data with GraphQL. Moreover, one of the key features of Apolo Client is that it provides support for data caching without any configuration. This feature has solved GraphQL's biggest drawback and simpler GraphQL development on the client-side.

Therefore, in this section, the author focused on the setup of Apollo Client into the application and give a simple example of how to fetch and display data with GraphQL.

6.7.1 Setting up Apollo Client

Firstly, the author created a function named **createClient** that took **header** and **initialState** then return a new **ApolloClient** instance. Listing 8 shows a simple version of the ApolloClient configuration file. The ApolloClient required two options, which were **link** and **cache**.

Inside the link option, the author used **ApolloLink** to configure the network connection to the GraphQL server. The first property that the author set up for the ApolloLink was the error handler message; this gave the author some informative message during debugging. The second property **createUploadLink** was the main setting for the **apollo-upload-client**, the author used this package to add the uploading file feature to the ApolloClient. The setup for this property was similar to **apollo-link-http**, it required the uri to the GraphQL server, and some additionals for the network request, which were the credentials in the fetchOptions and the headers. Besides the link option, the author set up the cache option to save the cache in the browser and restore the data if there was any initialState.

```
import { ApolloClient, ApolloLink, InMemoryCache } from '@apollo/client';
import { onError } from '@apollo/link-error';
import { getDataFromTree } from '@apollo/client/react/ssr';
import { createUploadLink } from 'apollo-upload-client';
import withApollo from 'next-with-apollo';

function createClient({ headers, initialState }) {
  return new ApolloClient({
    link: ApolloLink.from([
      onError(({ graphQLErrors, networkError }) => {
        if (graphQLErrors)
          graphQLErrors.forEach(({ message, locations, path }) =>
            console.log(
              `[GraphQL error]: Message: ${message}, Location: ${loca-
tions}, Path: ${path}`
            )
          );
        if (networkError)
          console.log(
            `[Network error]: ${networkError}. Backend is unreachable?`
          );
      }),

      createUploadLink({
        uri: process.env.GRAPHQL_SERVER,
        headers,
        fetchOptions: {
          credentials: 'include',
        },
      }),
    ]),
    cache: new InMemoryCache().restore(initialState || {}),
  });
}

export default withApollo(createClient, { getDataFromTree });
```

Listing 8. Apollo Client configuration file

After that, the author wrapped the **createClient** function and **getDataFromTree** under **withApollo** to instruct Apollo Client to execute all of the queries required by the tree's components. The getDataFromTree function looped through the entire application and executes every required query it encountered. It returns a Promise that resolves when all result data is ready in the Apollo Client cache. Then when the Promise resolved, it rendered the React component along with the current state of the Apollo Client cache.

```
import { ApolloProvider } from '@apollo/client';
import Page from '../components/Page';
import withApollo from './apolloClient';


function MyApp({ Component, pageProps, apollo }) {
  return (
    <ApolloProvider client={apollo}>
      <Page>
        <Component {...pageProps} />
      </Page>
    </ApolloProvider>
  );
}

export default withAppolo(MyApp);
```

Finally, to inject the Apollo Client instance into the application, the author wrapped the entire application in the **ApolloProvider** component tag and passing the apollo parameter to the prop client inside ApolloProvider. Then, the author needed to import the Apollo configuration file and wrapped the MyApp component.

6.7.2 Fetching and displaying data with GraphQL

In this section, the author demonstrated the fetching and displaying data with graphQL with a simple example component in listing 9

```
import { useQuery } from '@apollo/client';
import gql from 'graphql-tag';

const ALL_PRODUCTS_QUERY = gql`
  query ALL_PRODUCTS_QUERY {
    allProducts {
      id
      name
      price
      description
      photo {
        id
        image {
          publicUrlTransformed
        }
      }
    }
  }
`;

export default function Products() {
  const { data, error, loading } = useQuery(ALL_PRODUCTS_QUERY);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error.message}</p>;
  return (
    <div>
      {data.allProducts.map((product) => {
        const { id, name, price, description } = product;
        return (
          <div key={id}>
            Product name: {name}
            Price: {price}
            Description: {description}
          </div>
        );
      })}
    </div>
  );
}
```

Listing 9. Products component

Firstly, the author defined the ALL_PRODUCTS_QUERY using gql, which is a JavaScript template literal tag that parses GraphQL query strings into the standard GraphQL from the graphql-tag package. This query returned all the products in the database, including id, name, price, description, and photo for each product.

After defining the query, to run a query within a React component, the author called **useQuery** hook from the apollo package and passed it the ALL_PRODUCTS_QUERY. When the component is rendered, the useQuery hook returned an object from Apollo Client that contains loading, error, and data properties.

Finally, the author could render the content of the component based on these properties such as loading UI while waiting for fetching data, displaying the error message when there was an error, and displaying the product list after getting the fetching data.

7 Conclusion

Within the scope of the thesis, the author tried to provide a modern way to build a web application using the MERN stack but still keeps the benefit of the traditional CMS. The author mainly focused on discussing the common CMS, the changes in user behavior, and the benefits and drawbacks of the traditional and modern web development design pattern, language, and development stack. After that, the author provided a summary of setting up the GraphQL server with Keystone and some examples of how to set up and connect the frontend to the server. The backend used KeystoneJS, which provides a customized CMS dashboard and GraphQL API. The frontend used Next.js, which supports server-side rendering to have better SEO, and Apollo Client to provide the solution for GraphQL drawbacks, which are caching and file uploading.

In conclusion, the author's approach aims to improve the web development of the MERN stack by delegating the backend logic and the CMS to KeystoneJS while the developer can focus more on the frontend of the MERN stack. With the size of the application in this thesis, since the developer does not need to spending time to develop the backend, set up the GraphQL server, and building the Admin Dashboard, the author had concluded that with this approach, the simple and straightforward implementation of both frontend and backend will ultimately speed up the development process, improve the maintainability and the performance of the application while still be able to keep the benefit of MERN stack and CMS.

References

The layout of this page in the number (Vancouver) referencing system:

1 Rancic Moogk, D. 2012. Minimum Viable Product and the Importance of Experimentation in Technology Startups. Technology Innovation Management Review, 2(3): 23-26. URL: http://doi.org/10.22215/timreview/535. Accessed: 15 March 2020

2 Sanchit Aggarwal, Jyoti Verma. Comparative analysis of MEAN stack and MERN stack. Sanchit Aggarwal et al. International Journal of Recent Research Aspects. Vol. 5, Issue 1, March 2018, pp. 127-132

3 Sergey Valuy, Web dev zone – Analysis. A Comparison of Single-Page and Multi-Page Applications. Jun 17th, 2020. URL: https://dzone.com/articles/the-comparison-of-single-page-and-multi-page-appli. Accessed: 06 July 2020

4 Michael s. Mikowski, josh c. Powell. Single Page Web Applications. United States of America: Manning Publications Co.; 2014.

5 Cambridge, MA | September 14, 2009. Akamai Reveals 2 Seconds As The New Threshold Of Acceptability For ECommerce Web Page Response Times. URL: https://www.akamai.com/uk/en/about/news/press/2009-press/akamai-reveals-2-seconds-as-the-new-threshold-of-acceptability-for-ecommerce-web-page-response-times.jsp . Accessed: 15 March 2021

6 Sorensen, E., Mikailesc, M.: Model-View-ViewModel (MVVM) Design Pattern using Windows Presentation Foundation (WPF) Technology. The University of Southern Denmark. MegaByte J. 9, 1–19 (2010)

7 Usage statistics of PHP for websites. URL: https://w3techs.com/technologies/details/pl-php. Accessed: 24 April 2021

8 PHP main page. www.php.net. Access: 24 April 2021

9 Programming languages used by software developers worldwide as of 2020, by deployment type. URL: https://www.statista.com/statistics/869092/worldwide-software-developer-survey-languages-used. Access: 27 March 2020

10 About Node.js®. URL: https://nodejs.org/en/about/. Access: 24 April 2021

11 PHP vs JavaScript: How to Choose the Best Language for Your Project. URL: https://www.freecodecamp.org/news/php-vs-javascript-which-technology-will-suit-your-business-better/. Access: 24 April 2021

12 GitHut 2.0. URL: https://madnight.github.io/githut/#/pull_requests/2020/4. Access: 24 April 2021

13 PHP Debugging: Pitfalls and Solutions. URL: https://www.chrisgeelhoed.com/php-debugging-pitfalls-and-solutions/. Access: 24 April 2021

14 2008 Prediction 4: PHP Problems. URL: www.tbray.org/ongoing/When/200x/2008/01/04/Predictions-PHP. Access: 24 April 2021

15 Rick Grehan. Debugging PHP Web apps is hard to do. URL: https://www.infoworld.com/article/2628045/debugging-php-web-apps-is-hard-to-do.html. Access: 24 April 2021

16 Maintainability Pitfalls in PHP. URL: https://codefork.com/blog/index.php/2008/01/08/maintainability-pitfalls-in-php/. Access: 24 April 2021

17 Savan, K., Rathod, V.R., & Jigna, B.P. (2011). Performance Analysis of Content Management Systems- Joomla, Drupal and WordPress. International Journal of Computer Applications, 21, 39-43.

18 Vesic, Slavimir & Minović, Miroslav. (2015). Single Page Applications: Trend or Future. Info M. URL: https://www.researchgate.net/publication/286449751_Single_Page_Applications_Trend_or_Future. Access: 24 April 2021

19 Maura Monaghan. Website Load Time Statistics: Why Speed Matters in 2021. April 14, 2021. URL: https://www.websitebuilderexpert.com/building-websites/website-load-time-statistics/. Access: 24 April 2021

20 How Many Software Developers Are in the US and the World? URL: https://www.daxx.com/blog/development-trends/number-software-developers-world. Access: 24 April 2021

21 Karan Sood. December 1, 2020. Best Technology Stack for Web Application Development. URL: https://www.tekkiwebsolutions.com/blog/technology-stack-for-web-development. Access: 24 April 2021

22 Tanjona. March 31, 2020. What are the MERN stack's advantages? URL: https://www.bocasay.com/what-mern-stacks-advantages. Access: 24 April 2021

23 Cory Gackenheimer. Introduction to React. Springer Science+Business Media New York; 2015

24 Artemij Fedosejev. React.js Essentials. UK: Packt Publishing Ltd.; 2015

25 Virtual DOM and Internals. URL: https://reactjs.org/docs/faq-internals.html. Access: 24 April 2021

26 Porcello, E, Banks, A. Learning GraphQL: Declarative Data Fetching for Modern Web Apps. O'Reilly Media; 2018.

27 Ekene Eze. November 02, 2018. GraphQL - The Good and the Bad. URL: https://scotch.io/tutorials/graphql-the-good-and-the-bad. Access: 24 April 2021

28 Brian K. Beginning GraphQL. Birmingham: Packt Publishing; 2018

29 Configuring the cache. URL: https://www.apollographql.com/docs/react/caching/cache-configuration. Access: 24 April 2021

30 Tomasz Grabski. December 30. 2020Pros and cons of nextjs in 2021. URL: https://pagepro.co/blog/pros-and-cons-of-nextjs/. Access: 24 April 2021

31 Data Fetching. URL: https://nextjs.org/docs/basic-features/data-fetching. Access: 24 April 2021

32 James Kolce. July 5, 2017. KeystoneJS: The Best Node.js Alternative to WordPress. URL: https://www.sitepoint.com/keystonejs-best-node-js-alternative-wordpress/#:~:text=KeystoneJS%20is%20a%20content%20management,uses%20MongoDB%20for%20data%20storage. Access: 24 April 2021

33 MongoDB. URL: https://www.mongodb.com/. Access: 24 April 2021

34 Babel. URL: https://babeljs.io/. Access: 24 April 2021