APPLYING VUE.JS FRAMEWORK IN DEVELOPING WEB APPLICATIONS

Case: Personal Reminder web application

LAHTI UNIVERSITY OF APPLIED SCIENCES LTD
Bachelor of Business Administration
Degree Program in Business Information Technology
Spring 2020
Nguyen Tran

Abstract

Author(s)	Type of publication	Published
Tran, Nguyen	Bachelor's thesis	Spring 2020
	Number of pages 38	

Title of publication

APPLYING VUE.JS FRAMEWORK IN DEVELOPING WEB APPLICATIONS

Case: Personal Reminder web application

Name of Degree

Bachelor's Thesis in Business Information Technology

Abstract

The release of JavaScript, a programming language mark an essential step in the evolution of the web development industry, especially in recent years. The thesis consists of research that proves how applying one of the most popular JavaScript frameworks, Vue.js, can create positive effects for the development process based on certain features of the framework. Additionally, the thesis acts as an instruction to build a general Vue.js project and also provides the readers with the author's coding standards and best practices, which are very helpful for new developers.

To start the research, readers need to understand the theoretical background of fundamental concepts that required to advance further into the study. This background consists of definitions of the web application as well as its components, JavaScript frameworks, Vue.js, and its libraries. By examining the implementation of a study case, which is a Reminder web application. The application is the author's Vue.js powered project, which he uses to collect the data for later analyzing process. This process includes the comparisons between the implementations with and without the help of the Vue.js framework,

The final chapters of the thesis include research results, reliability, limitation, validation, and suggestions for future studies.

Keywords

Web application, JavaScript Frameworks, Vue.js

CONTENTS

1 INTRODUCTION

1.1 Background

For the last seven years, JavaScript has been ranked the most used programming language every single year, according to the Stack Overflow survey (Stack Overflow 2019). In 2019, statistics showed that 67.8% number of developers are using JavaScript as one of their coding languages. Besides, 95.2% of the world's websites, roughly around 1.52 billion websites, are written by JavaScript (DeGroat 2019).

JavaScript, along with Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) are the main languages of a website or web application. While HTML and CSS take care of the site's style and structure, JavaScript allows developers to create functions and behaviors for the page (DeGroat 2019). This language works with the document object model (DOM for short), which specifies what to do when the user makes some specific actions on your application. People often think of JavaScript as a front-end language. Still, we can use JavaScript to write the application's back-end through Nodejs environment, which uses JavaScript and also very popular as well.

Moreover, scroll transitions and object movement are also available, thanks to JavaScript. Because of the variety of functionalities JavaScript has to offer, many current browsers developer are trying to help its browser to run JavaScript as fast as possible for the sake of improving their user experiences. An article by Kyla Brown from Codeacademy tells how JavaScript gained its popularity, listed these factors, which the author said: "it turned web browsers into application platform," in bold. However, there is one more point that I haven't mentioned before, which is, in my opinion, the most crucial factor that makes writing JavaScript to become not only more comfortable but also more accessible and less time-consuming. It is because JavaScript has many frameworks and libraries that tremendously help the developers to improve their coding style.

React, Vue.js, and Angular are among the most popular JavaScript framework at the moment. While React and Angular are developed and maintain respectively by Facebook and Google. The Vue.js framework was created by Evan You and his team (Duomly 2019). Although there is no big company backing Vue.js, this framework still catches much attention from the web developer community. As on GitHub, Vue.js has surpassed React, the most well-known framework at the moment on the number of stars received in their repository (as for the time of the thesis) (GitHub 2019). Through this, we can tell that although most of the companies are still using React (Figure 1), people are turning their attention toward Vue.js, and in the future, it might become more popular.

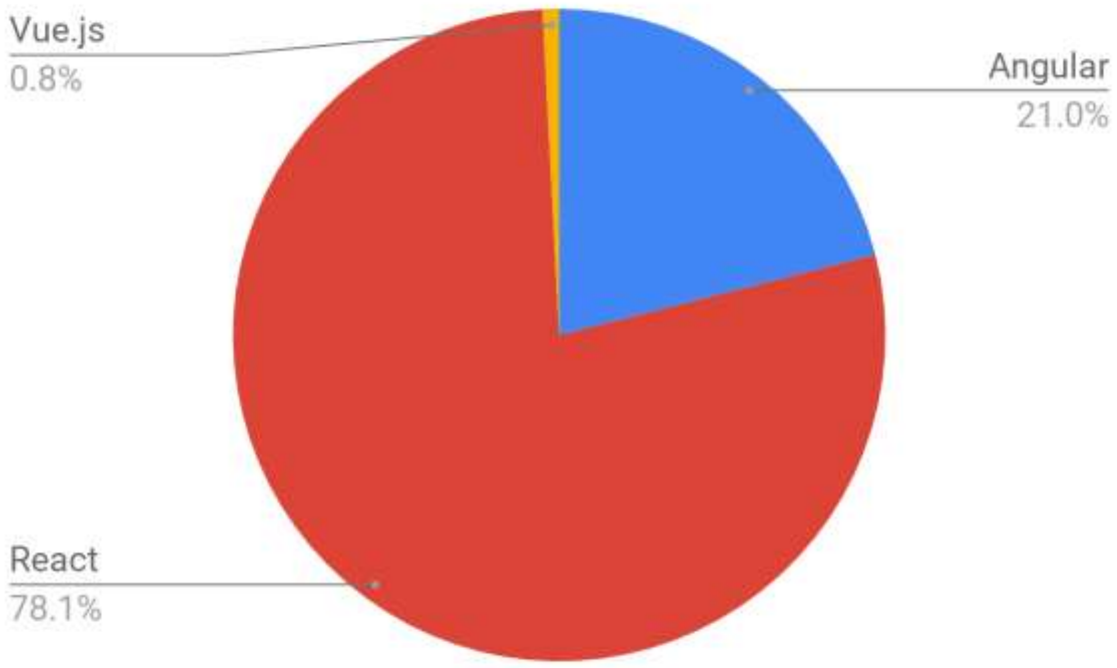


Figure 1 Frameworks job distribution (TechMagic 2019)

1.2 Motivation for the Thesis

As the popularity of the Vue.js framework grows, it is a natural behavior to turn the attention into this technology for the author. Therefore, the thesis studies some of this framework's basic functionalities to see how it impacts the development process of web applications. This research also acts as guidelines for beginner developers to understand the concept of those features and how to apply them in real-life projects. The Vue.js framework is believed to possess the steepest learning curve compared to React and Angular frameworks (Duomly 2019). Despite that, the thesis wants to provide developers with a whole picture of those features. This knowledge includes understanding the purposes of those functionalities and how to use them correctly. The motivation for this work is also to ensure that developers are following the coding standards through example implementations. This factor is especially crucial since the lack of it might cause security issues or negative impacts on performance (Multidots 2019). This Medium article by Multidots also states some advantages of implementing coding standards, such as enhancing efficiency, reducing project failure risk, allowing easier maintenance, and many more. Therefore, the thesis provides instructions to avoid coding standards and practices issues, especially for beginner developers.

1.3 Structure of the Thesis

The thesis has seven chapters. The first chapter is the introduction, including background, thesis motivation, and structure. The second chapter specifies the research questions and research design. Chapter 3 gives readers the theory background. Definition and usage of the JavaScript frameworks in general and Vue.js in particular. Additionally, in-depth comparisons between using Vue.js and Vanilla JavaScript is made to show why using Vue.js framework for building application is a step forward from using no frameworks at all, especially for new developers. This chapter also presents the concepts of a web application and its required components. The fourth chapter studies a project, which is a Reminder web application developed by the author. "Data collection and analyzing" is the topic of the fifth chapter, where the author presents collected data, then perform examinations and explanations of why he comes up with the answers. The last two sections give the thesis conclusion and summary, respectively. The author also presents some limitations as well as missing subjects that are not covered in the thesis. The graph below shows the layout of the thesis.

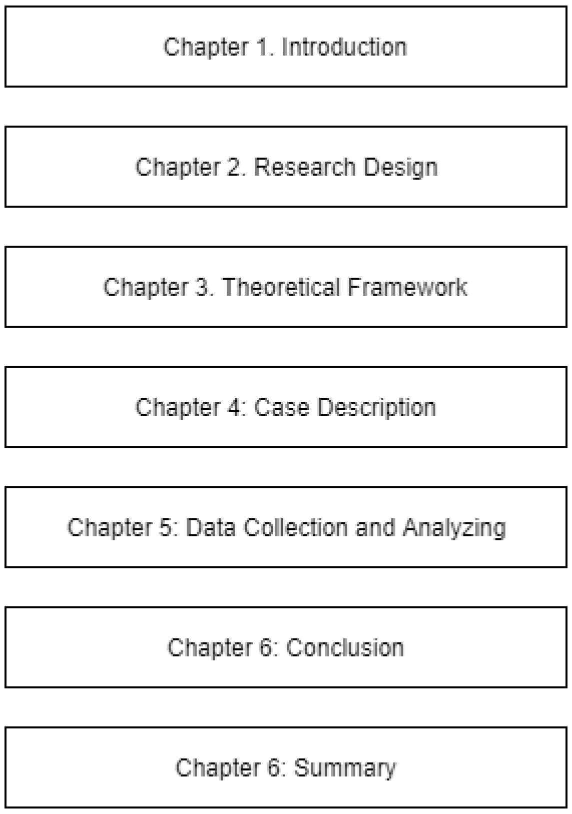

Chapter 1. Introduction
Chapter 2. Research Design
Chapter 3. Theoretical Framework
Chapter 4: Case Description
Chapter 5: Data Collection and Analyzing
Chapter 6: Conclusion
Chapter 6: Summary

Figure 2 Thesis structure

2 RESEARCH DESIGN

2.1 Research Questions

To start the research, the author comes up with some questions to help create the guidelines for the thesis. The motivation for the thesis is about helping new developers to understand some fundamental aspects of Vue.js, a JavaScript framework, and how they benefit the development process. To help achieve this goal, the author decides to point out some of the framework's highlight features and explains how those features become the advantages for the development process.

For that, the thesis aims to answer these two main questions:

- What are some of the main features of Vue.js framework, and what are their uses?
- How do these features benefit the development process?

2.2 Research Approaches

The article "The Difference Between Deduction and Induction Reasoning" written by Daniel Miessler, states that research is a tool to help confirm a concept which, in turn, contributes to the knowledge of the world. But to make people believe in research requires some kind of reasoning approach to convince them. The two most common research approaches are deduction and induction, and they are both opposite to each other. Deductive reasoning usually begins with a hypothesis, then moving toward observation and finally confirm the theory. On the other hand, "Inductive reasoning usually uses research questions to narrow the scope of the study." said Deborah Gabriel in her article (Gabriel 2013). Figure 3 below also describes the difference between the approaches. Both of these approaches are very useful in their way. The thesis is about proving how Vue.js framework creates positive effects on the development process by giving tested implementations and compare them with each other. Therefore, the deductive approach is more suitable for this research.

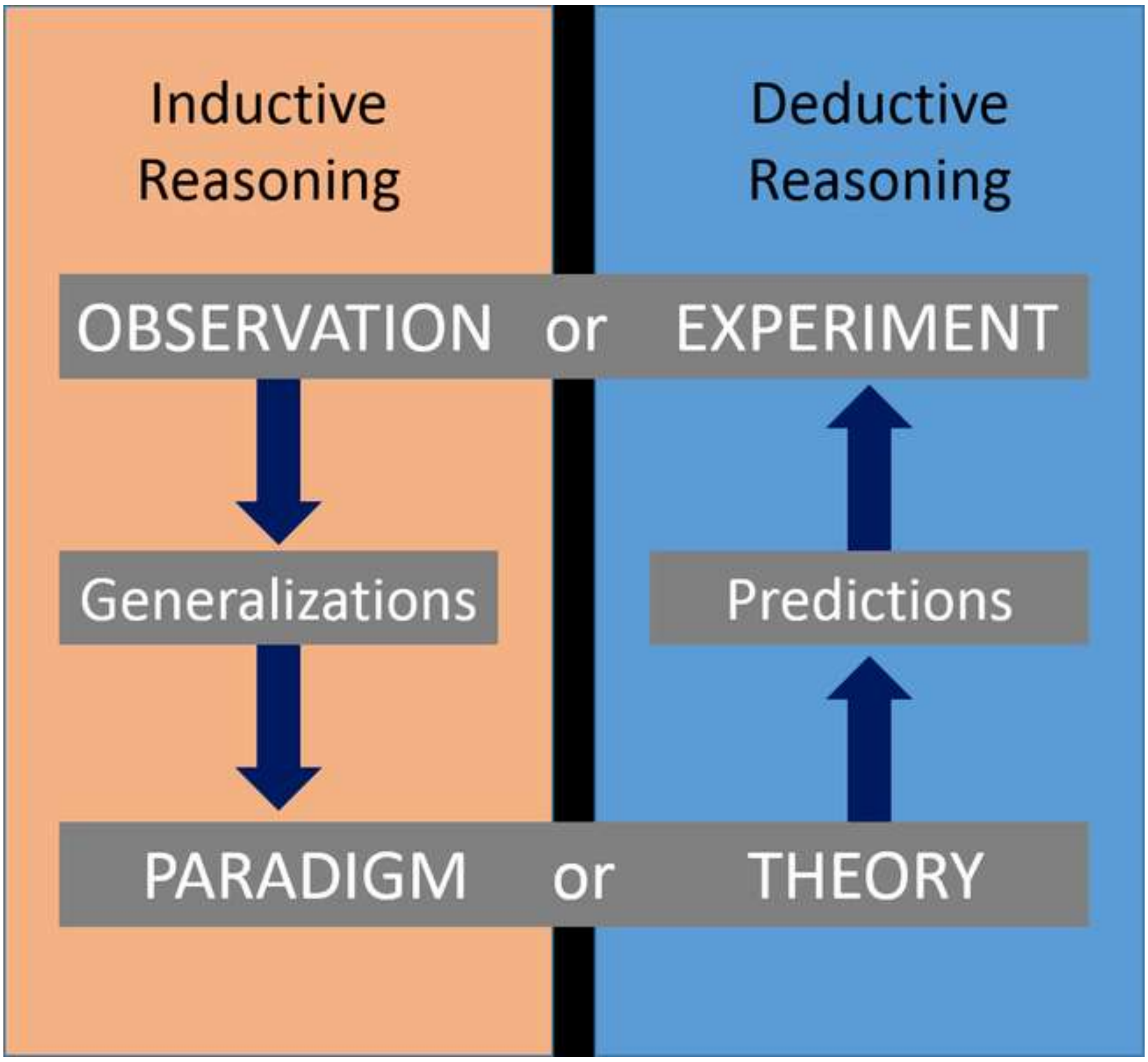
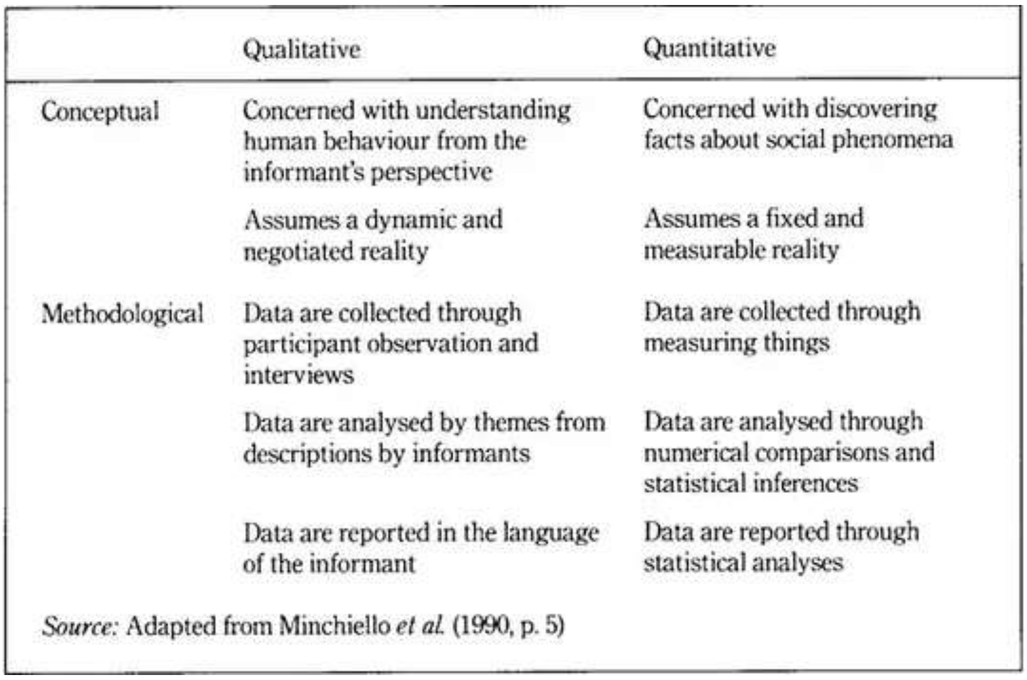


Figure 3 Inductive and Deductive Reasoning (Miessler 2019)

To complete the research, collecting and analyzing data is a crucial part and should not be missing in any research since the absence of it may create inaccurate conclusions, which in turn makes the study invalid (Bhatia 2018). Therefore, this research needs another approach for data collecting and analyzing. Qualitative and quantitative approaches are well-known for dealing with data; each has its purposes, methods, and essential for gaining different kinds of knowledge (Streefkerk 2019). The quantitative approach uses numbers, graphs, and statistics to analyze or establish theories. Surveys, experimental research, observational research, and content analysis are the ways researchers get data for a quantitative approach, which are words, examinations, and symbols (Bhatia 2018). Qualitative gain data from interviews focus groups, case studies, and discourse analysis (Streefkerk 2019). A case study is presented in chapter 4, and most of this study consists of images from the implementations provided by the author. Through these images, the author makes examinations to come up with the final answer for the research questions.

To put it another way, the author chooses the qualitative approach for the research. Some summarized detail about qualitative and quantitative methods are in the table below.

Table 1 Qualitative and Quantitative approaches (McLeod 2019)



	Qualitative	Quantitative
Conceptual	Concerned with understanding human behaviour from the informant's perspective	Concerned with discovering facts about social phenomena
	Assumes a dynamic and negotiated reality	Assumes a fixed and measurable reality
Methodological	Data are collected through participant observation and interviews	Data are collected through measuring things
	Data are analysed by themes from descriptions by informants	Data are analysed through numerical comparisons and statistical inferences
	Data are reported in the language of the informant	Data are reported through statistical analyses

Source: Adapted from Minchiello *et al.* (1990, p. 5)

3 THEORETICAL FRAMEWORK

In this chapter, the author defines core concepts (such as Web application, JavaScript frameworks, Vue.js) so that the reader can acquire enough knowledge before entering into a case study where these definitions are needed.

3.1 Web Application and Website:

The goal of the thesis is to implement Vue.js in the web application development process. Therefore, we need to understand what is a web application, and how it differs from a regular website? A website provides customers with information which nowadays mainly used by companies for marketing purposes (Benedict 2018). Websites like Wikipedia, newspapers, articles, and blog sites are great examples of a website. These sites contents usually stay the same every time we visit them.

In contrast to a website, a web application is an application that runs on an internet browser. An application requires interactions with customers, so the purposes of an application are not just for displaying information anymore. A web application can provide a wide variety of utilities such as communication, entertainment, commercing, and much much more (Benedict 2018). Some popular web applications are YouTube, Facebook, Gmail. The thesis is about implementing Vue.js into the development process of a web application to see the advantages it can bring.

3.1.1 Front-end Development

A general web application consists of four basic components: a front-end, a UI/UX design, a back-end, and a database. Front-end development is the process of handling the visual perspective for the user (Steward 2019). Mostly everything the user sees is taken care of by the front-end development process. For web development, front-end uses HTML, CSS, and JavaScript. The popularity of this trio forces most of the modern browsers such as Chrome, Mozilla, and Microsoft Edge to adapt them as soon as possible so that they can give their customers the newest features provided by these languages. As being said in a blog, HTML, CSS, and JavaScript are two of ten must-have skills for front-end developers (Morris 2017). Below is the full list of the most important knowledge required provided by the blog:

1. HTML/CSS
2. JavaScript/Jquery
3. CSS and JavaScript framework

4. CSS preprocessing
5. Version control/Git
6. Responsive design
7. Testing/ Debugging
8. Browser developer tools
9. Building and automation tools/ web performance
10. Command line

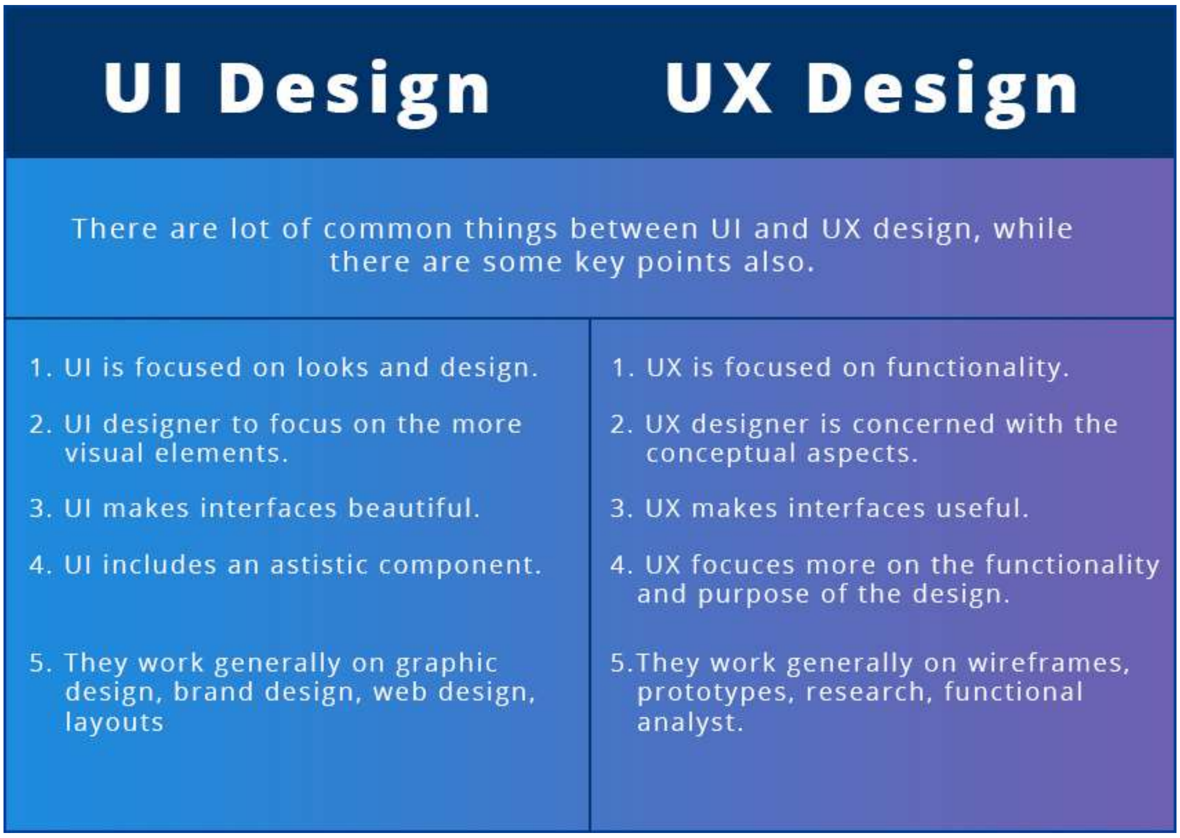
In general, these skills and pieces of knowledge are required by most front-end development positions. Each part of the front-end development requires a specific skill in this list. For example, responsive design is crucial if the developer decided to have their website or web application running on both desktop and mobile devices. The main topic of the thesis is Vue.js, a well-known JavaScript framework, which you can see falls into the third item on this list. Other skills might be mentioned throughout the thesis, but the thesis focuses on discussing the topic only.

3.1.2 User Interface/ User Experience Design

Commonly mistaken for user experience design (aka UX), user interface design (UI) is the process of deciding what your website looks like and also how all the interactions are (Hannah 2019). A UI focuses on visual presentation, from picking color schemes to typography; its goals are to ensure a good look and feel for the product. A UI design is also responsible for creating the responsiveness, efficiency, and accessibility of a website.

Different from a UI, a UX, on the other hand, is primarily work on improving customer feel and tends to guide the user to take the actions that the company wants the customer to do, fulfilling the final company goal. At the same time, it also boosts customer satisfaction (Lamprecht, 2019). A UX design decides how the product's UI design becomes as UI is a part of UX. User experience is not about visual aspect anymore but relates to anything that improves customer feel, pleasure, and satisfaction. The table below shows some key differences between a user interface design and user experience design.

Table 2 UI/UX Design (Dewebkiller 2020)



UI Design	UX Design
There are lot of common things between UI and UX design, while there are some key points also.	
1. UI is focused on looks and design.	1. UX is focused on functionality.
2. UI designer to focus on the more visual elements.	2. UX designer is concerned with the conceptual aspects.
3. UI makes interfaces beautiful.	3. UX makes interfaces useful.
4. UI includes an astistic component.	4. UX focuces more on the functionality and purpose of the design.
5. They work generally on graphic design, brand design, web design, layouts	5.They work generally on wireframes, prototypes, research, functional analyst.

This table helps developers to understand the idea of UI/UX design of a web application, which is necessary for most web applications. There is no further study or discussion about this topic in the thesis.

3.1.3 Back-end Development

In contrast with front-end development, back-end development indicates the server-side of the application. Back-end consists of codes that communicate between the client browser and the database. In detail, this includes update, manage, and monitor functionalities and data for the application. Most of these things can not be seen by the client, which is why it is called the back-end.

While front-end development uses HTML and CSS for its primary languages, back-end developers have to get familiar with programming languages such as Java, PHP, Ruby on Rails, .NET, or Python. Usually, developers only have to use one or two languages at a time, depending on the technologies their company is using, so properly learning all of these languages is not necessary. JavaScript also can be used to write back-end for applications using Node.js, a JavaScript runtime environment. Node.js is among the most successful back-end languages, and many big companies like NASA, IBM, Microsoft, PayPal, Walmart, Uber, LinkedIn, Netflix are using it (Spec India 2019).

Figure 4 Front End and Back End Developers (Ray 2018)

3.1.4 Database

A database is a collection of data that are stored systematically. This system allows querying, updating, manipulating, and managing data in a very effective and logical way. A database is usually operated by a database management system (aka DBMS), and together, they are known as a database system, or database for short (Oracle 2020). There are so many types of databases, such as Relational databases, NoSQL databases, Object-oriented databases. Each type of database is useful in its field. Like NoSQL databases, also known as nonrelational databases, can store and manage data without structure or semi-structured data. This type of database is widely used in the web industry, especially when the number of web applications is rising, and the apps themselves are getting more complicated. The author himself used a NoSQL database, MongoDB, to be specific, to create a database for the study case application, which is presented in the next chapter.

3.2 JavaScript

As the author has mentioned before in the introduction chapter, JavaScript is undeniably the most popular programming language nowadays, especially when we are talking in the web industry. Back then, when this programming language is not available, the internet only has full-static websites, which only use HTML and CSS. And Netscape has changed that by inventing, which we know for today, the most used programming language, JavaScript (Dionne 2019). At first, the appearance of JavaScript was not a total success. But eventually, JavaScript has become so popular that finding another language to replace it is very hard, if not impossible (Moerkerke 2019). JavaScript's role was to make the website become a fully functional application. It allows creating an interactive and dynamic website from which the logic can run from the client-side, which never happened before. JavaScript also got a lot of support from big technology companies such as Facebook or Google since these companies have created React and Angular, two of the most popular JavaScript frameworks, to expand its abilities further. People probably have heard a lot about these JavaScript libraries and frameworks, so what are these?

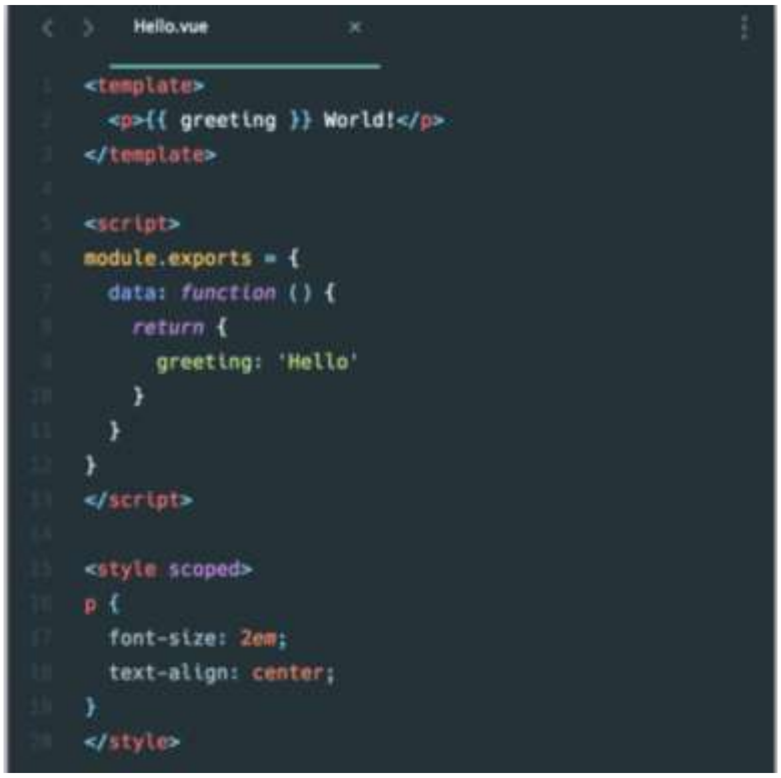
3.2.1 JavaScript Libraries and Frameworks

The definition of JavaScript might create the impression that JavaScript is an appropriate programming language for developers. But the fact is that to build a web application from scratch with Vanilla JavaScript is very hard and consumes much more time (Morris 2020). That is where JavaScript libraries and frameworks come into handy. These libraries and frameworks are templates for developers to follow to help the development process be much more efficient (Morris 2020). Specifically, they are collections of pre-written JavaScript code that can perform certain tasks with better performance and reliability. Especially if you are new developers because most of the libraries and frameworks are written by experienced and professional developers. Besides, you do not have to write those lines of code, which makes the development process faster. For example, JQuery was a popular JavaScript library that helps coding with JavaScript very convenient because this library provides a lot of single-line methods that correspond to many code lines if written with JavaScript only.

JavaScript frameworks and libraries did not stop at providing omissions and convenience in writing code. Some JavaScript frameworks even change the way developers build their projects. These include React, Angular, Vue.js, a trio of frameworks currently taking the lead in this industry. (Morris 2018.)

3.2.2 Vue.js

Vue.js was first released in 2014 by Evan You, who was working for Google using Angular before focus on developing a framework of his own (Wikipedia 2020). The version used in the thesis is Vue.js 2.0 and above, which means the thesis might not be accurate for other versions before Vue.js 2.0 and from 3.0 afterward (which releases later). According to the official Vue.js website, they define Vue.js as a progressive framework used for developing user interfaces that belong to front-end development (Vue.js 2020). Vue.js is a component-based framework from which developers can create custom elements that developers can reuse throughout the project. This feature is not exclusive for Vue.js as React and Angular also have their component feature. But for Vue.js, a component can include HTML, CSS, and JavaScript altogether, which form what a so-called single file component. In a single file component, HTML code is written within <template> tag, while JavaScript and CSS code belongs inside <script> and <style> tag respectively. Figure 6 below shows how a single file component, or a .vue file, should be.



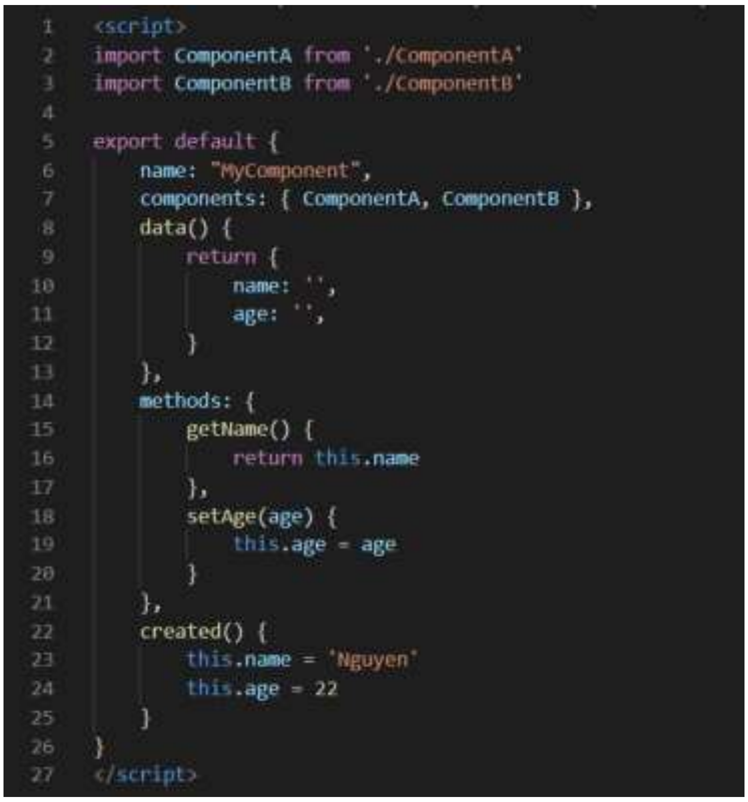
```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

Figure 5 Vue.js single file component template (Vue.js 2020)

Now, let us take a closer look at some cool features which make Vue.js shine and gaining its popularity in at present.

Component options

HTML and CSS might be familiar with most developers, but what creates the differences between frameworks is its component structure. A component is a reusable custom-made element with its methods, logic, and style. Components can communicate with each other by passing data (props) or listening to other component's event and then trigger particular actions. Take a look a figure 6 below; it is an example written by the author to demonstrate the content inside <script> tag of a single file component.



```
<script>
import ComponentA from './ComponentA'
import ComponentB from './ComponentB'

export default {
    name: "MyComponent",
    components: { ComponentA, ComponentB },
    data() {
        return {
            name: '',
            age: '',
        }
    },
    methods: {
        getName() {
            return this.name
        },
        setAge(age) {
            this.age = age
        }
    },
    created() {
        this.name = 'Nguyen'
        this.age = 22
    }
}
</script>
```

Figure 6 Component options example

Firstly, other components are imported and used in this component. After that, a default export of a JavaScript object, which is the component itself. Inside this object, developers declare all data, DOM, lifecycle, assets, and other options of the component. Figure 6 provides some basic and some of the most used options. Some options can be very straightforward, such as "name" for declaring the component's name and "data" for declaring component's data. The "Components" option is used to declare imported components that are used in this component. "Methods" option is used to declare the methods for the component. The last option in this example is "created" which is a lifecycle hook. When the component reaches the "created" stage of its lifecycle, it automatically calls this method. There are other lifecycle hooks, as well as other component options. A full list of options can be found in Vue.js official API page, or later in the reference section of the thesis (Vue.js 2020).

Reactivity

Inside the <script> tag, there is a data function from which developers can define data for the component. The object return by this function then becomes reactive resources of the component, which means, whenever a property in this object is changed, the component triggers re-render and update the view (Vuejs 2020). Another way to say it is, when developers declare data inside the data object, the component goes through every property of the object and convert them into setters and getters. Every component has its watcher to track the data changes with the help of the setters and getters. If a change is detected, the watcher triggers a re-render, which updates the user view with the latest data. Figure 7, provided by Vue.js official website below, shows how the reactivity process in Vue.js component work. This reactivity feature allows data binding to become easier and controllable. In a blog written by AltexSoft, a technology consulting company, mentioned this as an advantage of Vue.js, a cool feature Even You inherited from Angular framework and applied it into his product (Altexsoft 2019).

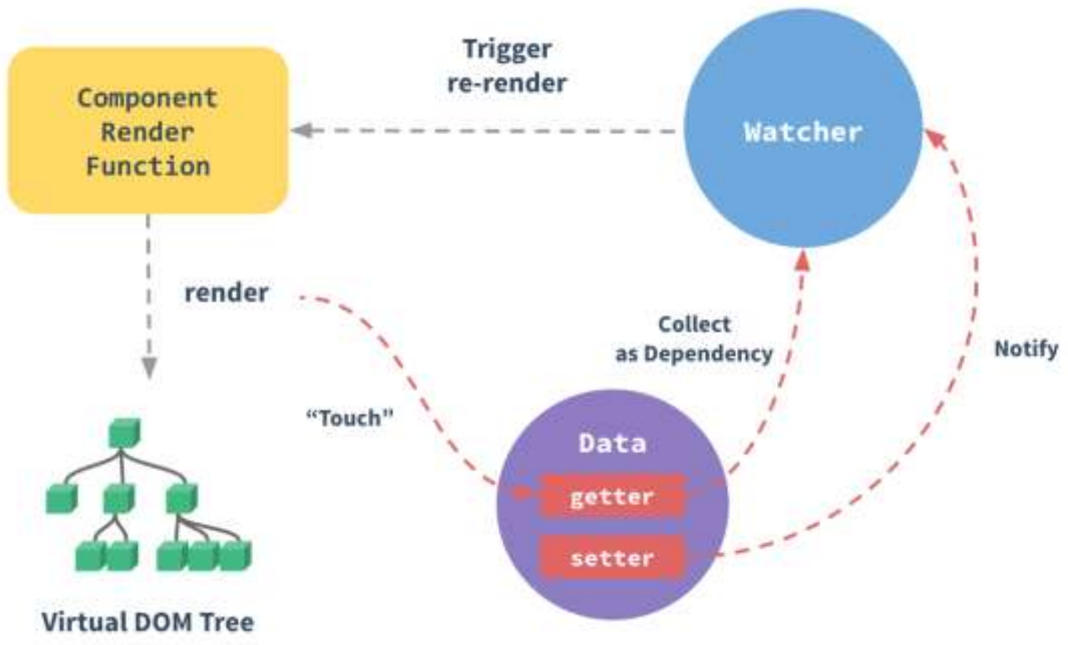
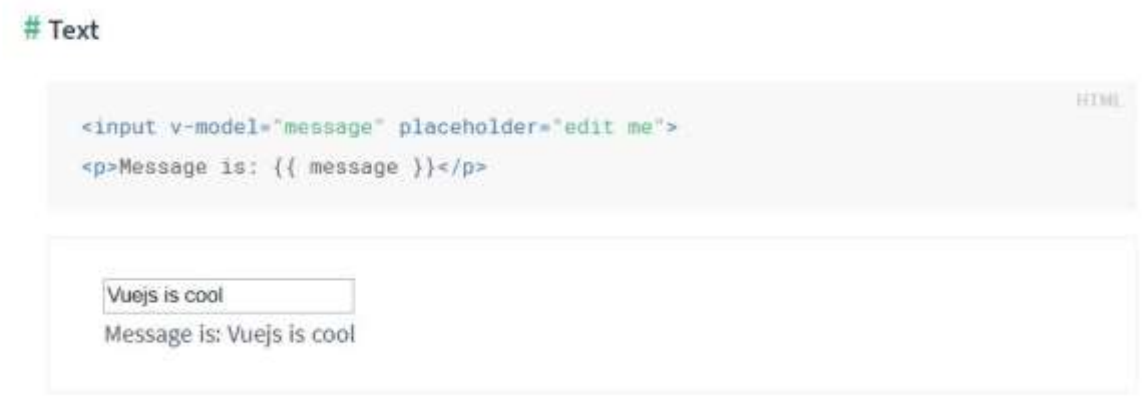


Figure 7 Reactivity process in Vue.js component (Vue.js 2020)

Two-way data binding

As being said above, two-way data binding is a compelling feature of the Vue.js framework. It is proved very useful in binding data in form inputs through the v-model directive. Figure 8 below shows an example of a two-way data binding in form using a v-model directive.


Text

```html
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

```
Vuejs is cool
Message is: Vuejs is cool
```

Figure 8 Two-way data binding example (Vue.js 2020)

The v-model directive binds whatever is written in it with the corresponding property in the "data" object. What this means is, whenever we write something inside the input, the property "message" of the "data" object is automatically updated equal to that value, and vice versa. This process allows state management in components much more comfortable and less time-consuming. The thesis explains how it is in detail in later chapters. In this example, we also encountered a new concept, a "Mustache" syntax, or double curly braces. The word "message" inside these braces is replaced with the value of the corresponding property in the "data" object. So whenever the value of the "message" property is changed, the {{ message }} is also replaced with that value. In figure 8, the word beneath the input shows its value, which results from this process.

The process of reactivity can be explained like this. V-model directive of the input element allows the corresponding property in the "data" object to update with the same input value every time users type something in the input. After observing the change of the property in the "data" object, the watcher of the component then trigger re-render to update the view with the latest data. It is why users always see from the example what they typed in the input in the paragraph underneath.

3.2.3 Vue.js Libraries

Vue.js alone can provide many impressive features, but developers can do much more with other Vue.js libraries. These libraries include Vuex, Vue Router, Vue Server Renderer, and many other libraries, but the thesis only presents Vuex and Vue Router.

Vuex

Vuex, as described from its official site, is a state management pattern and library for Vue.js application (Vuex 2020). It creates a central store that can be accessed by any component of the application. To ensure better state manageability and security, developers can create rules and methods which mutate the states of the Vuex store. This process ensures the data is only changed in a controlled way and with the given pattern to avoid data conflict and unwanted behavior. Regularly, Vue.js components pass data to other components through props. But when the application grows, this can becomes too complicated and very hard to maintain, as well as consumes more time. Vuex can help developers in this situation.

Vuex consists of three essential parts: the state, the view, and the actions (Vuex 2020). The state is the data of the application, which serves as the source of truth. The view is where we can see the data from the state and might also be where we can make manipulation to the state. Actions are the possible ways developers created to force predictable state changes. Actions usually consist of two steps, dispatching an action and committing a mutation. An action is used to perform arbitrary asynchronous operations, which can be an API call to server in the back-end of the application to retrieve, update, or delete data. After the operation is complete, the action method decides to commit a mutation that corresponds to the responses from the server. Mutations are where the Vuex states get changed (Gallagher 2019). A diagram from figure 9 below demonstrates the process of how Vuex works in Vue.js applications.

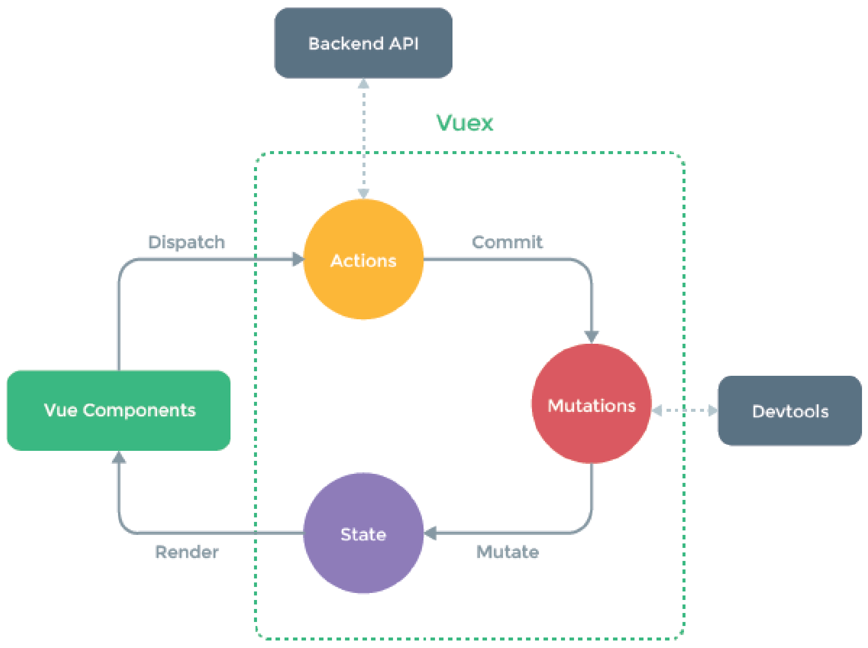


Figure 9 Vuex process in Vue.js application diagram

Vue Router

Another great library created for Vue.js application is Vue Router. Vue Router is a router library built primarily for Vue.js applications (Vue Router 2020). Since Vue.js does not have a built-in routing feature, another library is needed. This library allows developers to customize the URL pathing of their application and also control the navigation security of it. For example, developers can check if the user has logged in to let them go into the ap-plication dashboard URL, which otherwise is forbidden.

The official Vue Router site lists these features available from their product:

- Nested route/view mapping
- Modular, component-based router configuration
- Route params, query, wildcards
- View transition effects
- Fine-frained navigation control
- Links with automatic active CSS classes
- HTML5 history mode or hash mode
- Customizable Scroll Behavior

4 CASE DESCRIPTION

This chapter is about presenting readers with a study case. To be specific, this is a web application designed and developed by the author. Firstly, an introduction of the application, as well as its interfaces and features, is presented. After that, the author shows some of the tools he used to help him create the design and the project itself.

4.1 Reminder Application

4.1.1 Reminder Calendar

This sub-chapter focuses on demonstrating the web application. The application is designed and developed for a school project. Various tools available for free on the internet are also used to help push the effectiveness of the development process further. The application is powered by Vue.js and other libraries and frameworks such as Vuex, Vue Router, Buefy, Webpack. It is a Reminder application, from which users can use to create and manage plans or events in their daily life. A screen capture of the calendar interface can be found below.

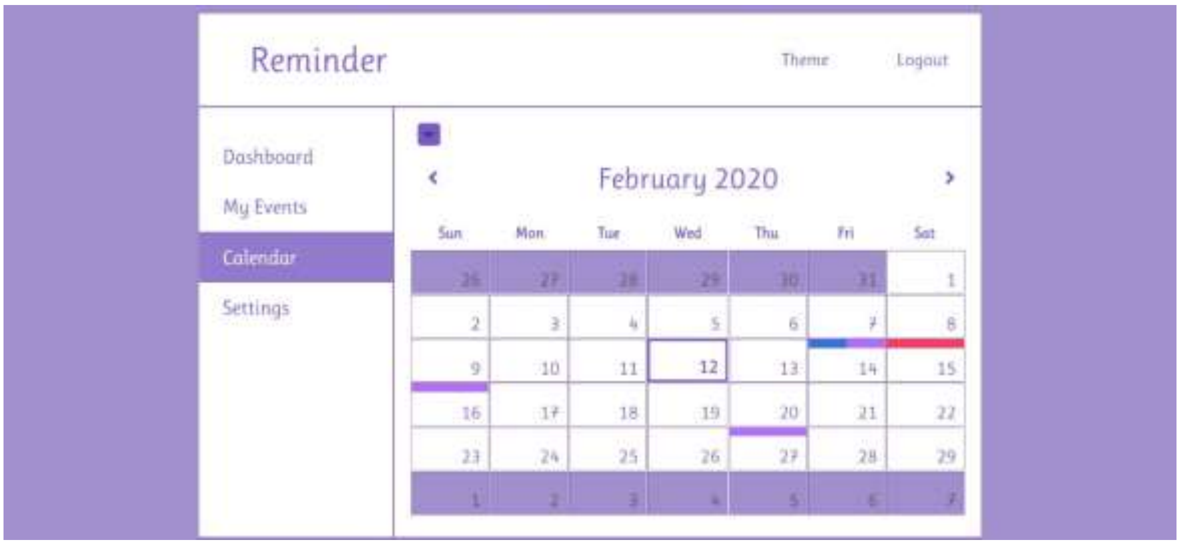
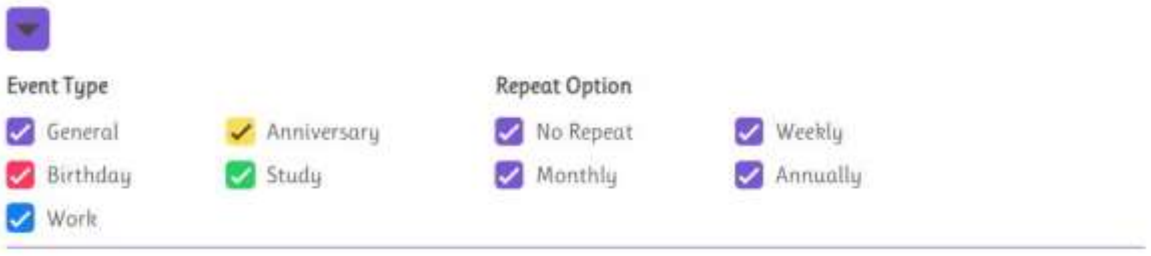


Figure 10 Reminder application calendar interface

The calendar interface is where users can find the events or plans they have created on the real calendar with its categories (color strips) show on the event's day grid. The application also has a filter from which helps users to filter events and plans to provide better manageability and personal preference.



Event Type

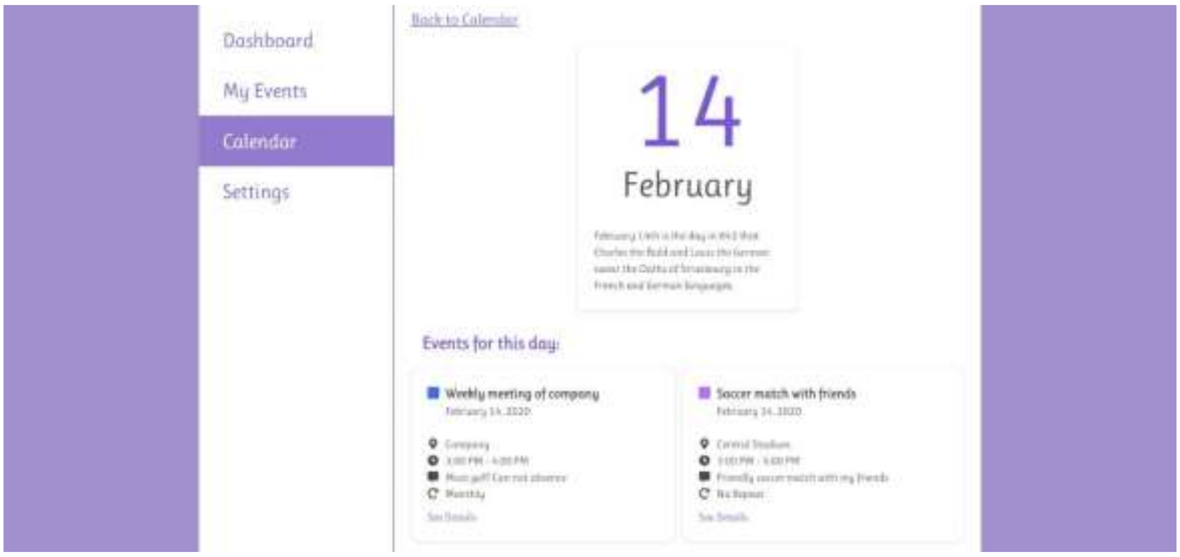
✓ General ✓ Anniversary
✓ Birthday ✓ Study
✓ Work

Repeat Option

✓ No Repeat ✓ Weekly
✓ Monthly ✓ Annually

Figure 11 Calendar filter options

As shown in figure 11, we can see that events also have their type and repeat option. These event types cannot be changed or customized, but the author is planning to implement this feature in later versions in the future. Each event type has its representing color, so users can easily distinguish events without having to look for its detail. Standing beside the event types are the repeat options. These options come in handy when users want to have their events automatically repeat after the same duration, such as weekly, monthly, or annually. Daily events are also available but do not show in the calendar because it is unnecessary and causes many distractions since these events appear every day on the calendar. Therefore, daily events are shown in a different place in the application, which is mentioned later in this sub-chapter. These additions are added to improve users' feel and comfort.



14
February

February 14th is the day in 842 that Charles the Bald and Louis the German swore the Oaths of Strasbourg in the French and German languages.

Events for this day:

Weekly meeting of company
February 14, 2020
Company
[illegible]
Must go!! Can not absence
Monthly
See Details

Soccer match with friends
February 14, 2020
Central Stadium
[illegible]
Friendly soccer match with my friends
No Repeat
See Details

Figure 12 Day details

From the calendar view, users can see any day details and events. Figure 12 shows an example of a day details interface.

4.1.2 Reminder Dashboard

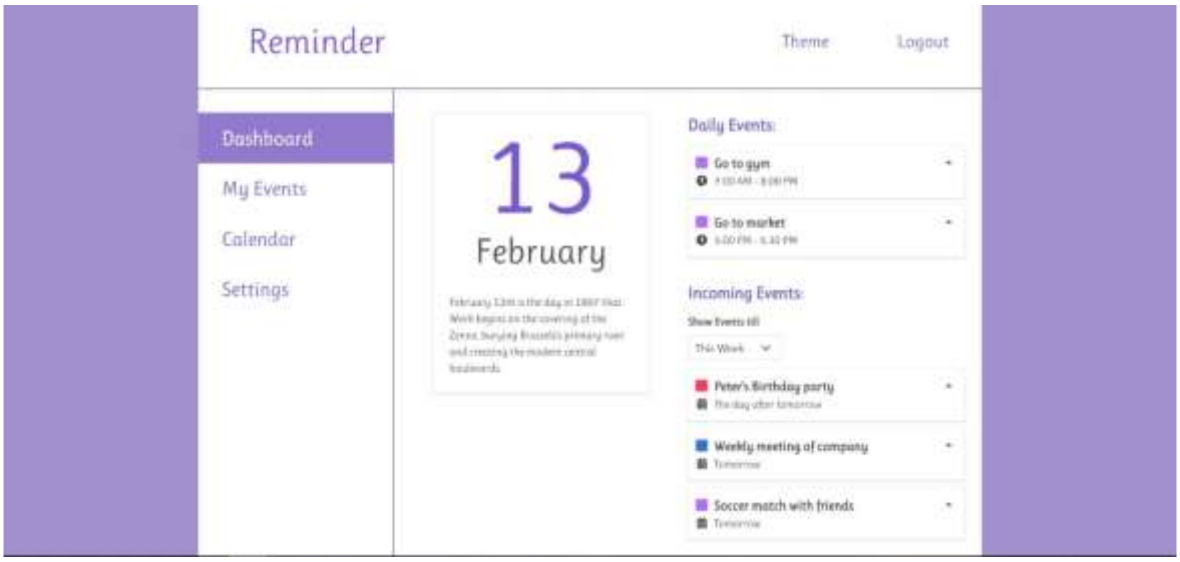


Figure 13 Reminder application dashboard interface

The dashboard is where users can see some general information about their events, plans, and today's fun fact. Daily events appear at the top right side of the dashboard. A list of incoming events also appears on the dashboard. Users can select how long in the future they want to see their events. These options can show events of this week, next week, this month, or next month. Repeatable events can appear many times in the selected duration, like weekly events, but the list only shows the nearest one of these events. Each event in these lists is an event card, which can be expanded to show more details of the event or closed to give a better overview of the list. Users can see these event cards appear in many places in the application.

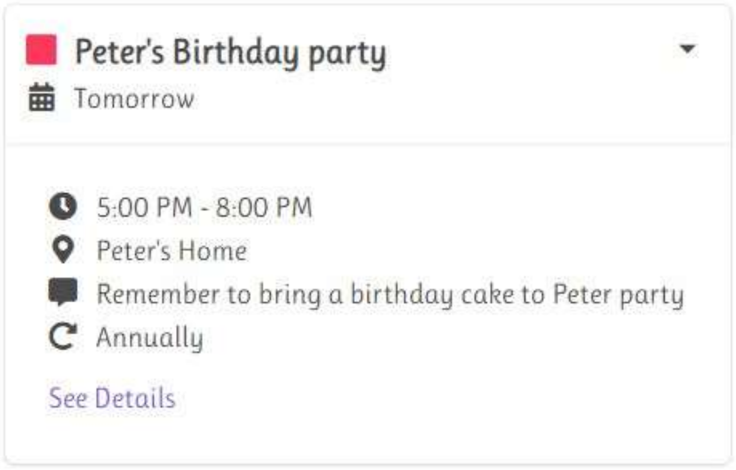
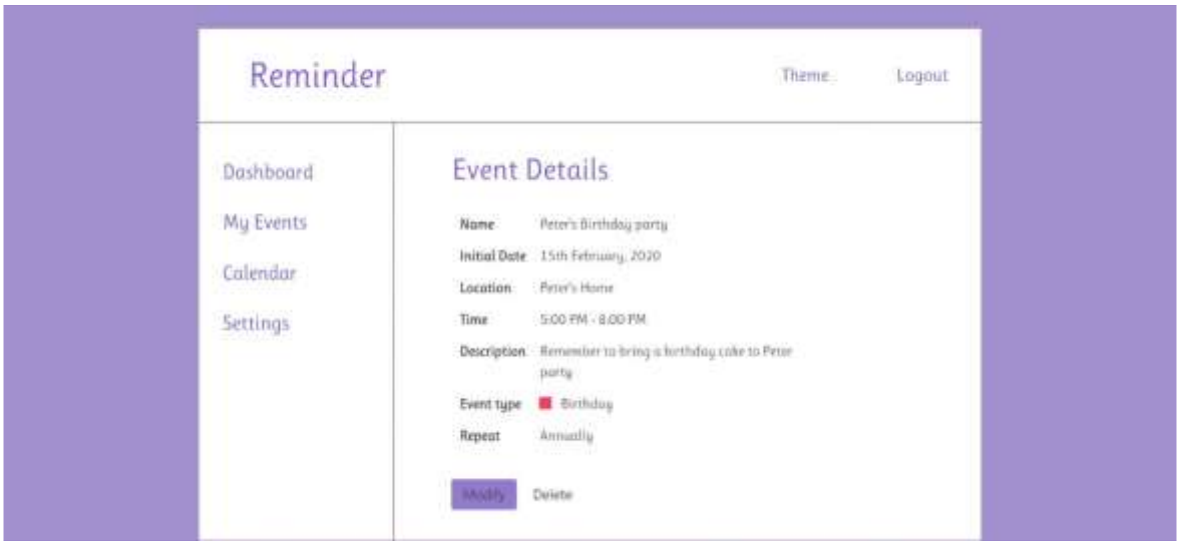

Peter's Birthday party
Tomorrow
5:00 PM - 8:00 PM
Peter's Home
Remember to bring a birthday cake to Peter party
Annually
See Details

Figure 14 Expanded event card

Figure 14 shows how an expanded event card looks. From the expanded view of the card, users can access to event details page to see all the event details and to modify or delete the event.

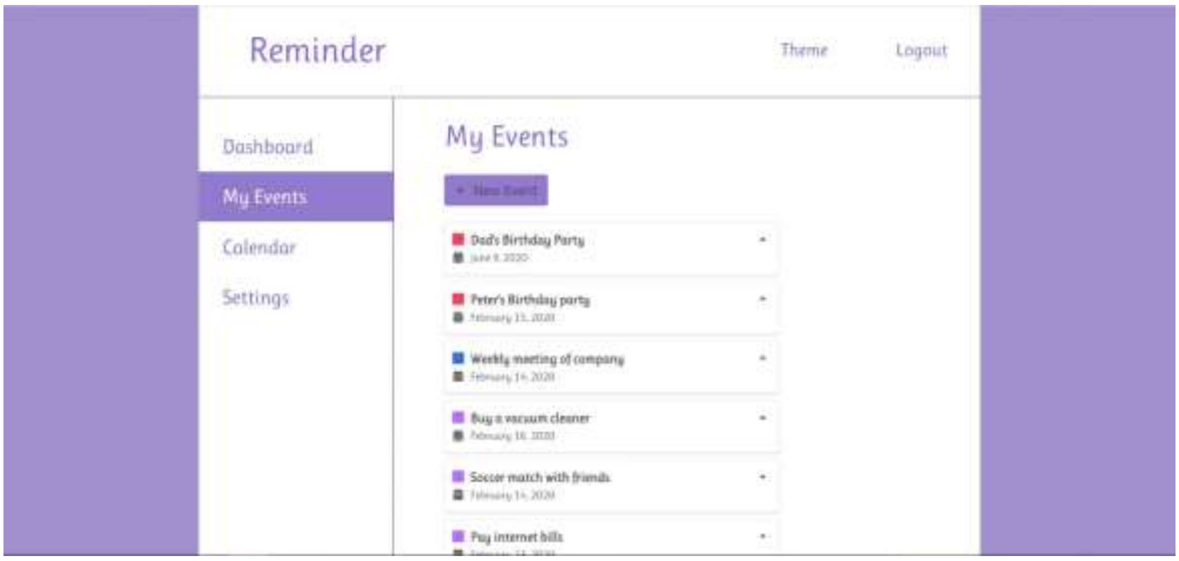


Event Details

Name	Peter's Birthday party
Initial Date	15th February, 2020
Location	Peter's Home
Time	5:00 PM - 8:00 PM
Description	Remember to bring a birthday cake to Peter party
Event type	🟥 Birthday
Repeat	Annually

Figure 15 Event details example

4.1.3 Event List and Form



My Events

- 🟥 Dad's Birthday Party — June 9, 2020
- 🟥 Peter's Birthday party — February 15, 2020
- 🟦 Weekly meeting of company — February 14, 2020
- 🟪 Buy a vacuum cleaner — February 16, 2020
- 🟪 Soccer match with friends — February 14, 2020
- 🟪 Pay internet bills

Figure 16 Event lists interface

The event list interface provides users with a full list of events, sorted in the order of event creation date. Each event here is also an event card like the one in the dashboard view. From here, users can create new events by going to another interface, an event form. Figure 17 below shows how an event form looks.

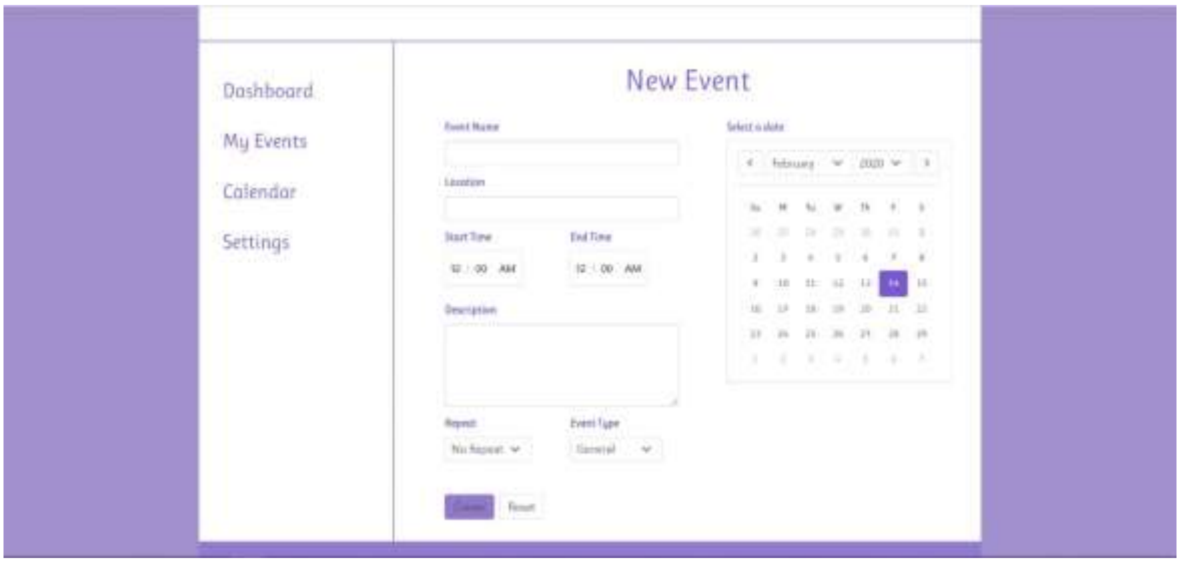
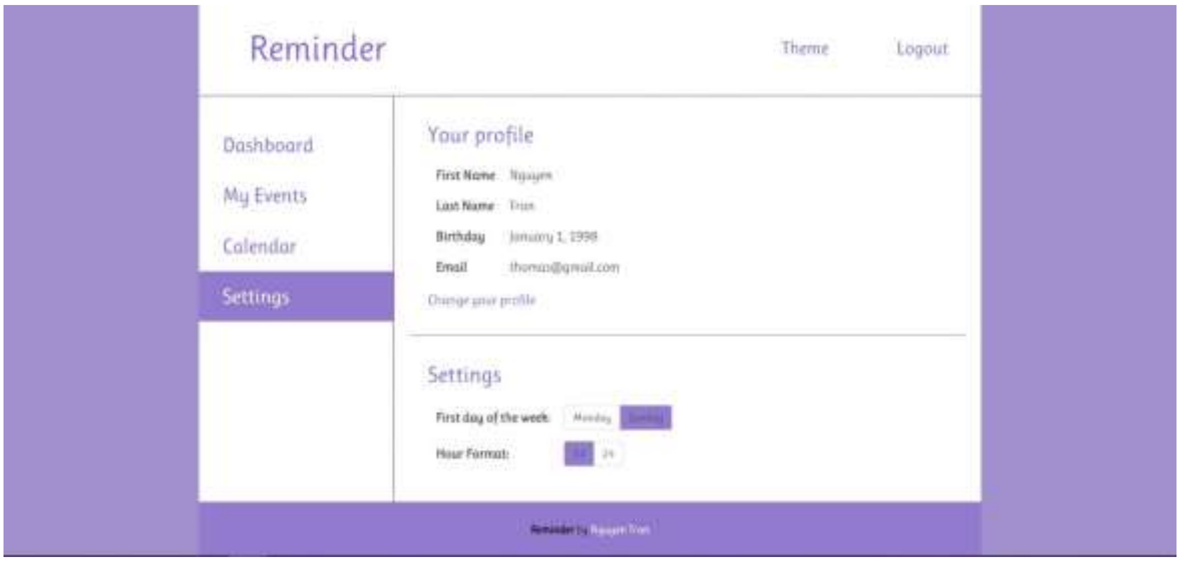


Figure 17 Event Form interface

From the event form, users can fill in their events information and select the event date, time, repeat option, and event type. The location and description field can be left empty, but a name is compulsory to create an event. Other fields use default values if users do not modify them. When modifying the events, users also use this form, but the titles and the methods that handle the form is different.

4.1.4 User and Application Settings

At the start, users have to create an account to use this application. After login in, users can choose to give other personal information such as first and last name, birthday, and email. These pieces of information are not necessary since there is no use for them at the moment. But the application is planned to be extended with other features where users might interact with each other. These features might be creating a group of users or creating shared events or plans. These are just merely plans, and there is no promise that they are available in the future. This Reminder application also provides two other settings. The first one is to set the first day of the week. It can be either Monday or Sunday, depends on user preference. This setting affects the calendar in both date pickers and the event calendar of the application. The other one is similar to the first settings, which is used to set time format between 12-hours and 24-hours format. Figure 18 below shows the settings interface of the application.



Your profile

First Name Nguyen
Last Name Tran
Birthday January 1, 1998
Email thomas@gmail.com

Settings

First day of the week: Monday [✓ Sunday]
Hour Format: [✓ 12] 24

Reminder by Nguyen Tran

Figure 18 User and application settings interface

These are an overview of some main features of the application. In the next sub-chapter, the author demonstrates some tools and support applications that helped him design and develop the application.

4.2 Other Tools and Support Applications

Draw.io

Draw.io is a platform used to create diagrams, charts, models, and much more (Draw.io 2020). The author uses this application to create the UML model for the database of the application. This platform is free to use and has many available templates to help with the structure design process.

Figma

Figma is a web application well-known for creating interface designs. This platform is a browser-based application but can also be a desktop application on Windows and Mac OS. Figma allows collaboration, which means designers can work together to create the same product (Bracey 2018). Below is a draft of the calendar interface created using Figma for the layout of the application.

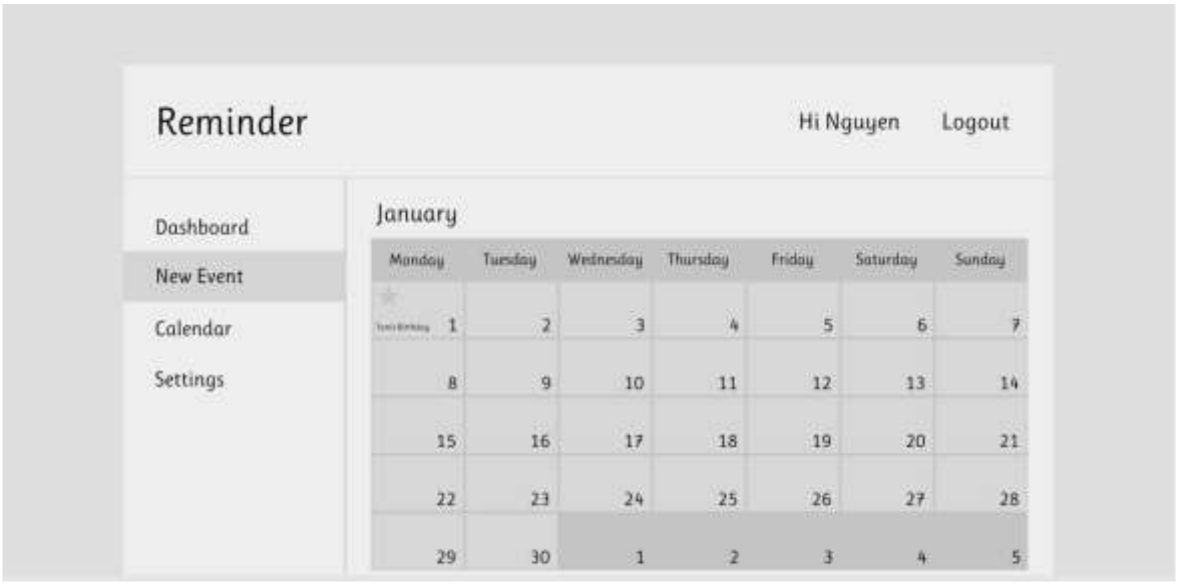


Figure 19 Calendar interface prototype designed on Figma

Clockify

Clockify is a free time tracker that runs on the browser. This product is also available on many desktops and mobile devices (Clockify 2020). The author uses this web application to track and keep records of how many hours he has invested in developing the application. The total time so far is around 150 hours.

This is the end of chapter 4. In the next chapter, the thesis presents the application from the code perspective. In other words, it shows how some Vue.js features help to create the project in the study case and how this implementation differs from using Vanilla JavaScript.

5 DATA COLLECTING AND ANALYZING

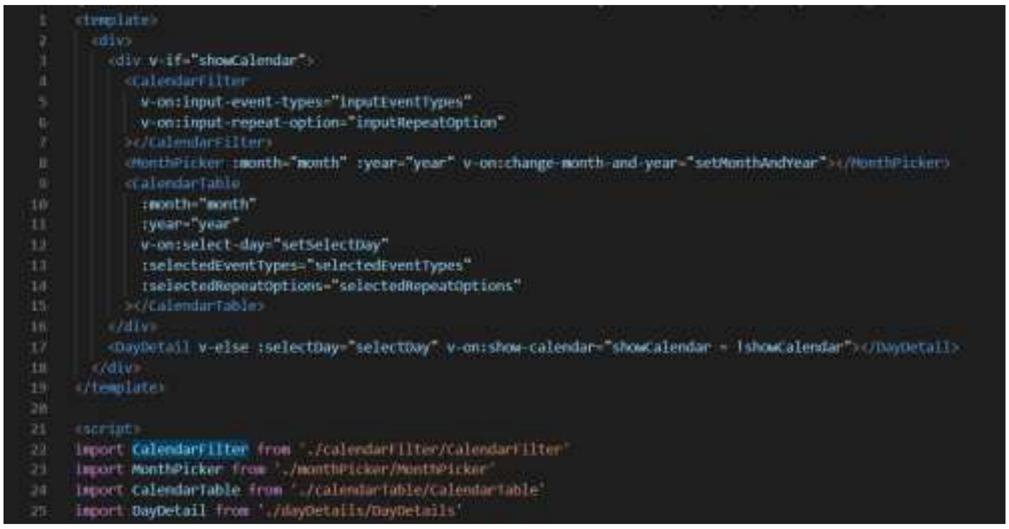
5.1 Data Collecting and Analyzing Process

To advance further into the research, technical or code perspective is used to examine and explain the study case. The process can be understood like this. Some features of Vue.js frameworks that this research proves they would bring some advantages to the development process are presented through code examples from the case study. After that, the thesis shows a similar implementation that produces the same effect but only written by Vanilla JavaScript, or pure JavaScript without using frameworks. Then, the comparisons between the examples are made to point out possible benefits brought by using the Vue.js framework in the project. But sometimes, when example implementations cannot be given because the amount code is large, or the example would be so complicated or inappropriate for the research purpose or skill level of the thesis. In those cases, articles and blogs are given instead to support the research's argument. The data collecting and analyzing processes are done simultaneously to give more direct, precise comparisons and results.

5.2 Collecting and Analyzing Data

Vue.js component vs. no component

As been explained in chapter 3, component architecture is a dominant feature of the Vue.js framework, especially single file components, where they include all three fundamental elements of web development (HTML, CSS, and JavaScript). Take a look from the example below; this is the code from the **CalendarLayout** component of the Reminder application. A component can be imported and used just like adding any HTML element, like **<CalendarTable>**, for example.



```
<template>
  <div>
    <div v-if="showCalendar">
      <CalendarFilter
        v-on:input-event-types="inputEventTypes"
        v-on:input-repeat-option="inputRepeatOption"
      ></CalendarFilter>
      <MonthPicker :month="month" :year="year" v-on:change-month-and-year="setMonthAndYear"></MonthPicker>
      <CalendarTable
        :month="month"
        :year="year"
        v-on:select-day="setSelectDay"
        :selectedEventTypes="selectedEventTypes"
        :selectedRepeatOptions="selectedRepeatOptions"
      ></CalendarTable>
    </div>
    <DayDetail v-else :selectDay="selectDay" v-on:show-calendar="showCalendar = !showCalendar"></DayDetail>
  </div>
</template>

<script>
import CalendarFilter from './calendarFilter/CalendarFilter'
import MonthPicker from './monthPicker/MonthPicker'
import CalendarTable from './calendarTable/CalendarTable'
import DayDetail from './dayDetails/DayDetails'
```

Figure 20 Component use and communication

In the author's opinion, component brings many conveniences and save a tremendous amount of time for developers because, after all, those frameworks are productions of many big companies like Facebook, Google, or Google's old employees (Duomly 2019). The logic behind creating these components is far beyond the author and the thesis's range of knowledge. Therefore, the thesis does not discuss how using the Vue.js component is better than building a component structure by plain JavaScript. Instead, a comparison between using components and not using them is a more suitable approach. The comparison is not about using the Vue.js component versus building and using the Vanilla JavaScript component anymore. But at least, developers save a large amount of time not to create a component structure themselves but use it from a framework or library.

So how can using components bring advantages to the development process? As explained in the theory chapter, components are reusable custom-made elements. Let us use the event cards as an example for this, these cards appear many times within the application, in many pages.

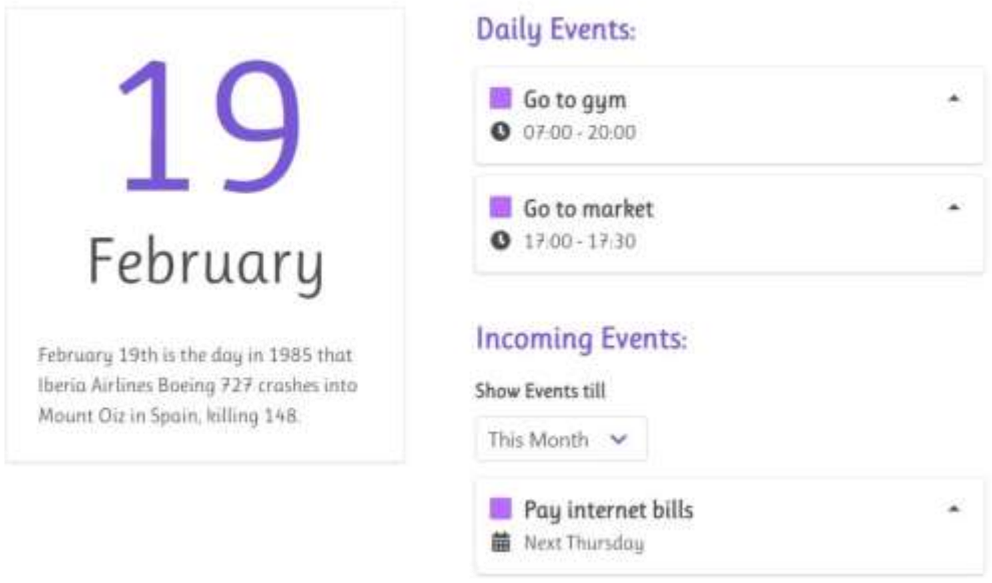
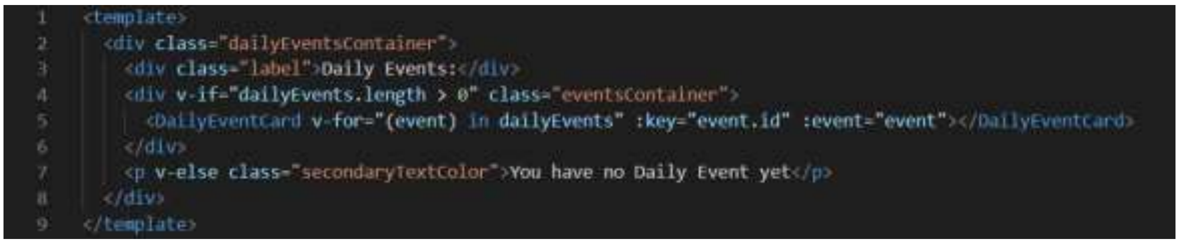


Figure 21 Event card appears many times within the dashboard

These event cards are almost identical in structural styling and functionality, except for the data of each card. All other functionalities of the cards are the same because they use one component to render. If these event cards are added manually in our application, then every time developers want to change something of the event cards, they have to apply it to all other event cards of the application. This repetitive process can be so frustrating and consumes a massive amount of time. Moreover, from the author's experiences, in the development process, changes are a crucial part and occurs a lot to reach the final product or desire. This reason means using components helps developers saving a lot of time doing the same works overtime and encourages them for not having to do tedious and laborious works.

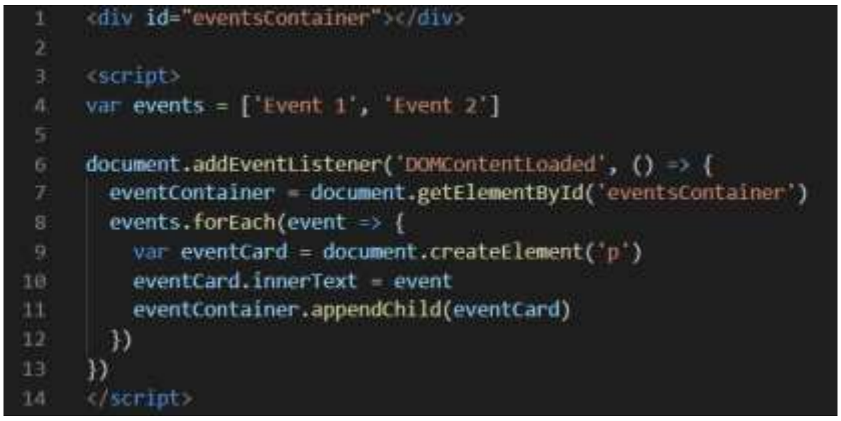
List rendering



```
<template>
  <div class="dailyEventsContainer">
    <div class="label">Daily Events:</div>
    <div v-if="dailyEvents.length > 0" class="eventsContainer">
      <DailyEventCard v-for="(event) in dailyEvents" :key="event.id" :event="event"></DailyEventCard>
    </div>
    <p v-else class="secondaryTextColor">You have no Daily Event yet</p>
  </div>
</template>
```

Figure 22 Daily event list component

As we can see from the daily event list in the dashboard, more than one card is rendered, but we only see one **<DailyEventCard>** element from the code. The reason behind this is because the element uses a particular directive from Vue.js, which is **v-for**. This directive allows list rendering, which means it takes an array of data, "dailyEvents" to be specific, and renders a **<DailyEventCard>** component for each item of the array. This directive helps create dynamic rendering with the use of HTML code only. On the other hand, creating dynamic functionality like this operation requires JavaScript intervention if not using any frameworks.



```
<div id="eventsContainer"></div>

<script>
var events = ['Event 1', 'Event 2']

document.addEventListener('DOMContentLoaded', () => {
  eventContainer = document.getElementById('eventsContainer')
  events.forEach(event => {
    var eventCard = document.createElement('p')
    eventCard.innerText = event
    eventContainer.appendChild(eventCard)
  })
})
</script>
```

Figure 23 List rendering with plain JavaScript

Take the implementation above, for example. This implementation uses the events array to create a paragraph for each item of the array. The paragraph's texts are the content of the item in the array. This example is just a simple implementation of how to use pure JavaScript to create the list rendering. The result is below.

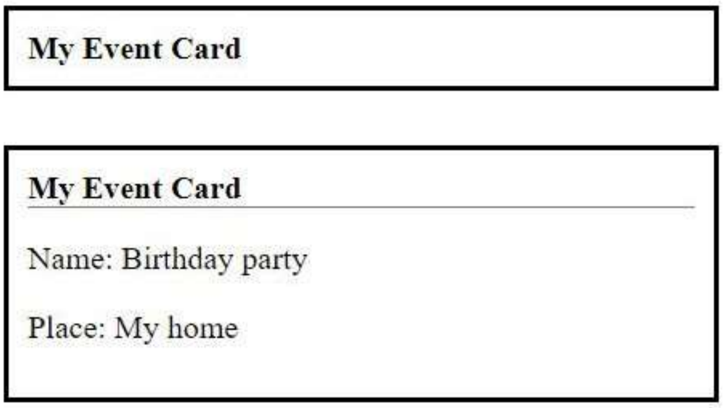
Event 1

Event 2

Figure 24 List rendering result

The result is very modest compared to the event card presented in the previous chapter. Despite the simplicity of the result, the amount of code needed is quite large and harder to read. Besides, this example only contains simple HTML paragraphs and does not have any other style or interactions implemented.

To manually render a complete event card with style and functions requires a lot of works and time. Look at another example implementation below.


My Event Card

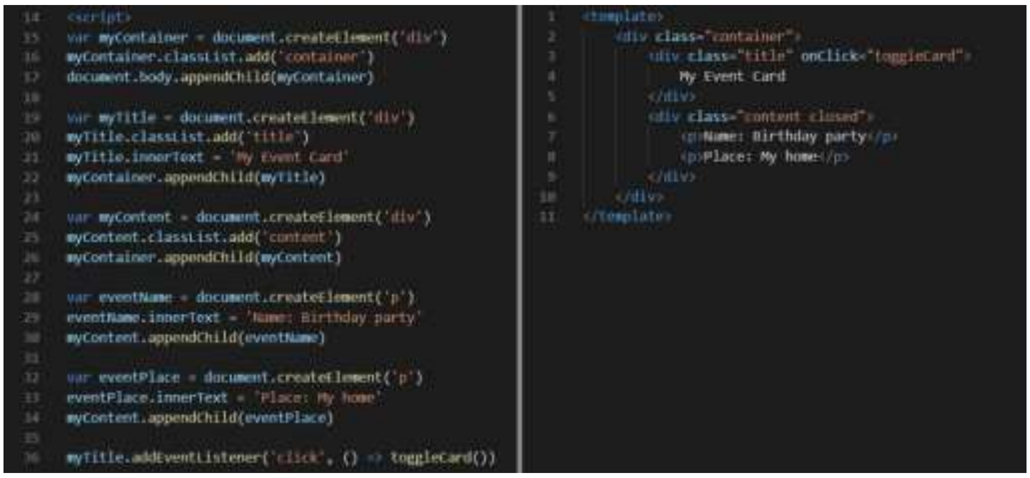
My Event Card

Name: Birthday party

Place: My home

Figure 25 Closed and opened event card example implementation

There are two ways to create this effect. The first one is through regular HTML (can be used as Vue.js components) and another one through pure JavaScript.


```
<script>
var myContainer = document.createElement('div')
myContainer.classList.add('container')
document.body.appendChild(myContainer)

var myTitle = document.createElement('div')
myTitle.classList.add('title')
myTitle.innerText = 'My Event Card'
myContainer.appendChild(myTitle)

var myContent = document.createElement('div')
myContent.classList.add('content')
myContainer.appendChild(myContent)

var eventName = document.createElement('p')
eventName.innerText = 'Name: Birthday party'
myContent.appendChild(eventName)

var eventPlace = document.createElement('p')
eventPlace.innerText = 'Place: My home'
myContent.appendChild(eventPlace)

myTitle.addEventListener('click', () => toggleCard())
```

```
<template>
    <div class="container">
        <div class="title" onClick="toggleCard">
            My Event Card
        </div>
        <div class="content closed">
            <p>Name: Birthday party</p>
            <p>Place: My home</p>
        </div>
    </div>
</template>
```

Figure 26 JavaScript (left) and HTML (right) implementation comparison

As can be seen in figure 26, the differences are quite significant, as HTML implementation only takes 11 lines of code, and JavaScript takes 22 lines. That is two times the work, not mention about JavaScript implementation's code line is longer and harder to visualize the structure like the HTML one. Therefore, the author believes that using HTML to perform dynamic operations like list rendering is a better choice. Besides, some specific processes (rendering event cards from arrays) like this require dynamic rendering from JavaScript, but Vue.js framework makes it possible to render from HTML. These advantages are more distinct as the application scales up and becomes more complex.

Reactivity

As have seen in the example above, using Vanilla JavaScript to render anything into the DOM is very frustrating. But things do not stop there; let us consider reactivity. The author already explained about reactivity in chapter 3. In short, reactivity helps update the view whenever the data changes. Figure 27 below presents a simple example of a counter.

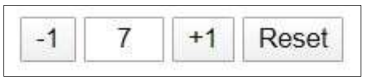
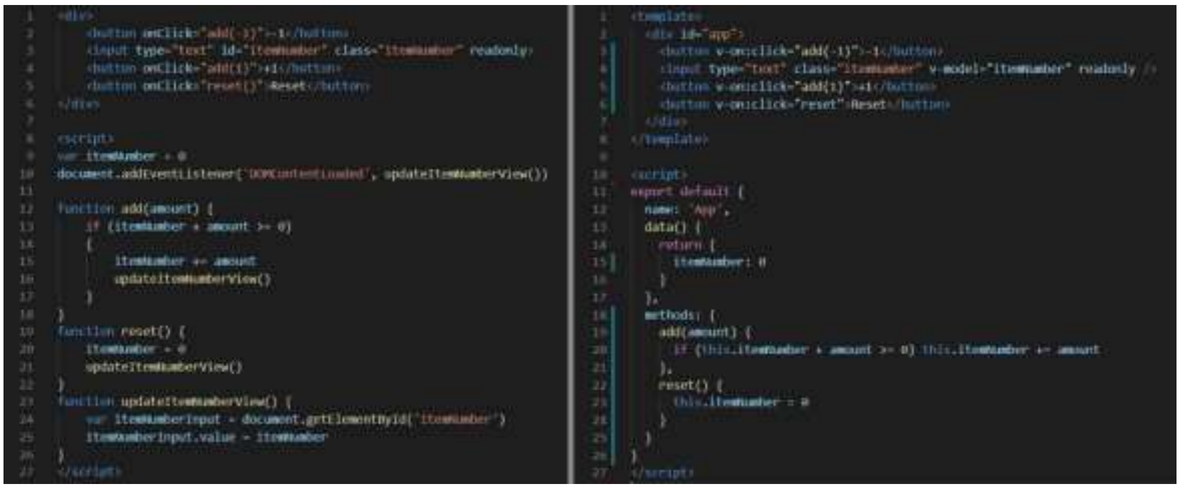


Figure 27 Simple counter mechanism example

This simple counter has an initial value of 0 and can add or subtract the number by 1 (not below 0) or reset the value back to its initial value. Below are the implementations of this example.



```
<div>
    <button onClick="add(-1)">-1</button>
    <input type="text" id="itemNumber" class="itemNumber" readonly>
    <button onClick="add(1)">+1</button>
    <button onClick="reset()">Reset</button>
</div>

<script>
var itemNumber = 0
document.addEventListener('DOMContentLoaded', updateItemNumberView())

function add(amount) {
    if (itemNumber + amount >= 0)
    {
        itemNumber += amount
        updateItemNumberView()
    }
}
function reset() {
    itemNumber = 0
    updateItemNumberView()
}
function updateItemNumberView() {
    var itemNumberInput = document.getElementById('itemNumber')
    itemNumberInput.value = itemNumber
}
</script>
```

```
<template>
  <div id="app">
    <button v-on:click="add(-1)">-1</button>
    <input type="text" class="itemNumber" v-model="itemNumber" readonly />
    <button v-on:click="add(1)">+1</button>
    <button v-on:click="reset">Reset</button>
  </div>
</template>

<script>
export default {
  name: 'App',
  data() {
    return {
      itemNumber: 0
    }
  },
  methods: {
    add(amount) {
      if (this.itemNumber + amount >= 0) this.itemNumber += amount
    },
    reset() {
      this.itemNumber = 0
    }
  }
}
</script>
```

Figure 28 Vanilla JavaScript (left) and Vue.js (right) implementation comparison

The author left out the CSS code because it is the same for both implementations. At the first look, these two implementations do not have significant differences. Both implementations have the same number of code lines and look quite similar to each other, especially since both have the same HTML structure and two functions with the same purpose: **add** and **reset** method. But in the Vanilla JavaScript implementation, it requires another function, **updateItemNumberView**, which updates the view. And every time the application call method **add** or **reset**, this method must be called again to ensure the view always shows the newest data.

Meanwhile, in the Vue.js implementation, the item number is kept in the data object, making it a reactive resource of the component. Every time the item number changes (by other operations or methods), the view automatically updates them. To show the data in the application, developers can use data binding (v-model) in forms or use mustache syntax to show any data in the data object. This operation makes sure the view always shows the latest data without setting them manually for each data. The process of updating view with pure JavaScript in the implementation above may look effortless and straightforward. But imagine this data appears many times in the view, and there are many pieces of data to take care of, not only one data in like this example. That is when Vue.js reactivity becomes a compelling feature for developers to save time and does not have to worry about this frustrating and tedious issue. An article written by Alberto Gimeno on Medium, titled "The deepest reason why modern JavaScript frameworks exist." mentions plenty of good reasons. Those reasons include having component-based architecture, active and large community, third party dependencies, browser extensions, and suitable for making SPAs (Single Page Application). But this article also points out that these are not the most important reasons. Instead, it lies upon a simple yet powerful and handy functionality, syncing the UI with the state (Gimeno 2018). In other words, it is what developers can achieve using Vue.js framework's **Reactivity**.

This paragraph marks the end of chapter 5. In the next chapter, the thesis concludes the studies and evaluations done in this chapter. After that, answers to the research questions at the start of the thesis are given. Additionally, other aspects of the study that haven't been covered, such as reliability, validation, limitation, and suggestions for future works, are also presented.

6 CONCLUSION

In this chapter, the thesis focus on concluding and giving answers to the research questions. After that, the thesis presents research reliability, validation, limitation, and suggestions for future studies to complete the conclusion of the thesis. The start of this chapter is the revision and concluding of the research questions.

6.1 Answering the Research Questions:

Comes back from the start of the thesis are the two research questions. These questions provide the topic and the guideline for the studies. These two questions are:

- What are some of the main features of Vue.js framework, and what are their uses?
- How do these features benefit the development process?

Based on the studies from chapter 5, **Component-based Architecture**, **List Rendering** (or dynamic rendering from components), and **Reactivity** are three of the many features provided by Vue.js framework. These selected features are also the answer to the first question. From the studies and evaluations in earlier chapters, the thesis tries to point out the positive impacts created by these functionalities, which are presented in the next paragraph.

Firstly, examples show Vue.js lessens the amount of code needed for specific tasks or operations. Besides, this framework also prevents developers from doing repetitive and frustrating tasks by using components. These benefits result in reducing the development time as well as keeping developers' morale high. Secondly, through the implementations, Vue.js provides better code visualization, structuralization, and readability with **Component-based** and **List Rendering** features. These effects can be translated in improving the project's manageability and effectiveness, which, if done right, can also lower development time. Finally, Vue.js allows developers to perform certain tasks easily, which very difficult to do without the help of the framework, especially for new developers. Using Vue.js **Reactivity** is an example; this helps the development process in various ways. It includes providing less time-consuming, better performance with minimum code, solving the most challenging issue in web development (data syncing). All of these advantages also becomes more significant when the application start scaling up. Generally, in some specific situations, these three features help the development process by creating better project manageability and scalability while also lessening the development time and providing better solutions for specific difficult issues in many projects. This conclusion is the answer to the second question.

6.2 Research Reliability and Validation

The thesis not only demonstrates some theories using trusted sources related to the topic of the study but also gives example implementations and articles to support the ideas of the research. Despite that, the thesis tries to improve reliability and validation in some other ways.

Firstly, When presenting a feature, many example implementations are also demonstrated and explained. All the examples are created and tested by the author before bringing them into the analysis. Tested can be made using the exact code to ensure the reliability and validation of the examples.

Secondly, the main subject of this research is the study case. The application project can be found from this link: https://github.com/tom2lua/reminder. The whole application can be found in this repository. The back-end of the application is not included because it is is not the subject of the thesis. For testing purposes, a development version can be run from this repository or using another link: https://amazing-shirley-0738ea.netlify.com. This link contains the production version of the project, which is built from the project. The application can be accessed directly in this second link without needing to set up the project.

6.3 Research Limitation

For the limitation of the research, as the thesis mentions in sub-chapter 3.2.2. The Vue.js framework version used in both the study case and example implementations is 2.0. This limitation can cause the result to differ in earlier or later versions of Vue.js framework. Another thing worth mentioning is the complexity and size of the developing application. The framework is only proved significant if the application has a medium or high complexity and size. Small and simple projects or applications might not gain benefit from these features. On the other hand, there can be some downsides from using them, such as following extra framework principles and rules to ensure the best performance. These things are worth taking into consideration when starting a new project to predict a better outcome.

6.4 Suggestions for Future Studies

Finally, some suggestions for future studies and researches that took the thesis for inspiration are given. Studies on other features of Vue.js can be sufficient since this research only working with three basic features. Other great topics can be focusing on in-depth analysis with these features to get a better understanding and optimize on how to use them. As the time the thesis is written, Vue.js version 3 is coming to release. This new update might become a proper subject for future studies as well.

This chapter consists of concluding the research questions and giving many other perspectives of the study, such as reliability, validation, and limitation. There are also some ideas for future works that got inspired by the thesis provided by the author. The next chapter summarizes the content of the study and marks the end of the thesis.

7 SUMMARY

The high demand for web applications nowadays leads to the rising need for many new developers. The thesis presents developers with fundamental theories of the required components of a web application. Other important definitions are JavaScript programming language and its libraries and frameworks. Since the thesis focus on studying about the front-end of the application, the study subject shifts onto Vue.js, one of the most well-known JavaScript framework for front-end development.

To get a better understanding of the framework, the thesis provides a study case of a Vue.js powered web application. In general, the study case is a Reminder web application where users can create plans to help remind them in the future. Mostly all the main features of the application are demonstrated in this chapter.

After the case description, three main features of Vue.js framework are presented and explained for the data collecting and analyzing. This process consists of comparisons of example implementations with and without applying Vue.js to find the differences in the approaches.

Conclusions are made from the analysis in the previous chapter to answer research questions. Additionally, other aspects of the study are given, such as research reliability, validation, limitation, and suggestions for possible future works.

LIST OF REFERENCES

AltexSoft. 2020. The Good and the Bad of Vue.js Framework Programming. [accessed 10 February]. AltexSoft. Available at: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

Benedict, G. 2018. Web Application vs. Website – What's the Difference? [accessed 5 February 2020]. Techuz. Available at: https://www.techuz.com/blog/web-application-vs-website-whats-the-difference/

Bhatia, M. 2018. Your Guide to Qualitative and Quantitative Data Analysis Methods [accessed 26 September 2019]. Humans of data. Available at: https://humansofdata.atlan.com/2018/09/qualitative-quantitative-data-analysis-methods/

Bracey, K. 2018. What is Figma? [accessed 11 February]. Envato Tuts+. Available at: https://webdesign.tutsplus.com/articles/what-is-figma--cms-32272

Brown, K. 2018. JavaScript: How Did It Get So Popular? [accessed 26 September 2019]. Codeacademy. Available at: https://news.codecademy.com/javascript-history-popularity/

Clockify. 2020. About us. [accessed 11 February]. Clockify. Available at: https://clockify.me/about-us

DeGroat. 2019. The History of JavaScript: Everything You Need to Know [accessed 26 September 2019]. Stringboard Blog. Available at: https://www.springboard.com/blog/history-of-javascript/

Dewebkiller. 2019. UX vs. UI in Depth. [accessed 7 February 2020]. Dewebkiller. Available at: https://www.dewebkiller.com/ux-vs-ui-in-depth/

Dionne, M. 2019. Reasons Why JavaScript is Omnipresent in Modern Development. [accessed 8 February 2020]. Snipcart. Available at: https://snipcart.com/blog/why-javascript-benefits

Draw.io. 2020. Online Diagram. [accessed 11 February]. Draw.io. Available at: https://drawio-app.com/

Duomly. 2019. The Best Front-end Framework to Learn in 2019 [accessed 26 September 2019]. DEV. Available at: https://dev.to/duomly/the-best-front-end-framework-to-learn-in-2019-dn7

Gabriel, D. 2013. Inductive and Deductive Approaches to Research [accessed 26 September 2019]. Dr Deborah Gabriel. Available at: https://deborahgabriel.com/2013/03/17/inductive-and-deductive-approaches-to-research/

Gallagher, M. 2019. Vuex Showdown: Mutations vs. Actions. [accessed 11 February]. LogRocket. Available at: https://blog.logrocket.com/vuex-showdown-mutations-vs-actions-f48f2f7df54b/

Gimeno, A. 2018. The Deepest Reason Why Modern JavaScript Frameworks Exist. [accessed 21 April 2020]. Available at: https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445

Hannah, J. 2019. What is a User Interface, And What Are the Elements that Comprise One? [accessed 6 February 2020]. CareerFoundry. Available at: https://careerfoundry.com/en/blog/ui-design/what-is-a-user-interface/

Lamprecht, E. 2019. The Difference Between UX and UI Design – A Layman's Guide. [accessed 6 February 2020]. CareerFoundry. Available at: https://careerfoundry.com/en/blog/ux-design/the-difference-between-ux-and-ui-design-a-laymans-guide/

McLeod, S. 2019. What's the Difference between Qualitative and Quantitative Research? [accessed 26 September 2019]. SimplyPsychology. Available at: https://www.simplypsychology.org/qualitative-quantitative.html

Miessler, D. 2019. The Difference Between Deductive and Inductive Reasoning [accessed 26 September 2019]. DanielMiessler. Available at: https://danielmiessler.com/blog/the-difference-between-deductive-and-inductive-reasoning/

Moerkerke, D. 2019. JavaScript Doesn't Need to Be Replaced. [accessed 8 February 2020]. Medium. Available at: https://medium.com/javascript-in-plain-english/javascript-doesnt-need-to-be-replaced-bd01e2f12d51

Morris, S. 2020. Tech 101: What Is a JavaScript Framework? Here's Everything You Need to Know. [accessed 8 February 2020]. Skillcrush. Available at: https://skillcrush.com/blog/what-is-a-javascript-framework/

Morris, S. 2020. The 10 Skills You Need to Land Your First Front End Developer Job. [accessed 6 February 2020]. Skillcrush. Available at: https://skillcrush.com/blog/front-end-developer-skills/

Multidots. 2019. Importance of Code Quality and Coding Standard in Software Development. [accessed 9 April 2020]. Medium. Available at:

https://medium.com/@multidots/importance-of-code-quality-and-coding-standard-in-software-development-260cb0367f1a

Oracle. 2020. What is Database. [accessed 8 February 2020]. Oracle. Available at: https://www.oracle.com/database/what-is-database.html

Ray, A. 2018. Front-end Developers v/s Back-End Developers- Let's Compare. [accessed 8 February 2020]. Capital Numbers. Available at: https://www.capitalnumbers.com/blog/front-end-developers-vs-back-end-developers-lets-compare/

SPEC INDIA. 2019. Why Node.js is Very Popular Among Fortune 500 Companies? [accessed 7 February 2020]. Medium. Available at: https://medium.com/quick-code/node-js-and-fortune-500-companies-fewer-efforts-more-rewards-282db19160c0

Steefkerk, R. 2019. Qualitative vs. Quantitative Research [accessed 26 September 2019]. Scribbr. Available at: https://www.scribbr.com/methodology/qualitative-quantitative-research/

Steward, L. 2019. Front End vs. Back End Development. [accessed 5 February 2020]. Course Report. Available at: https://www.coursereport.com/blog/front-end-development-vs-back-end-development-where-to-start

TechMagic. 2019. React vs. Angular vs. Vue.js – What to Choose in 2019? (updated) [accessed 26 September 2019]. Medium. Available at: https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d

Vue Router. 2020. Introduction. [accessed 11 February 2020]. Vue Router. Available at: https://router.vuejs.org/#introduction

Vue.js 2020. Single File Components. [accessed 9 February 2020]. Vue.js. Available at: https://vuejs.org/v2/guide/single-file-components.html

Vue.js. 2020. Component Options. [accessed 9 February 2020]. Vue.js. Available at: https://012.vuejs.org/api/options.html

Vue.js. 2020. Form Input Bindings. [accessed 10 February 2020]. Vue.js. Available at: https://vuejs.org/v2/guide/forms.html

Vue.js. 2020. Reactivity in Depth. [accessed 9 February 2020]. Vue.js. Available at: https://vuejs.org/v2/guide/reactivity.html

Vue.js. 2020. What is Vue.js? [accessed 9 February 2020]. Vue.js. Available at: https://vuejs.org/v2/guide/index.html#What-is-Vue-js

Vuex. 2020. What is Vuex? [accessed 10 February 2020]. Vuex. Available at: https://vuex.vuejs.org/#what-is-vuex

Wikipedia. 2020. Vue.js. [accessed 9 February 2020]. Wikipedia. Available at: https://en.wikipedia.org/wiki/Vue.js