The building of Web Pages

A comparative study of MERN and MEVN

MAIN FIELD: *Informatics*
AUTHOR: *Sonia Mianji Johnson*
SUPERVISOR: *Jasmin Jakupovic*
JÖNKÖPING February 2020

This final thesis has been carried out at the School of Engineering at Jönköping University within Informatics. The authors are responsible for the presented opinions, conclusions and results.

Examiner: Anders Adlemo
Supervisor: Jasmin Jakupovic
Scope: 15 hp (first-cycle education)
Date: 2020-03-01

Postal address: Box 1026 551 11 Jönköping
Visiting address: Gjuterigatan 5
Phone: 036-10 10 00

Abstract

The ecosystem of JavaScript is rapidly changing and regularly a new framework or library is introduced claiming to have better features than its predecessors. This makes the decision-making process in choosing one framework over the other very challenging for the developers. This study aims to give a guideline to the reader in the process of decision making by comparing the runtime performance of the MERN (MongoDB, Express, React.js and Node.js) and MEVN (V stands for Vue.js) stack as well as increasing their viability in the job market by giving them an insight into the Swedish-based companies preference of software stacks in building single page applications. To fulfill the purpose, an experiment is conducted to determine how fast the mentioned stacks perform in building a single page application. For the experiment two simple to-do applications are built with MERN and MEVN and loading time, adding time, updating and deleting time of the tasks are measured. According to the results gathered, both MERN and MEVN performed similarly when executing the CRUD operations in the experiment. It must be noted that the run-time performance comparison was based on DOM manipulation of text-only content and due to the limited time frame, no image rendering was included in the applications. Moreover, to be able to reveal the software stack trend among the Swedish-based companies, a survey study was conducted. Out of approximately 70 companies contacted, 12 responded. Due to the low number of respondents on the survey drawing conclusions from the survey and generalizing, the result was challenging. However, the results gathered show that all the respondents use either Vue.js or React.js or both as their client-side software though they are not always combined with Node.js and MongoDB. Other preferred server-side software that are used in combination with React.js or Vue.js are Java, Go and Django. Some of the main factors that the respondents pointed out that affects their choice of software was the ease of learning, the community behind the software, clients' need and availability of that specific software developers.

Keywords – JavaScript, Software Stack, Framework, Library, Performance Analysis, React.js, Vue.js, MongoDB, Node.js, Express.js

Terminology

BSON– Binary JavaScript Object Notation
CMS – Content Management System
CPU – Central Processing Unit
CRUD – Create, Read, Update, Delete
CSS – Cascading Style Sheet
DOM – Document Object Model
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
I/O – Input / Output
JSON – JavaScript Object Notation
JSX – JavaScript XML
MERN – MongoDB, Express, React.js, Node.js
MEVN – MongoDB, Express, Vue.js, Node.js
MPA – Multipage Application
MVC – Model, View, Controller
OS – Operating System
RDBMS – Relational Database Management System
SPA – Single Page Application
SQL – Structured Query Language
UI – User Interface
URI – Uniform Resource Identifier
URL – Uniform Resource Locator

Table of contents

1 Introduction

The chapter provides a background for the study and the problem area the study is addressing. Further, the purpose and the research questions are presented. The scope and delimitations of the study are also described. Lastly, the disposition of the thesis is outlined.

1.1 Background

JavaScript is a high-level, dynamic, programming language that is one of the most popular and widely used programming languages in the world today as most of the websites and web browsers use and support JavaScript (areknawo.com, 2019).

With JavaScript becoming very popular, many tools, libraries, and frameworks are built upon it over the years, that solves the common programming problems more easily and efficiently. Often the lifespan of the frameworks/libraries is not too long as a newer one gets to be introduced almost every 6 months claiming to have a better user interface development according to another study published on stack overflow website (Allen, 2018).

With the rapid changes in the tech industry regarding the frameworks and libraries, developers are forced to keep up and constantly learn new technologies which are commonly referred to as JavaScript Fatigue. Each of these technologies come with their advantages and disadvantages which makes the process of decision making when building a web application pretty hectic.

Angular.js, Backbone.js, Amber.js, Vue.js and React.js are a few very popular and commonly used frameworks and libraries with Vue.js and React.js having the lead positions. According to the survey published by the website ashlynolan.com, there has also been an increase of 4.3% in usage of Vue.js and React.js among the developers, compared to last years' survey study (ashlynolan.com, 2019). Having to choose one of these frameworks and libraries when creating a single page web application, questions arise such as what aspects a developer should consider when choosing one of these frameworks or libraries. As performance is one of the main technical aspects, which of these technologies have a better run-time performance? Which technologies are more commonly combined with these frontend frameworks and libraries and why are they preferred?

As it will exceed the scope of this thesis to explore all the technologies built on JavaScript over the years, this study will focus on comparing the run-time performance of two of the most popular JavaScript libraries React.js and Vue.js in combination with MongoDB (a NoSQL database), Node.js (a server-side run-time environment) and Express (a Node.js framework). The reason behind this choice of stacks in this thesis is the knowledge gap identified by researching the previous studies. Although previous studies are addressing a range of overlapping topics, there are gaps in these inquiries that should be addressed to provide a more complete comparative overview of the software stacks in terms of performance, most notably between React.js and Vue.js, which will be the primary topic of this study. Moreover, the reason Vue.js and React.js was chosen to be combined with MongoDB, Node.js and Express are that the mentioned software is all programmed throughout using JavaScript and in order to get valid data in the performance analysis of Vue.js and React.js, the other parameters need to be kept the same. In continuation these two stacks will be referred to by their abbreviations, MERN (MongoDB, Express, React.js, Node.js) and MEVN (MongoDB, Express, Vue.js, Node.js)

Furthermore, this thesis will cover the preferential trend between the software stacks among industry practitioners in Sweden to further understanding of trends and relevance between the various stacks.

1.2 Purpose and research questions

This thesis intends to provide a guidance to the reader in the process of decision making for the development of a single page web application as well as increasing the readers' viability in the job market.

The research questions that this paper aims to answer are as the followings:

1. How fast do MERN and MEVN stacks perform when building a single page application in comparison to one another?

2. What are the preferred software stacks for single page application development among Swedish based companies?

To answer these questions and thereby fulfill the purpose, an experimental study will be conducted to analyze the run-time performance of the mentioned stacks in regard to building a single page application. In addition, a survey study will be conducted to gather primary data from industry practitioners regarding the preferential trend in software combination and usage.

1.3 The scope and delimitations

Due to the limited timeframe and to fulfill the purpose of this thesis, the scope is limited to comparing stacks MERN and MEVN. The comparison will be measured according to their run-time performance in building a single page application. This study is delimited to not include any other software stack in the comparison as it is focused on comparing the two most popular client-side software. Moreover, as measuring the run-time performance of an application from all the possible aspects is a very tedious process and due to the limited scope of this study, the performance comparison will be focused on the Document Object Model (DOM) manipulation based on text-only content.

In addition, a survey study is conducted to find out the trend among the industry practitioners and it is limited geographically to companies located in Sweden. Participants of the survey are the software developers that currently are working at companies in Sweden.

1.4 Disposition

This report follows the Jönköping University's School of Engineering thesis report template and is divided into the following chapters.

- Technical background
- Method and implementation
- Findings and analysis
- Discussions and conclusions

2 Technical Background

This chapter provides explanation to the building of web applications as well as describing each of the technologies used in the respective stacks, MERN and MEVN.

2.1 JavaScript

JavaScript is a high-level, dynamic, object-oriented programming language that was developed by Brenden Eich in 1995. It is one of the most popular and widely used programming languages in the world today as it is shown in a recent study published by the website stack overflow in Figure 1 -Most Popular Technologies (stackoverflow.com 2019). JavaScript was traditionally only used for manipulating Document Object Model (DOM) and Cascading Style Sheets (CSS) and was known to be a client-side scripting language. In 2009 an engineer called Ryan Dahl developed a JavaScript run-time environment named Node.js that can be used to run server-side applications.

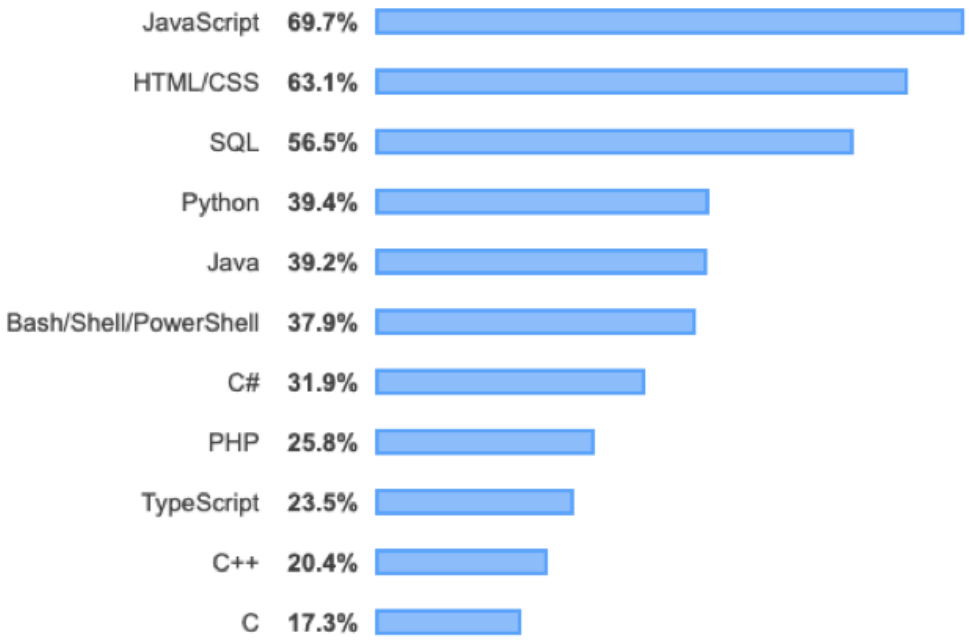


Figure 1 -Most Popular Technologies (stackoverflow.com 2019).

2.2 Software stack

A software stack is a number of independent software that are brought together in to support the execution of an application. Different software stack may be put together to support different applications depending on the requirements. For example, the software stacks for a web application may include client-side software, a server-side runtime environment, and a server-side database that communicates and work together through a set of complex instructions. In the hierarchy of a software stack, the lower level typically interacts with the hardware and the higher level executes the tasks in the client-side (Margaret Rouse 2019).

2.3 Multiple page application vs Single page application

There are two ways of building a web application. Multi-page applications (MPA) and Single page applications (SPA). MPA refreshes the entire page every time the user interacts with the page, forcing the server to handle a high number of requests. SPA, on the other hand, fetches data on-demand and dynamically updates the content, and the HTML (HyperText Markup Language) layout is previously loaded from the earliest request. Although the number of MPA on the internet today is much higher than the SPA, this is actually changing, and SPA is becoming more popular because of the enhanced user experience and lighter server-calls which are some of the main advantages of SPA. Building a SPA, a developer needs to choose an appropriate client-side software that will reduce the cost of the development. Vue.js, Angular

and React.js are just a few examples of JavaScript client-side software that are popular today (Kryzhanovska 2019).

2.4 MERN Stack

The MERN Stack is an acronym for MongoDB, Express, React.js and Node.js which are all open-source software optimized to build a SPA. Together the stack consists of a database, a back-end framework, a front-end library and a server-side runtime environment respectively. Using Node.js allows the MERN stack to be very efficient as a hosting web service due to being non-blocking, meaning it is able to serve a great number of clients concurrently. Each of these technologies used in the MERN stack will be explained in detail further below in this section. MERN stack is programmed throughout using JavaScript Language meaning the developer needs to feel conformable using only one programming language. Aside from the obvious advantage of not having to switch language when working on the different parts of the application, it also makes the code sharing (logic, validation, etc.) throughout the application possible. See Figure 2 for MERN stack architecture.

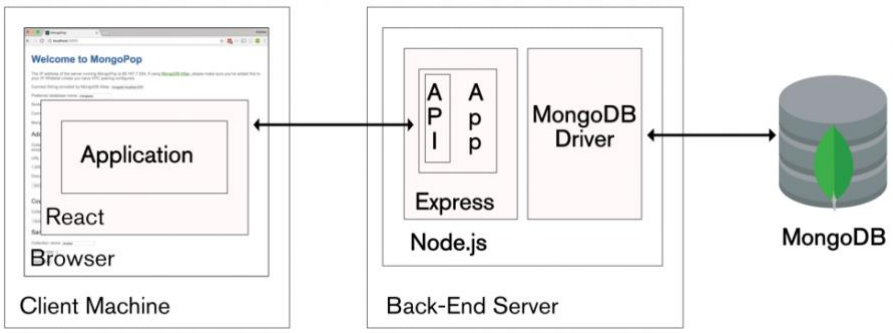


Figure 2 -MERN Stack Architecture – (Clusterdb.com 2019).

2.4.1 MongoDB

MongoDB is a NoSQL (non SQL), non-relational, document-oriented database, that is used by the MERN stack. MongoDB stores the data into collections of documents. Documents are collections of key-value pairs that can store various datatypes. Collections are a way for MongoDB to organize and keep track of the documents in a collection and also prevent name-overlapping. Every document in MongoDB has a unique Object Identifier that is automatically generated through which the document can be accessed. The data is stored natively in the form of BSON (Binary JavaScript Object Notation). The reason for that is that MongoDB supports inserting various datatypes in different programming languages, given the driver for the language is supported. Then the driver converts the different data types used in the applications (which can be written in various programming languages) to BSON types. This design allows a faster scan when querying and increased efficiency in data storage (MONGODB.COM 2019).

2.4.1.1 Querying

MongoDB is based on the JavaScript Language and the query language is based on JSON which is an abbreviation for (JavaScript Object Notation). Using JSON as the query language brings a considerable amount of ease of use in terms of querying. By specifying the type of operation in a JSON object a document can be created, deleted, retrieved, updated or searched for and allows the developer to construct more programmatic queries. SQL database on the other hand is more limiting when it comes to programmatic queries since the query language is "English-like" and a specific record can be reached using query words such as SELECT or WHERE. Storing the data in an object rather than storing it in a form of relations or tables as it is in a more traditional SQL database, gives the developer some flexibility on structuring models (Subramanian 2019 p. 10-11).

2.4.1.2 No predefined Schemas

No predefined schemas mean that there is no need to specify the data type or sizes before storing the data. This allows the developer to test and iterate without the need of restructuring the model every time, a new change is made. Unlike an SQL database that the model has to be modified for a change to be accepted. Therefore, the development in MongoDB becomes faster. The downside with this is that it could potentially introduce some other issues since the datasets are less organized. If a more structured dataset environment is preferred or required, Mongoose, an express MongoDB database driver can be used that emulate a similar structured environment as SQL.

2.4.1.3 Complex data models

In MongoDB a document can be seen as an equivalent of rows and a collection or a multi-document can be an equivalent of a table in the SQL database. MongoDB supports arrays and nested objects, allowing the developer to have complex hierarchical models in one document.

2.4.1.4 Scalability

When it comes to data sizes and scalability of an application, developers have two options, either to scale up meaning they have to invest on a bigger machine that is capable of storing more data or scale out which is a term used for distributing the data across multiple machines. Scaling up can be very expensive and can eventually reach up to the hardware's physical limit where purchasing another big machine can be too costly. MongoDB is designed to scale-out, it partitions the data across multiple servers and automatically balances the data load. Mongo's routing system is used to assure that the data is read and written to the correct machine. This design provides a more sustainable scalability option as more servers and storage space can be added as needed (Bradshaw 2019).

2.4.2 Express

Express.js or simply Express is a Node.js framework originally built by TJ Holowaychuk It simplifies building server-side code. Although this technically, could be done with Node.js alone, Express helps the developer to achieve the same result with less and cleaner code while having access to additional functionalities. Express can essentially be described as a set of middleware that is built to manage all responses expected of a web-application such as response codes, setting cookies or sending custom headers. While Express is lightweight and has a limited set of built-in middleware, the developer can customize their own sets to fit their specific needs or choose to use middleware from third parties (Subramanian 2019 p. 8-10).

2.4.2.1 Express Routing

A prominent feature of the framework is routing, the specification of how to process HTTP-request matching certain patterns in the application's endpoints (URIs). It essentially parses the request URL, headers, parameters, and responds as specified (Subramanian 2019 p. 8-10).

2.4.2.2 Template Engine

Express Template engines enables the use of static template files in the application. The variables in the templates are replaced by the actual values and transformed into the HTML file before sending it back to the client. This is to offer an easier way of designing the HTML page. Express uses the "Jade" template engine by default, but there are a few others that can be integrated such as Pug and Mustache (expressjs.com 2019).

2.4.3 React.js

React.js is the client-side software used in the MERN stack. It is an opensource, JavaScript component-based library that was developed by Facebook in 2011. Unlike other JavaScript libraries such as jQuery that is used to create an MVC (Model, View, Controller) pattern, React.js creates only views in HTML. In MVC architecture, the Model manages the application's data, View renders the model for the client and Controller updates the model when there is a change. In a regular MVC model, Views listen to the models and get updated according to the changes accruing in the model. Using MVC pattern, it can be more difficult to maintain since every time there is a small change in the view, the model gets updated and that update might

cause another update in the view. That is why React.js was developed in the first place, to prevent the struggle of dealing with the transitions of the updates (Subramanian 2019 p. 5-6).

2.4.3.1 Declarative

Views in React.js are declarative. A declarative view allows the code to be more predictable and easier to maintain since the developer only needs to design the component for each view and React.js efficiently updates only the component that the change has occurred in. This is done with the help of virtual DOM (Document Object Model) which is an in-memory data structure that is built to represent the data when the view is declared. React.js compares the virtual DOM with the actual DOM and if there is a difference then the actual DOM is updated accordingly. Compared to the alternative way of updating the DOM as it is in jQuery, the computation of the difference is more costly, whereas React.js offers an optimized way to perform minor changes (Subramanian 2019 p. 5-6).

2.4.3.2 Component-based

Components are the building blocks of React.js which are smaller and reusable code snippets that maintain their own state and renders. Technically, they are JavaScript classes or functions that are capable of receiving inputs and properties (props) and return a React.js element. Each of these React.js elements describe how the UI (User interface) should look. Other components can be put together to form a parent component which can represent a full view or a page in an application (Kaga, 2018). See Figure 3 for the illustration of React.js Component.

Figure 3 - React.js Components - (Edureka.com 2019).

2.4.3.3 Data exchanges and lifecycles

The data can be exchanged between the components using properties (props) and it can be stored in the components state. Every component has its own lifecycle which can be seen as a series of events that the component goes through from when it is mounted to when it is destroyed. The data in the component can be manipulated as needed during the lifecycles. Figure 4 shows different React.js lifecycle methods and when they are invoked.

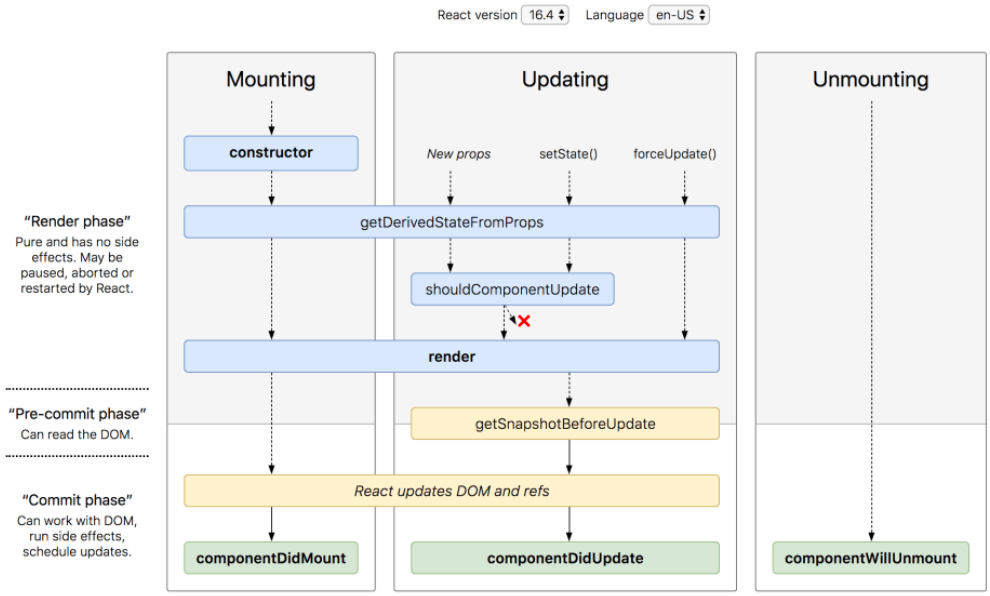


Figure 4 React.js lifecycle methods - (reactjs.org 2019).

2.4.4 Node.js

Node.js is a JavaScript runtime environment. JavaScript itself, is only a computational language, to execute the JavaScript code on the server-side, a runtime environment is needed. Node.js runtime environment consists of additional functionalities to JavaScript including access to the file system (read and write the files in the file system with JavaScript) and access to the network (using JavaScript, Http requests can be sent, and responses received). That means Node.js can execute JavaScript code that was written in an environment where it has access to the functionalities that are not natively part of the JavaScript language (Subramanian 2019 p. 8-10).

Node.js is built on Chrome's V8 engine (an open-source runtime engine that is written in C++) and can run on Linux, Mac OSX and Microsoft Windows systems after 2008R2/Windows 2012. Moreover, any other language that compiles into JavaScript before execution can be run by Node.js. TypeScript, CoffeeScript and Dart are a few examples of many (Patel 2018).

2.4.4.1 Node.js Modules

To reuse or combine multiple JavaScript files, Node.js makes use of its own model that is based on Common.js. Node.js modules allow a JS file to be exported and imported in another file as long as the module specifications are followed. Node.js modules allow the developer to split the codes into smaller and reusable chunks and offer better structure and organization of code. These modules can be seen as libraries that use the functions written in another JavaScript file. Node.js includes a few built-in modules that have access to the file system and network system. Other third-party libraries can be downloaded and installed using NPM (Node Package Manager) which is a default package manager for Node. (Subramanian 2019 p. 8-10)

2.4.4.2 Event Driven

Node.js uses an event-driven, non-blocking I/O (inputs/outputs), asynchronous model to perform multitasking. I/O events refer to any events related to the modification of local files or handlings of Http requests. Other languages used in the server-side, make use of threading to imitate simultaneous code execution. For example, when the server receives 2 requests of A and B, the request B is put on hold (it is blocked) and once the request A is handled, only then request B will be carried out and the response of both requests will be sent back to the client. Node.js, on the other hand, is non-blocking and uses callbacks to keep track of the functions that are pending. In the example mentioned above, Node.js would process both requests concurrently as illustrated in Figure 5, omitting the waiting time and resulting in an efficient, lightweight development (Subramanian 2019 p. 8-10).

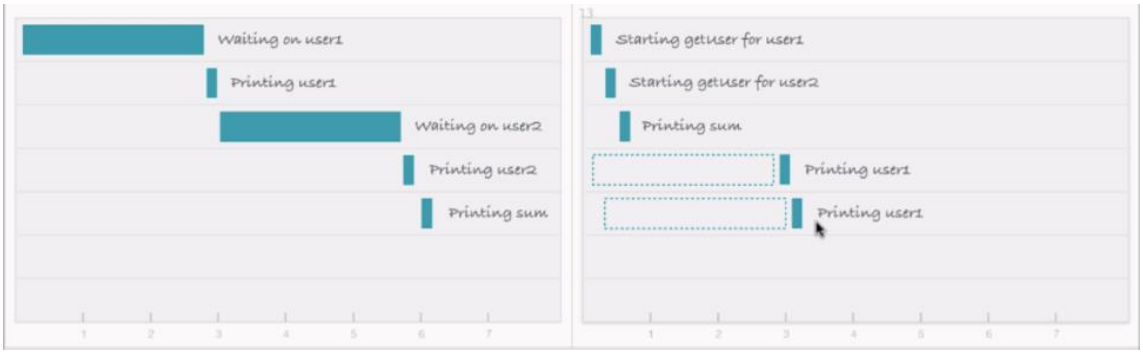


Figure 5 Blocking I/O vs Non-Blocking I/O (Patel 2018).

2.5 MEVN Stack

The MEVN stack is another software stack combination that uses the same technologies as MERN with the only difference being in the client-side software. MEVN uses Vue.js for the client-side software which is a JavaScript framework developed by Evan You in 2014. Evan You is a former Google developer who was working on Angular.js (another client-side software framework). While working on developing Vue.js, Evan You focused on omitting the complexities that developers had while learning Angular.js and wished to create a more lightweight framework. Currently, Vue.js is maintained by a community of international developers (Mobilunity.com 2020).

2.5.1 Vue.js

Vue.js is a progressive JavaScript framework that is used to build dynamic user interfaces and when combined with other modern tooling it can be used for creating effective SPAs. Very similar to React.js, Vue.js' core library is also focused on the view layer only (Vuejs.org 2020).

2.5.1.1 Two-way data binding

Vue.js supports two-way data binding by keeping the Model and the view as in MVC pattern synchronized. However, it does not check for view changes at regular time intervals but rather uses methods to watch for the changes made in the view and renders it accordingly (Mobilunity.com 2020).

2.5.1.2 Components in Vue.js

Very similar to React.js Vue.js also uses components which are reusable instances. These instances go through a number of initialization steps including, data set up observation, compiling the template. In addition, it also runs life-cycle hooks that allow the developer to execute their own code at different stages. Some of the main lifecycle hooks in Vue.js include created, mounted, updated and destroyed. See Figure 6 for Vue.js lifecycle diagram.

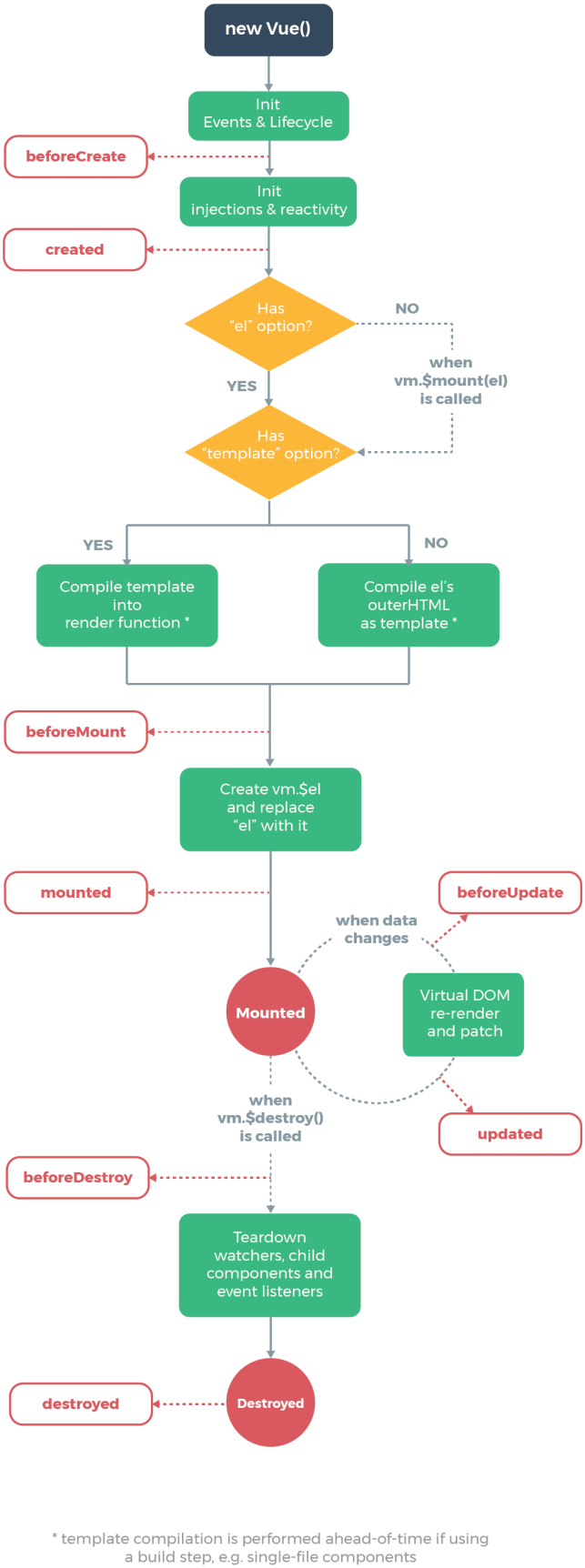


Figure 6 -Vue.js Lifecycle diagram (vuejs.org 2019).

There are 4 different ways of creating components in Vue.js.

- SFC (Single File Component)
- Component with a template
- Component with a render function
- Component with a render function and JSX syntax (Lilo, 2019)

Single file component is the most common way of creating a component in Vue.js. In Vue.js, every component accepts a few options including data, computed, watch, methods and lifecycle hooks.

2.5.1.3 Data in Vue.js components

Data in Vue.js components must be a function that returns an object holding the values of the data object for the component. This is to prevent the changes happening to the other data objects in other components. See Figure 7 that is taken from the official documentation in Vuejs.org (Vuejs.org 2020).

```JS
data: function () {
  return {
    count: 0
  }
}
```

Figure 7 -Data function in Vue.js (vuejs.org 2019)

2.5.1.4 Computed, watchers and methods in Vue.js

In Vue.js, JavaScript expressions can be used directly in the template as shown in Figure 8 however this can be problematic when wanting to reuse the same result. That is when the computed property comes into the game. All these expressions can be included in the computed property in a function and then reused as many times as needed. See Figure 9 and Figure 10 (Vuejs.org 2020).

```HTML
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

Figure 8 -Inline expressions in Vue.js components (vuejs.org 2020).

```HTML
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

Figure 9- Expressions in Vue.js components using computed (vuejs.org 2020).

```
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

Figure 10 -Computed property in Vue.js components (vuejs.org 2020).

The same result can be achieved using methods. In the above example, instead of writing computed, it could be written methods and include the reversedMessage function within the methods object. They operate exactly the same, the only difference is the caching. When using the computed option, if the message is not changed, the function won't be executed, and the previous result will be returned. While using methods object the function will be executed every time the reversedMessage function is called (Vuejs.org 2020).

Watchers are used to track changes in the data object. Although the official documentation suggests the use of computed property rather than custom watchers, watchers can be used when wishing to perform an asynchronous operation (Vuejs.org 2020).

3 Theoretical Framework

3.1 Previous studies

Although there have been prior studies relating to the comparison of various JavaScript framework/libraries as well as inquiries into developers choice of frameworks there is to the best of the author's knowledge no studies that solely focuses on the comparison of React.js, Vue.js and contingent technologies in regards to trend, architecture, and performance in development of single-page applications. Besides, considering that these technologies are constantly updated and improved, the exploration of the latest iterations of these stacks is crucial. To add reliability and validity to this study two similar studies are introduced.

"What affects the choice of a JavaScript framework" is a bachelor thesis (Duvander, Romhagen, 2019) written by Jacob Duvander & Oliver Romhagen for Jönköping University. In the thesis, the authors aim to find out the reason behind JavaScript framework preference among developers. The authors concluded that an active community, reputation, learning curve and documentation of frameworks are the key factors impact the choice of frameworks among developers. (Duvander, Romhagen, 2019)

"Supporting Web development decisions by comparing three major JavaScript Frameworks" is a bachelor thesis written by Eric Wohlgethan for faculty of engineering and Computer Science of Hamburg. Wohlgethan in his thesis explores some of the features and accessibility aspects of React.js, Vue.js and Angular.js, though he does not perform any performance tests. Wohlgethan's aim is to find out whether any of these three stands out in a way that can be used for most of the use cases (Wohlgethan, 2020).

Although these mentioned studies are addressing a range of overlapping topics there are gaps in these inquiries that should be addressed to provide a more complete comparative overview of the software stacks in terms of performance, most notably between React.js and Vue.js, which will be a primary topic of this study. Moreover, this paper will contribute an analysis of preference among industry practitioners in Sweden based on quantitative data to further understanding the trends and relevance between the various stacks.

4 Method and implementation

The chapter provides an overview of the work process of the study. Further, the approach and design of the study are described as well as the data collection and data analysis. The chapter ends with a discussion about the validity and reliability of the study.

4.1 Link between research questions and methods

The first research question involves the comparison of data sets, namely the run-time performance of four distinct operations of two software stacks. More specifically the comparison the question is addressing is the runtime performance of components in the following stacks respectively, React.js in MERN and Vue.js in MEVN. In order to acquire the necessary data an experiment was an appropriate choice of methodology. An experiment is a research method in which you manipulate one or more independent variables and measure their effect on one or more dependent variables (Bevans, 2019).

To provide an answer to the second question related to the trend and the preferred software stack among developers in Sweden, a survey was conducted. The aim was to gather quantitative and qualitative primary data from individuals who are currently active in the industry. A survey study allows information collected from a group of people by asking them questions and analyzing the result (McCombas, 2020), which in this case fits the purpose well. Conducting interviews could possibly be an alternative to the chosen methodology for the second research question, but since time was a limitation and such a process would be greatly time consuming conducting a survey was a more feasible choice.

4.2 Work process

For the experimental study, two simple “to-do” applications were created using MERN and MEVN respectively and their run-time performance was compared using Google Chrome development tool while executing CRUD- operations (Create tasks, Read tasks, Update tasks and Delete tasks). Both the stacks make use of the same database (MongoDB), server-side run time environment (Node.js) and the Node.js framework (Express) and their only difference are in the client-side framework/libraries. This approach is taken to minimize the risk of misevaluation. The experiment is iterated 20 times to ensure that the collected data is valid. All the tests can be replicated by anyone who has access to the source code, the same version of Google Chrome and a cloud based instanced as specified below. Tools that were used in the experiment are as follows:

- Google Chrome (Version 79.0.3945.130)
- Google Chrome’s built-in developer tools
- Linux/Unix, Ubuntu 18.4 LTS Bionic | 64 bit (installed on a virtual machine hosted by AmazonEC2)

The reason Google Chrome was chosen is that according to statcounter.com Chrome is among the most commonly used web browser worldwide holding 63% of browser market share. See Figure 11. The other reason for choosing Google Chrome is the built-in developer tools that among other useful features also allows the analysis of runtime performance.

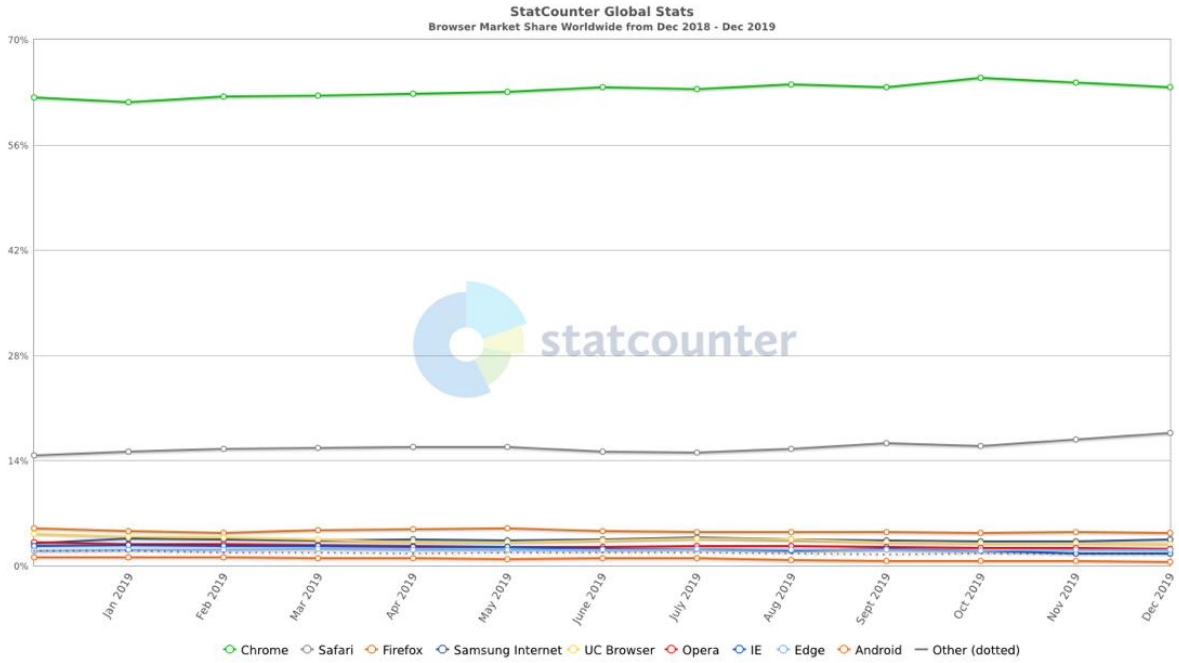


Figure 11 Browser market share worldwide (gs.statcounter.com 2019).

During the experiments, the CPU throttling[1] (6x slowdown) was used to be able to monitor the performance in a clearer way. Four tests were made using Google Chrome's built-in "Web developers' tools". These tests are as follows:

1. Loading the application
    - Recording starts from the reloading of the page and ends when all of the text and the graphical elements are loaded.
2. Adding a to-do task
    - Recording starts after writing the text in the text area and before pressing the enter key and stops when the task is added to the list and displayed.
3. Removing a to-do task
    - Recording starts from when the delete button is pressed and stops when the task is deleted, and the new list is fetched.
4. Updating a to-do task
    - Recording starts after the new text is written in the text area and before the update button is clicked. The recording stops after the task is updated and the new list is displayed.

Software and libraries used in building the to-do applications are as follows:

- Visual Studio Code (version 1.38.0)
- Mongoose (Object data modelling for MongoDB) (version 5.8.9)
- Express.js (version 4.17.1)
- Node.js (version 10.16.3)
- Vue.js (version 3.11.0)
- React.js (version 16.12.02)

1 Throttling is relative to your computer's capabilities. For example, the 2x slowdown option makes the CPU operate 2 times slower than its usual ability. DevTools can't truly simulate the CPUs of mobile devices, because the architecture of mobile devices is very different from that of desktops and laptops.

- NPM (version 6.9.0)

The to-do app is a basic application that allows adding, removing, deleting and displaying the list of added to-do tasks. The application is built based on the 3-tier model that includes a frontend application, a backend application and a database.

Two frontend applications are built with Vue.js and React.js and both contain similar design and functionalities. Both of the applications contain 4 components, each handling one of the CRUD operations and then they are brought together in the main file of the application (app.js and Vue.js) See Figure 12 and Figure 13 for the User Interfaces of each application. Also, the styling of the frontend applications was done using inline CSS.

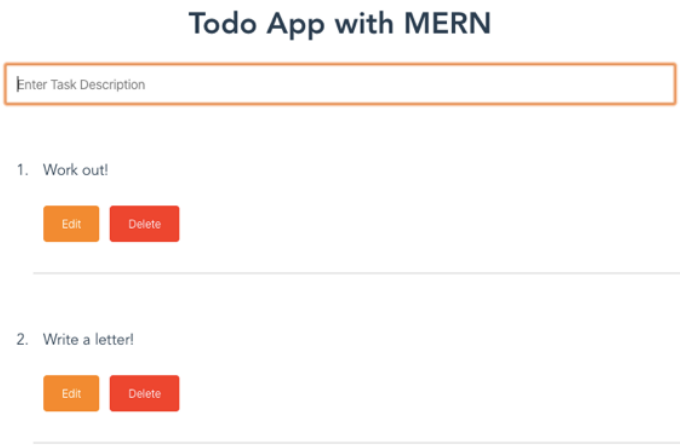


Figure 12 To-do app UI built with MERN stack.

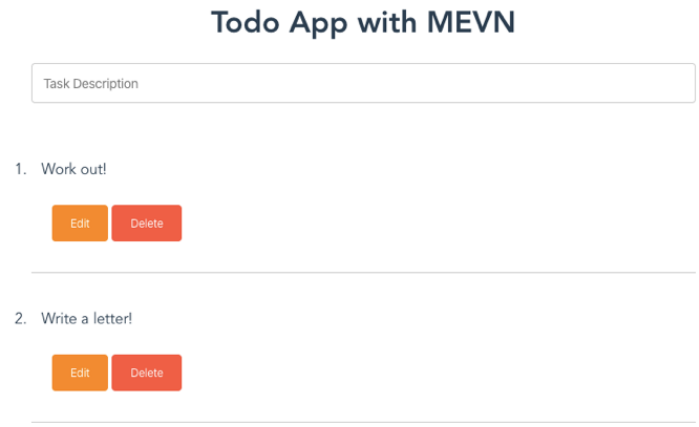


Figure 13 To-do app UI built with MEVN stack.

The backend application is built with Node.js and Express.js and it serves both of the frontend applications. To store the task data, Mongoose (Object data modeling for MongoDB) is used and the model contains only one property “taskDescription”. See Figure 14 for the database schema.

```
const mongoose = require("mongoose");

const TaskSchema = new mongoose.Schema({
  taskDescription: String
});

module.exports = mongoose.model("tasks", TaskSchema);
```

Figure 14 Mongoose database schema.

The survey was created using Google Forms and questions were limited to two, a multiple-choice question and one open-ended. On the first question, the participants were asked to choose the software stack they use when building a single page application and the choices included were MERN or MEVN, Both of the above and other. If they chose the "other" option, the they could write the software stack they use. The MERN and MEVN stack were included in the choices to measure the popularity of the stacks and the option "other" was included to gather data on their preferred software combination. On the second open-ended question, the participants were asked to elaborate on their choice of stack and share the factors that play a role in their decision-making process.

The survey was sent to approximately 70 companies located all over Sweden, the chosen companies were mostly tech companies who were assumed to work with web development. The companied mostly were directly contacted through the email address they provided in their official webpage and they were asked to forward the email to their development team, if they are not the right person to respond to the survey. LinkedIn and Facebook developers' group were also used to reach out to developers who are currently working at a Sweden based company.

4.3 Data Collection and analysis

The data gathered from the experiment is of quantitative nature and is collected using the Google Chromes developers' tool. As mentioned earlier, Chrome's developer's tool allows for run-time performance analysis of an application. The data collected on this experiment is based on the data that the chromes developers' tool provides. The data collected from all the iterations for each task are presented are presented in a table and the results are analyzed by averaging the iterations.

The data collected from the survey includes both quantitative and qualitative data. The quantitative data is used to illustrate the result using a pie-chart which shows the most preferred software stacks as well as the popularity of MERN and MEVN stacks in Sweden. The collected qualitative data is then reviewed and the common ideas and the pattern that come up repeatedly are used to draw conclusions.

4.4 Reliability and Validity

The data collected from the experiment was based on the data Google Chrome provided, though the applications were not running on a physical machine but rather on a virtual machine hosted by Amazon Elastic Compute Cloud (Amazon EC2). Amazon EC2 is a web service that allows users to create and run virtual machines in the cloud (aws.amazon.com, 2020).

This approach was chosen to prevent the invalidity of data. The alternative was to run the applications on a physical machine, in which case, it would have been difficult to conduct the experiment based on equal measures and there would have existed the possibility of other applications running on the background that could affect the performance of the applications and by extension the results. To prevent this a virtual machine was chosen to run the tests which in this case offers a more stable environment to conduct the experiment.

To ensure that the data gathered are from developers currently working at companies located in Sweden, their company email addresses were collected through the form, though to protect their anonymity, their first names are hidden. See Appendix 4 for the list of respondents.

5 Findings and Analysis

The chapter answers the research questions by processing the collected empirical data and presents formulated results of the study based on the research questions and purpose. The analysis is performed according to those principles stated in the methods chapter.

5.1 Research question 1

To answer the first research question, an experiment was conducted to measure the run-time performance of the to-do applications built with MERN and MEVN stacks. The tests in the experiment included measuring the loading time of the application, adding a task, updating a task and deleting a task. In this experiment, each task was iterated 20 times and the average of the 20 iterations are then compared. The data gathered from the tests are presented and analyzed below. St DEV refers to the standard deviation of the sample data, see Eq. (1). and ME in the tables below refers to the margin of error determined with 95% confidence (Z score =1.96) interval Eq. (2).

Formula used for determining standard deviation

$$s = \sqrt{\frac{\sum(\chi - \overline{\chi})^2}{N - 1}} \tag{1}$$

*Where **s** is the sample standard deviation, **N** the number sample data, χ is the mean value of the sample data*

Formula used for determining Margin of Error

$$ME = \mathrm{Z}\,\frac{s}{\sqrt{N}} \tag{2}$$

*Where **Z** is the z-score (Critical Value)*

The loading time of each application built with the MERN and MEVN stacks was recorded using the performance tool from Google Chromes' developers' tools. The measurement was taken from the beginning of evaluating the script and to the end of the composite layer. The start time was then deducted from the end time, to gather the duration of the run time. Table 1 displays the results gathered from the test for loading the to-do applications.

#	*MERN*	*MEVN*
1	90	107
2	70	91
3	64	85
4	77	96
5	99	97
6	66	86
7	54	88
8	65	127
9	90	89
10	56	103
11	63	100
12	71	86
13	65	89
14	88	107
15	87	89
16	70	90
17	68	98

18	52	107
19	62	94
20	52	100
Mean	**70**	**96**
St. DEV	**14**	**10**
ME	**6**	**4**

Table 1- Running time (ms) when loading the applications

As it is shown in Table 1, the average runtime performance of MERN (with 95% confidence interval) is between 64 and 76ms (70 ± 6ms) while MEVN is between 92 and 100ms (96 ± 4ms) based on the 20-sample data shown above. Because the calculated lower bound average of MEVN at (92) is higher than the average upper bound of MERN at (76) by 15ms it is safe to say that MERN outperformed MEVN at loading. However, from a user experience perspective, this difference might not even be perceivable.

The recording for adding a task started after writing the text on the inputs and before clicking enter. Therefore, the measurement started from when the submit event was registered and ended when the compositing layer was over. The duration was calculated by deducting the start time from the end time. Table 2 shows the results gathered from the test on adding a task.

#	*MERN*	*MEVN*
1	439	428
2	153	409
3	446	427
4	149	176
5	416	182
6	149	407
7	422	174
8	159	396
9	465	433
10	437	420
11	436	552
12	151	405
13	447	471
14	421	412
15	423	398
16	469	420
17	160	414
18	403	412
19	419	291
20	427	195
Mean	**350**	**371**
St. DEV	**133**	**107**
ME	**58**	**47**

Table 2- Running time (ms) when adding a task

Table 2 shows that MEVN performed the loading task in the average range of 324-418ms and MERN executed the same task with an average interval of 291-407ms. Because these average ranges overlap it is not possible to conclude that either of the stacks performed better then the other. However, the difference between the mean average of MERN (350) and MEVN (371) is 21ms and the difference in the margin of error for MERN (58) and MEVN (47) is 11ms. Because these differences are so small, it is not unreasonable to suspect that the respective stacks' runtime performance with regards to adding a task, is very similar.

The measurement for updating a task started from when the update button was clicked (event: clicked) and ended when the list was updated on the User Interface. Similar to the other tests, the duration was calculated by deducting the start time from the end time. Table 3 displays the results gathered from the tests for updating a task.

#	*MERN*	*MEVN*
1	497	407

2	441	414
3	288	398
4	440	433
5	424	373
6	409	408
7	427	447
8	331	526
9	411	412
10	442	394
11	476	419
12	444	428
13	409	412
14	426	390
15	432	422
16	433	393
17	406	412
18	417	409
19	413	431
20	428	405
Mean	**420**	**417**
St. DEV	**44**	**31**
ME	**19**	**13**

Table 3- Running time (ms) for updating a task.

The analysis of the data in Table 3 follows a similar pattern as in Table 2. MERN averaged on 420±19 and MEVN averaged on 417±13. Averaged upper and lower bounds overlap, so a definitive conclusion is not possible. Noted is that the calculated differences in mean averages (3ms) and margin of errors (6ms). This implies that the runtime performance between MERN and MEVN in terms of update-operations might also be similar.

Recording for deleting a task started before clicking the delete button and the measurement started from the click event and ended when the task was removed from the user interface. Table 4 displays the results gathered from the test.

#	*MERN*	*MEVN*
1	1044	441
2	418	432
3	524	431
4	424	433
5	431	447
6	424	421
7	404	465
8	412	427
9	410	658
10	420	433
11	430	398
12	430	424
13	434	416
14	434	405
15	414	441
16	418	386
17	460	428
18	407	493
19	422	407
20	506	386
Mean	**463**	**439**
St. DEV	**140**	**57**
ME	**61**	**25**

Table 4- running time (ms) when deleting a task

Data from Table 4 follow the same pattern as in Table 3 and Table 2, where MERN average value is 463±61 and MEVN average value is 439±25 and overlapping upper and lower average bounds prohibits any definite conclusions. Calculated differences in mean averages (24ms) and margin of errors (36ms) suggest similar runtime performance between MERN and MEVN with regards to delete-operations.

5.2 Research Question 2

To answer the second research question regarding the software stack preference among the Swedish based companies, a semi-structured survey was conducted, and companies located in different parts of the country including, Stockholm, Jönköping, Gothenburg and Malmö were contacted. Out of the approximately 70 companies that were contacted 12 of them responded. From the semi-structured survey, both qualitative and quantitative data were collected which in this section will be presented and analyzed.

The first question of the survey aimed to gather preferred software stacks among the participants. The participants were asked to either choose between MERN and MEVN or both and if they did not use any of them, they were asked to mention the software stack they use. The second survey question was asked to get some insight into why they prefer the ones they chose. The quantitative and qualitative data gathered from the survey are discussed below. Figure 15 displays the preferred software stacks among the respondents of the survey.

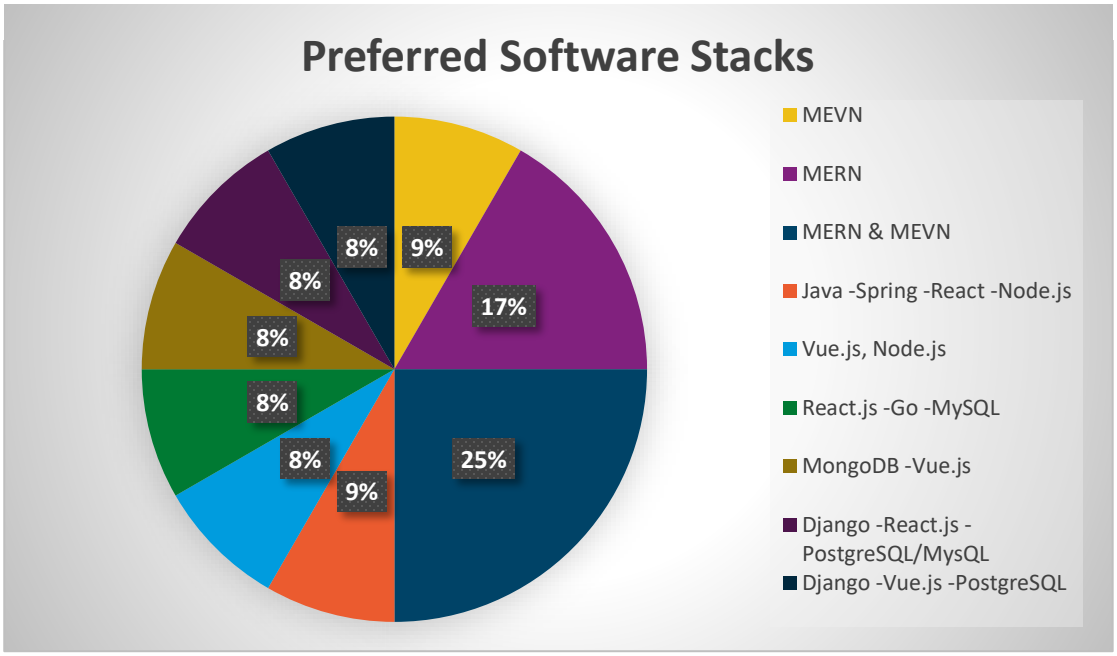


Figure 15- Preferred software stacks among the respondents.

As shown in Figure 15, the responses gathered from the surveys are quite varied. Because the number of respondents are low making generalizations becomes challenging and using it as a basis to confidently determine the popularity of the stacks even harder. That said, according to this dataset, the number of companies that use both MERN and MEVN is the highest at 25%. The MERN stack is the second most preferred stack among the respondents. The respondents who chose these stacks mentioned that developers enjoy working with JavaScript and using only one language both for the frontend and the backend adds to the strength of the team.

Furthermore, it is also possible to see a pattern regarding the client-side software in the prefered stacks. All the respondents have mentioned that they use either React.js or Vue.js or both as their preferred client side software as shown in Figure 16.

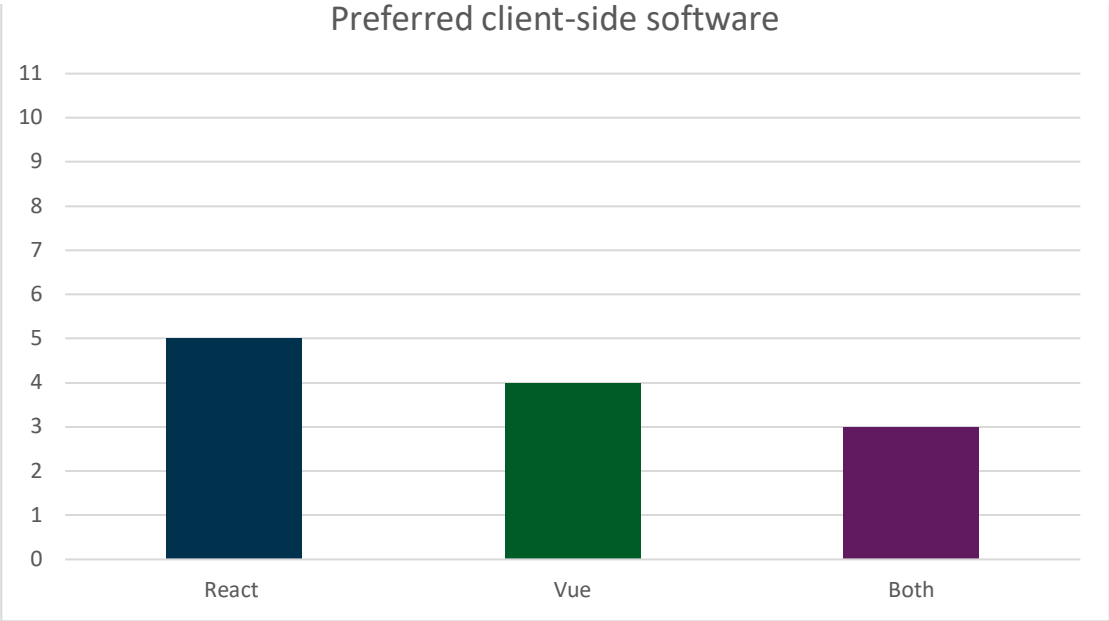


Figure 16- Preferred client-side software among the respondents.

The respondents that preferred Vue.js over other frameworks mentioned that it has a less steep learning curve and therefore it is easier for new employees to learn. Another reason the respondents gave for preferring Vue.js was that it was not affiliated with Facebook yet they did use React.js when it was demanded. The respondents that preferred React.js as their client-side software stated its superior modularity as reason and that it allowed the use of styled-components which made React.js easy to maintain. Another stated preference was the current ease of which they could find good React.js developers. Lastly, another respondent stated it was more "powerful" then Vue.js.

React.js and Vue.js were not always combined with Node.js or MongoDB. In Figure 17, stated by the respondents are preferences of other server-side software in combination with React.js or Vue.js which include Java (a programming language), Go (a programming language) and Django (a Python framework). Some of the reasons they preferred the other mentioned software were the maturity of the ecosystems like Python and Django, the client's need and already existing knowledge of the developers. One of the respondents also stated that the choice of software combinations are not permanent but depend on the project and the resources available.

Aside from combining Vue.js or React.js with MongoDB, the respondents mentioned that they prefer to use a relational database such as PostgreSQL or MySQL in combination to React.js or Vue.js. Because Relational database guarantees providing ACID (atomicity, consistency, isolation and durability) which is a set of properties in database transactions.

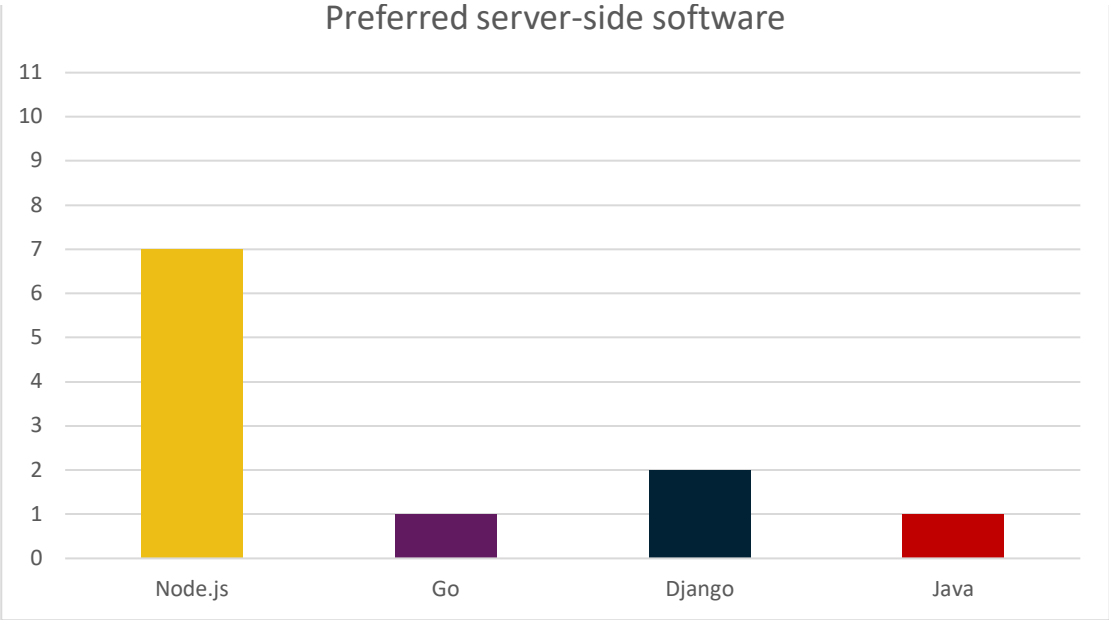


Figure 17- Preferred server-side software among the respondents.

6 Discussion and conclusion

This chapter summarizes the results of the study. Further, it describes the implications and limitations and also describes the conclusions and recommendations of the study. Suggestions for further research are given at the end of the chapter.

6.1 Discussion of findings

The purpose of conducting the experiment was to find out how fast MERN and MEVN perform when executing CRUD operations of a single page application in comparison to one another. Experimentation successfully allowed the collection of primary and empirical data, in this case the speed of which the respective stack executed the four CRUD operations. For the experiment, two simple "to-do" applications were built with both MERN and MEVN. Both of the applications had the same functionality and structure. The backend application which was built with Node.js and Express framework served both of the front-end applications which were built in accordance with Vue.js and React.js documentation.

Each corresponding CRUD operation; creating, reading (loading), updating and deleting a task was executed and the runtime was measured with the two "to-do"-applications. 20 iterations were performed. The data collected were then averaged to be able to compare the run-time performance of each task tested between the MERN and MEVN stack. However, it must be noted that, due to the limited timeframe, the comparison was based on the DOM manipulation of text-only content and that no image rendering was included in the tests. Therefore the mentioned results is a performance analysis of the stacks in regards to text-only content rendering.

The results gathered from the tests which are displayed in section 5.1 ,shows that MERN stack performed faster than the MEVN stack in loading with a very minimal difference. When comparing the results gathered from the adding, updating and deleting, it was noted that both the MERN and MEVN stacks perform very closely to one another. However, reporting exact differences was not possible but since the average values and the margin of error of both the stacks were very close, it was safe to say that the performances of both the stacks were very similar when adding, deleting and updating a task.

The applications were installed on a Cloud-based Virtual Machine to prevent the collection of invalid data. If the applications were running on a physical machine there could be other applications running in the background that could affect the performance of the to-do applications and proving that the measurement was done on equal grounds would not have been possible. That being mentioned, there were still fluctuations in the data collected from each test. This could be a result of network latency that according to Popescu, et al. is inevitable among the applications deployed on cloud-based computing instances. When choosing a computing instance from the providers such as Google Cloud Platform, Amazon EC2 or Microsoft Azure, it is possible to choose the specifications of the used instances, but it is not fully possible to guarantee that network latency will not affect the performance of the applications (Popescu, et al, 2017) which in this case could explain the fluctuations reported in the test results. That being said, with 20 iterations it was possible to reach a relatively low standard deviation in the sample data and measure the run-time performances of both of the applications.

The purpose with conducting the semi-structured survey was to identify the preferred software stack among the swedish based companies for better understanding of trends and the relevance between the various stacks in Sweden. To meet the purpose, a semi structured survey was conducted and the survey was sent to approximately 65-70 companies in Sweden that were assumed to be working with web development related services. It's not possible to give an exact number on how many were contacted, as some of the companies were asked to forward it through their relevant channel. Out of all the companies contacted, only 12 people responded, therefore generalising the result became quite hard, as there could be other preferences and opinions that are not listed in the results.

Although the number of respondents were low, the results collected relieved that even though Vue.js and React.js are quite commonly used for building a single page application they are not always combined with MongoDB , Express and Node.js. Depending on the requirement, the community behind the software and availability of the specific software developer companies do change their software combination and they are not strictly sticking to one software stack.

Moreover, some of respondents of the surveys mentioned that availability of the developer or the pre-existing knowledge of the developer was the reason they preferred that software stack, this indicates that as a developer , one gets to have quite a flexibility in choosing a software depending on the knowledge and the preference they have.

6.2 Limitations

Due to time restrictions the experiment was limited to measure the runtime performance of rendering text only, not images. Including image-rendering too would better reflect how the respective software actually performs. Also a larger dataset of measurements would increase the accuracy of the results.

Not getting enough survey responses from the companies was a major limitation as it severely limited the ability to draw more general conclusions. Further, not collecting specific data such as the respondent's position within the company also limited analysis.

6.3 Conclusions

In comparison MERN and MEVN both execute CRUD operations in single page applications at very similar speeds. This implies that runtime performance is not a deciding factor in the choice between MERN or MEVN, and more specifically between React.js and Vue.js.

According to the results reported in section 5.2, Vue.js and React.js are commonly preferred by all the responding Swedish based companies, though they are not always preferred in combination with Node.js or MongoDB. One of the reasons for their preference of Vue.js was the ease of learning of the framework, a possible reference to how Vue.js' components are structured. As Vue.js keeps the markup, logic and style separate and allows the developer to write pure HTML in the template section of the component and pure CSS in the styling section. This approach decreases the steepness of the learning curve for Vue.js which gives it an advantage over React.js. The learning curve is an important aspect in a developers choice of framework as Duvander and Romhagen also reported in their thesis when investigating the factors that influence developers' choice of framework. In contrast to Vue.js, React.js uses JSX and keeps the markup and the logic together allowing to use JavaScript features fully when building the view of the page. This presents an additional complexity that increases the steepness of the learning curve for the developers but makes it "powerful" as one of the respondents stated. React.js' strong focus on JavaScript gives a lot of power to the developer to manipulate the data and the view, a point also concluded by Wohlgethan referred to in section 3.1.

Further, the community behind a framework is another factor for developers in choosing a framework or a library. This was reflected in a response from the survey where Vue.js was preferred because it is not maintained by Facebook. The importance of the community was also a conclusion also made by Duvander et. all in "what affects developers choice of framework" referred to in section 3.1.

Ultimately it is not unreasonable to assume that there are not a given set of reasons that determine the choice of software combinations in the development of web applications, but a multitude. Besides the ones elaborated upon above, the factors that determine the company's choice of software stacks range from what resources are available, the skill sets of the developers, the clients' needs and even the individual biased preference of any developer involved in the decision-making process.

6.4 Further research

To give the study more legitimacy and validity a similar study can be performed where the experiment regarding the run time performance of MERN and MEVN includes image rendering as well as text rendering, to see whether the results would still be similar to the results gathered in this study.

In addition, a broader survey study could also be conducted where it is not only limited to Sweden and only to the developers in the companies but rather includes both freelance developers as well as companies located all over Scandinavia to get a broader overview on the software preferences.

7 References

Anastasya Kryzhanovska 2019, Pros and Cons of building single page application in 2019
https://gearheart.io/blog/pros-and-cons-building-single-page-applications-2019/
(Acc. December 14, 2019)

Ashleynolan.co.uk 2019, The Front-End Tooling Survey 2019 – Results
https://ashleynolan.co.uk/blog/frontend-tooling-survey-2019-results
(Acc. December 27, 2019)

aws.amazon.com, 2020, Launch a Linux Virtual Machine,
https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/
(Acc. February 18, 2020)

Clusterdb.com 2019, The modern application stack part 5, figure
http://www.clusterdb.com/tag/react
(Acc. December 14, 2019)

Edureka.com, 2019, in React everything is a component, figure 3
https://www.edureka.co/blog/react-components/
(Acc. December 25, 2019)

Eric Wohlgethan 2018, Supporting Web Development Decisions By Comparing Three Major JavaScript Frameworks: Angular , React and vue.js
http://edoc.sub.uni-hamburg.de/haw/volltexte/2018/4350/pdf/BA_Wohlgethan_2176410.pdf
(Acc. March 21, 2020)

Expressjs.com (2019) Using template engines with Express
https://expressjs.com/en/guide/using-template-engines.html
(Acc. December 22, 2019)

gs.statcounter.com 2019, Browser Market Share Worldwide,
https://gs.statcounter.com/
(Acc. January 28, 2020)

Ian Allen 2018, The brutal lifecycle of JavaScript frameworks,
https://stackoverflow.blog/2018/01/11/brutal-lifecycle-javascript-frameworks/
(Acc. January 25, 2020)

Jacob Duvander and Oliver Romhagen 2019, What affects the choice of a JavaScript framework
http://www.diva-portal.org/smash/get/diva2:1352822/FULLTEXT01.pdf
(Acc. January 22, 2020)

John Kaga, 2018, Understanding React Components
https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb
(Acc. December 25, 2019)

Lilo 2019, 4 different ways to create Vue components
https://dev.to/lilotop/4-different-ways-to-create-vue-components-3nma
(Acc. January 4, 2020)

Margaret Rouse (2019), Software Stack
https://searchapparchitecture.techtarget.com/definition/software-stack
(Acc. December 14, 2019)

Margaret Rouse, 2020, searchservervirtualization.techtarget.com , Virtual Machine,
https://searchservervirtualization.techtarget.com/definition/virtual-machine
(Acc. February 18, 2020)

Mobilunity.com 2019, Vue JS Framework: An Overview of Vue Application
https://mobilunity.com/blog/vue-js-developer-for-hire/
(Acc. January 4, 2020)

Mongodb.com (2019), BSON Types
https://docs.mongodb.com/manual/reference/bson-types/
(Acc. December 18, 2019)

P. Patel, 2018, freecodecamp.org, What exactly is Node.js?
https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/
(Acc. December 22, 2019)

Popescu, D., Zilberman, N., & Moore, A. (2017). Characterizing the impact of network latency on cloud-based applications' performance,
https://www.repository.cam.ac.uk/bitstream/handle/1810/270654/UCAM-CL-TR-914.pdf?sequence=1&isAllowed=y
(Acc. February 22, 2020)

Reactjs.org, 2019, Introducing Hooks
https://reactjs.org/docs/state-and-lifecycle.html
(Acc. December 25, 2019)

Rebbeca Bevans 2020, Scribber.com, A guide to experimental design
https://www.scribbr.com/methodology/experimental-design/
(Acc. February 7, 2020)

Shannon Bradshaw, Eoin Brazil, Kristina Chodorov, (2019) *MongoDB: The definitive guide*, United States Of America, O'Reilly Media.inc

Shona McCombes 2020, Scribber.com, How to do survey research
https://www.scribbr.com/methodology/survey-research/
(Acc. February 7, 2020)

Simplelearn.com 2019, MongoDB replication and sharding tutorial
https://www.simplilearn.com/replication-and-sharding-mongodb-tutorial-video
(Acc. December 25, 2019)

Stackoverflow.com 2019, Developer Survey Results,
https://insights.stackoverflow.com/survey/2019
(Acc, December 27, 2019)

Vasan Subramanian, (2019) *Pro MERN Stack* Bangalore, Karnataka, India, ISBN-13 (electronic): 978-1-4842-4391-6

Vuejs.org 2019, Components basics,
https://vuejs.org/v2/guide/components.html
(Acc, January 4, 2020)

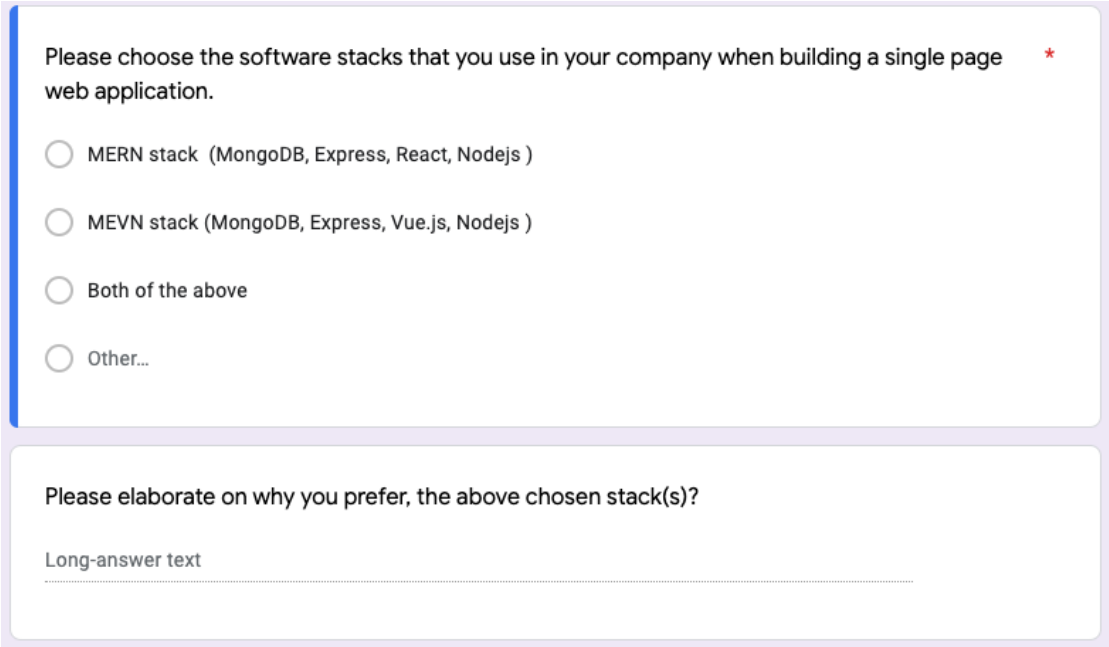
Vuejs.org 2019, Computed properties and watchers
https://vuejs.org/v2/guide/computed.html
(Acc, January 4, 2020)

Vuejs.org 2019, Lifecycle diagram
https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram
(Acc, January 4, 2020)

8 Appendices

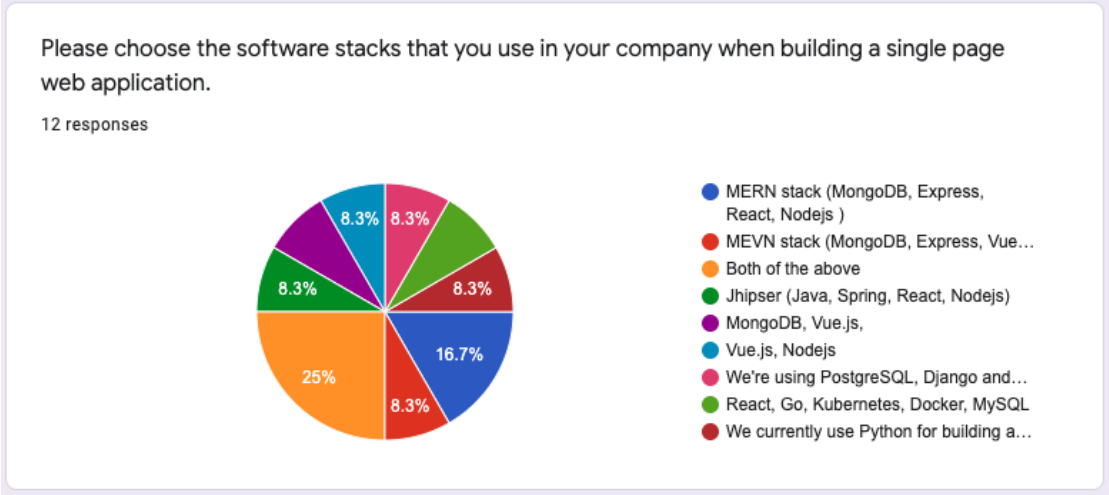
8.1 Appendix 1

Screen shot from the survey questions



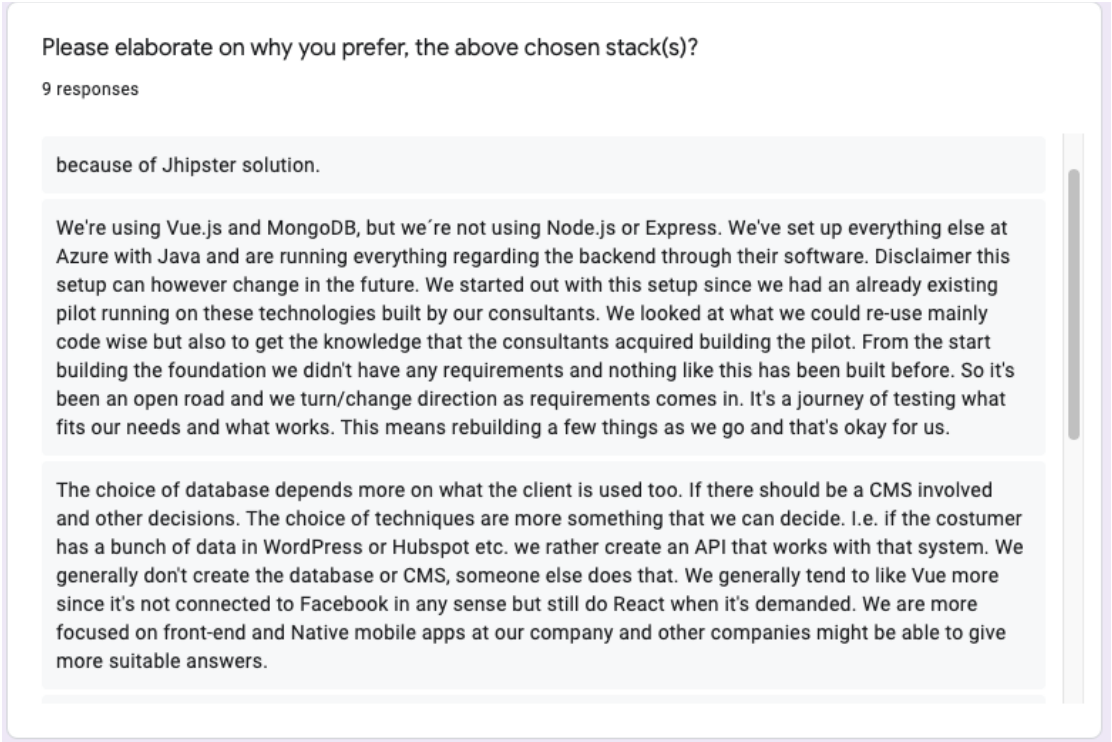
8.2 Appendix 2

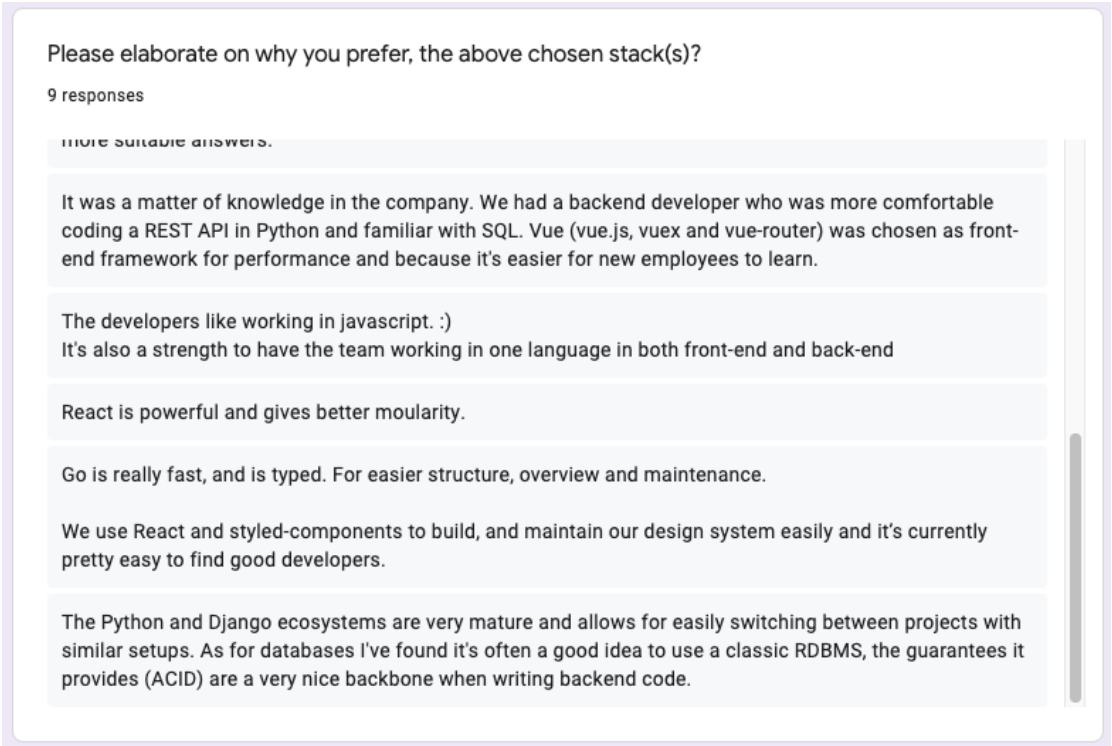
Screenshot from the quantitative data gathered from the survey using Google form



8.3 Appendix 3

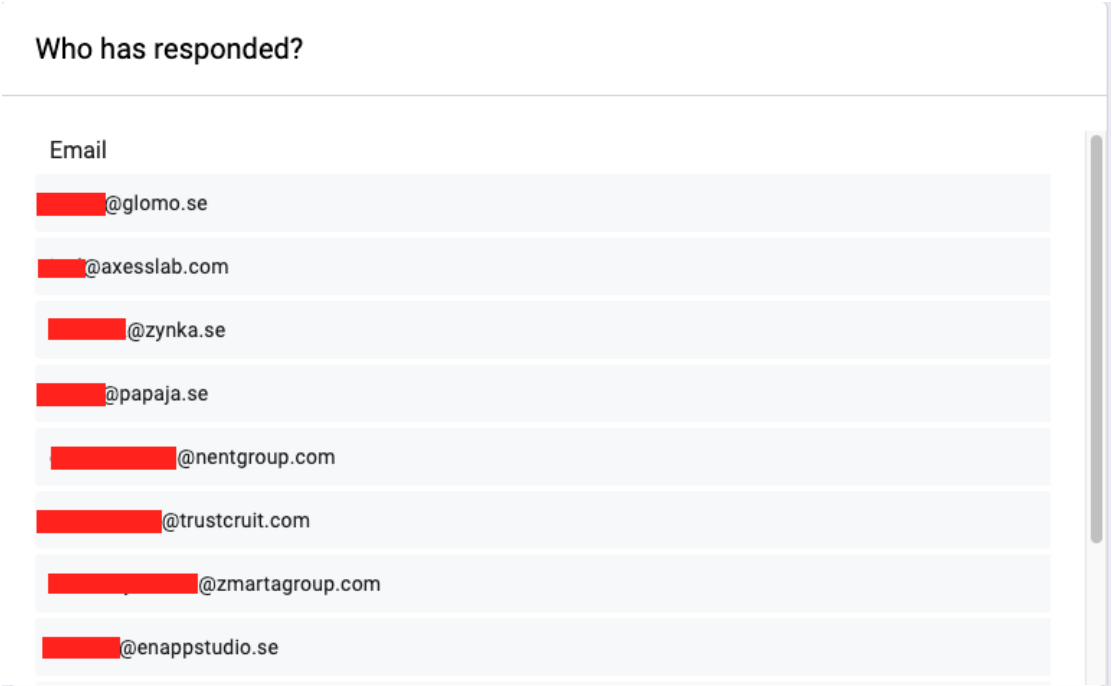
Screenshots from the qualitative data gathered from the survey using Google form

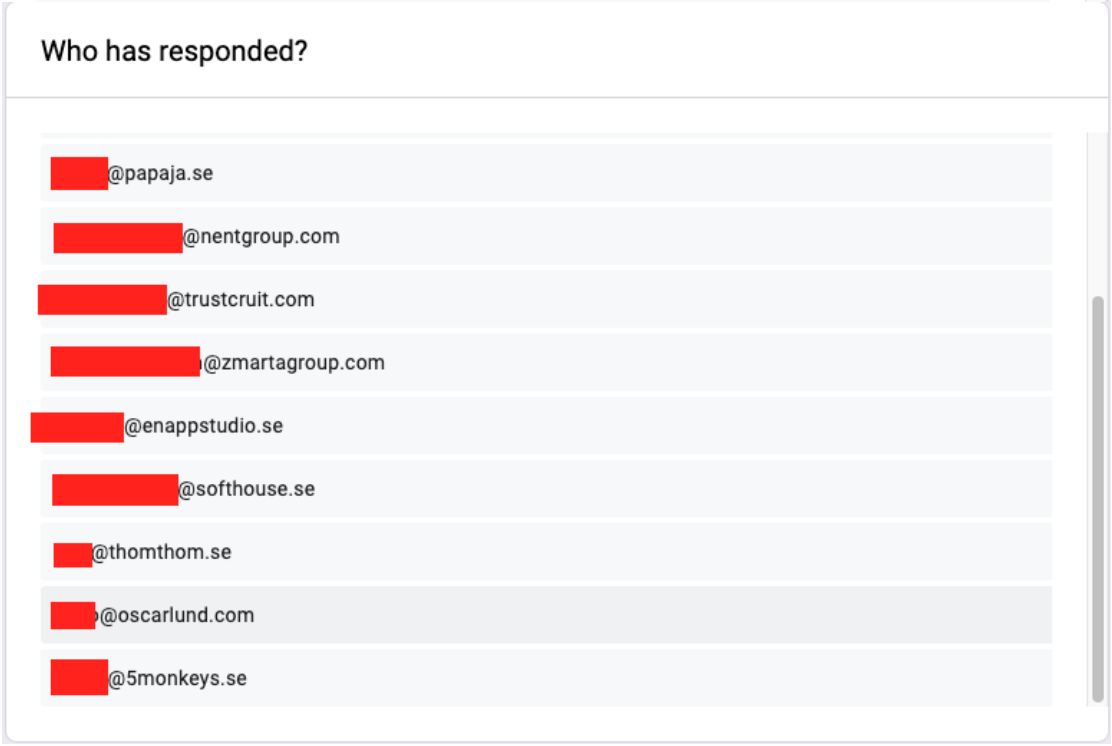




8.4 Appendix 4

List of developers responded to the survey. The name of respondents is hidden to protect their privacy. This list is added to share the name of the companies only.





8.5 Appendix 5

Screenshots from frontend application built with React

App.js

todo-app > src > App.js > ...

```
import React, { Component } from "react";
import AddTask from "./components/AddTask";
import ShowTasks from "./components/ShowTasks";

const divStyle = {
  fontFamily: "Avenir",
  textAlign: "center",
  color: "#2c3e50",
  margintop: "60px"
};
class App extends Component {
  state = {
    bool: false,
    task: ""
  };

  taskAdded = (task, bool) => {
    console.log(task, bool);
    this.setState({
      bool: bool,
      task: task
    });
  };
  render() {
    return (
      <div style={divStyle}>
        <h1>Todo App with MERN</h1>
        <AddTask taskAdded={this.taskAdded} />
        <ShowTasks
          isTaskAdded={this.state.bool}
          valueTask={this.state.task}
          taskAdded={this.taskAdded}
        />
      </div>
    );
  }
}

export default App;

```

AddTask.js

todo-app > src > components > AddTask.js > AddTask > render

```
import React, { Component } from "react";
const inputStyle = {
  width: "50%",
  padding: "12px",
  border: "1px solid #ccc",
  borderRadius: "4px",
  resize: "vertical",
  fontSize: "14px"
};
export default class AddTask extends Component {
  constructor(props) {
    super(props);
    this.state = {
      taskDescription: "",
      taskAdded: false
    };
  }
  onSubmitHandler = e => {
    e.preventDefault();
    const task = {
      taskDescription: this.state.taskDescription
    };
    fetch("http://localhost:5000", {
      method: "post",
      body: JSON.stringify(task),
      headers: {
        "Content-Type": "application/json"
      }
    })
      .then(response => response.json())
      .then(body => {
        this.setState({
          taskAdded: true,
          taskDescription: ""
        });
        this.props.taskAdded(body, this.state.taskAdded);
      });
  };
  render() {
    return (
      <div>
        <form onSubmit={this.onSubmitHandler}>
          <input
            type="text"
            onChange={e => {
              this.setState({
                taskDescription: e.target.value
              });
            }}
            value={this.state.taskDescription}
            name="taskDescription"
            style={inputStyle}
          />
        </form>
      </div>
    );
  }
}

```

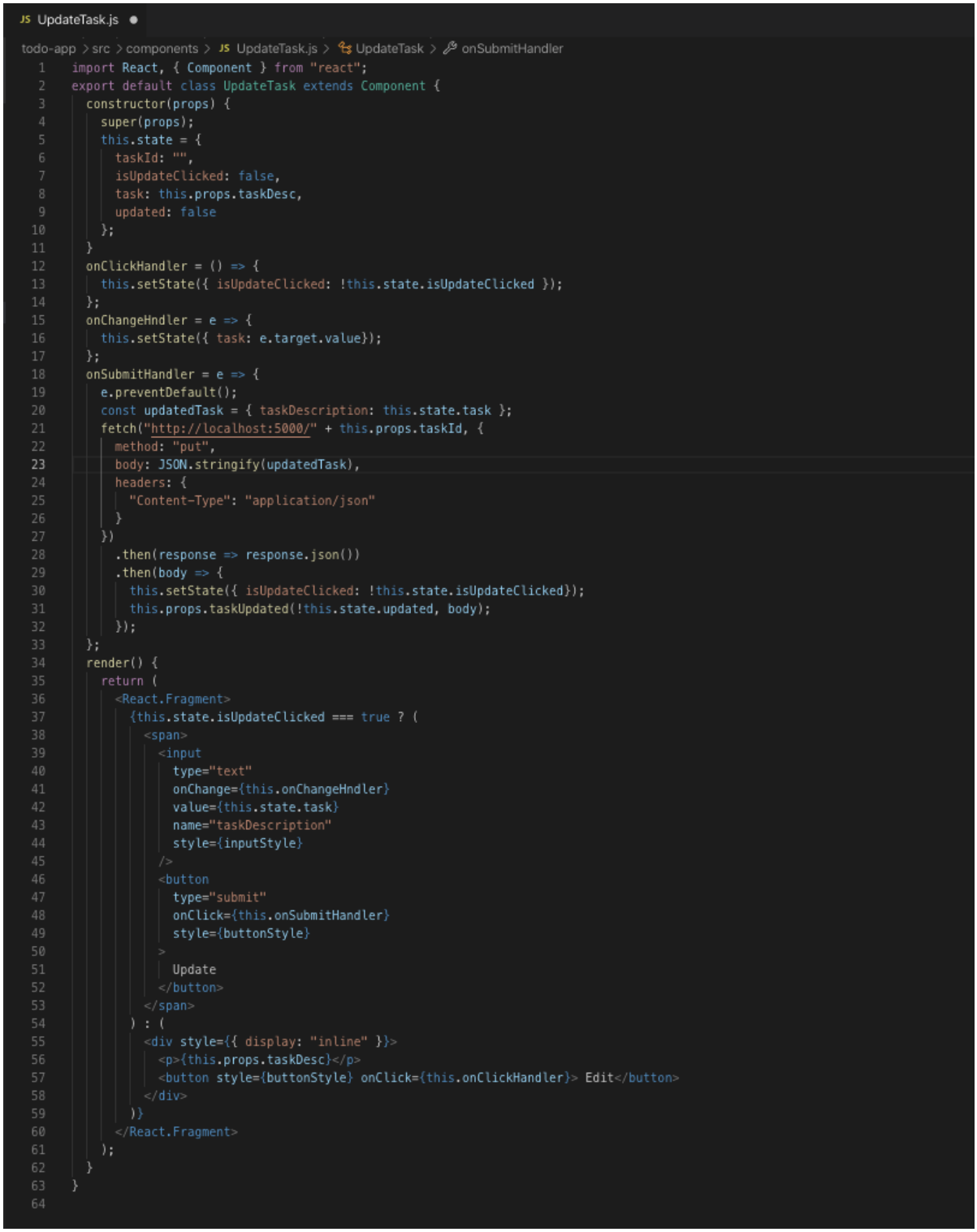
DeleteTask.js

todo-app > src > components > DeleteTask.js > DeleteTask > onClickHandler

```
import React, { Component } from "react";

const buttonStyle = {
  margin: "11px",
  backgroundColor: "red",
  color: "white",
  border: "none",
  padding: "12px 20px",
  borderRadius: "4px",
  fontSize: "12px",
  cursor: "pointer",
  marginright: "4px",
  display: "inline",
  display: "inline"
};

export default class DeleteTask extends Component {
  constructor(props) {
    super(props);
    this.state = {
      taskDeleted: false
    };
  }

  onClickHandler = () => {
    fetch("http://localhost:5000/" + this.props.taskId, {
      method: "delete",
      headers: {
        "Content-Type": "application/json"
      }
    }).then(response => {
      this.props.taskDelete(this.props.taskId);
    });
  };

  render() {
    return (
      <button style={buttonStyle} onClick={this.onClickHandler}>
        Delete
      </button>
    );
  }
}

```


ShowTasks.js
todo-app > src > components > ShowTasks.js > ShowTasks > componentDidUpdate
import React, { Component } from "react";
import UpdateTask from "./UpdateTask";
import DeleteTask from "./DeleteTask";
export default class ShowTasks extends Component {
constructor(props) {
super(props);
this.state = {
tasks: [],
taskUpdated: false,
taskDeleted: false,
taskId: "",
updatedTask: ""
};
}
componentDidMount() {
fetch("http://localhost:5000", {
method: "get",
headers: {
"Content-Type": "application/json",
Accept: "application/json"
}
})
.then(response => response.json())
.then(body => this.setState({ tasks: body }));
}
componentDidUpdate(prevProps, prevState) {
if (this.props.isTaskAdded !== prevProps.TaskAdded) {
if (this.props.isTaskAdded != "") {
this.setState({ tasks: [...this.state.tasks, this.props.valueTask] });
this.props.taskAdded(false);
}
}
if (this.state.taskUpdated !== prevState.taskUpdated) {
const index = this.state.tasks.findIndex(
x => x._id === this.state.updatedTask._id
);
if (index !== -1) {
const updatedTasks = this.state.tasks.slice();
updatedTasks[index] = this.state.updatedTask;
this.setState({ tasks: updatedTasks, taskUpdated: false });
}
}
if (this.state.taskDeleted !== prevState.taskDeleted) {
this.setState({
tasks: this.state.tasks.filter(task => task._id !== this.state.taskId)
});
}
}
isTaskAdded = value => {
this.setState({
taskAdded: value
});
};
taskUpdated = (bool, task) => {
this.setState({
taskUpdated: bool,
updatedTask: task
});
};
taskDeleted = id => {
this.setState({
taskDeleted: !this.state.taskDeleted,
taskId: id
});
};

```
  render() {
    return (
      <div style={{ textAlign: "left" }}>
        <ol style={{ margin: "22 auto" }}>
          {this.state.tasks.map((item, index) => (
            <li
              key={index}
              style={{
                borderBottom: "1px solid #ccc",
                width: "50%",
                margin: "22px auto",
                padding: "22px 11px"
              }}
            >
              <UpdateTask
                taskId={item._id}
                taskDesc={item.taskDescription}
                taskUpdated={this.taskUpdated}
              />
              <DeleteTask taskId={item._id} taskDelete={this.taskDeleted} />
            </li>
          ))}
        </ol>
      </div>
    );
  }
}

```



Appendix 6 – Screenshots from the frontend application built with Vue.js

App.vue

client > todo-mevn > src > App.vue > {} "App.vue" > script > methods > taskAdded

```vue
<template>
  <div id="app">
    <h1>Todo App with MEVN</h1>
    <AddTask @taskAdded="taskAdded" />
    <ShowTasks :isTaskAdded="isTaskAdded" />
  </div>
</template>

<script>
import AddTask from "./components/AddTask.vue";
import ShowTasks from "./components/ShowTasks.vue";
export default {
  name: "home",
  components: { AddTask, ShowTasks },
  data() {
    return {
      isTaskAdded: ""
    };
  },
  methods: {
    taskAdded(value) {
      this.isTaskAdded = value;
    }
  }
};
</script>


<style>
* {
  box-sizing: border-box;
}

#app {
  font-family: "Avenir", Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

```

AddTask.vue

client > todo-mevn > src > components > AddTask.vue > {} "AddTask.vue" > style > input[type="text"]

```vue
<template>
  <div class="home">
    <form @submit.prevent="addTask">
      <input type="text" class="form-input" v-model="description" placeholder="Task Description" />
    </form>
  </div>
</template>
<script>
export default {
  name: "home",
  data() {
    return {
      description: ""
    };
  },
  watch: {
    description(newVal, oldVal) {
      this.$emit("taskAdded", false);
    }
  },
  methods: {
    addTask() {
      const newTask = { taskDescription: this.description };
      fetch("http://localhost:5000", {
        method: "post",
        body: JSON.stringify(newTask),
        headers: {
          "Content-Type": "application/json"
        }
      })
        .then(response => response.json())
        .then(body => this.$emit("taskAdded", body));
      this.description = "";
    }
  }
};
</script>
<style >
input[type="text"] {
  width: 50%;
  padding: 12px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
button[type="submit"] {
  background-color: #4caf50;
  color: white;
  padding: 12px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
button[type="submit"]:hover {
  background-color: #45a049;
}
</style>
```

DeleteTask.vue

client > todo-mevn > src > components > DeleteTask.vue > {} "DeleteTask.vue" > script

```vue
<template>
  <button @click="deleteTask">Delete</button>
</template>

<script>
export default {
  name: "DeleteTask",
  props: ["taskId"],
  data() {
    return {
      id: this.taskId
    };
  },
  methods: {
    deleteTask() {
      fetch("http://localhost:5000/" + this.id, {
        method: "delete",
        headers: {
          "Content-Type": "application/json"
        }
      }).then(response => this.$emit("taskDelete", this.taskId));
    }
  }
};
</script>

<style scoped>
button {
  margin-top: 11px;
  background-color: #ff6347;
  color: white;
  font-size: 12px;
  border: none;
  padding: 12px 20px;
  border-radius: 4px;
  cursor: pointer;
}
button:hover {
  background-color: #ff4500;
}
</style>
```

ShowTasks.vue

client > todo-mevn > src > components > ShowTasks.vue > {} "ShowTasks.vue" > script > watch

```vue
<template>
  <ol class="taskLists">
    <li v-for="task in tasks" :key="task.index" class="list">
      <UpdateTask :taskId="task._id" @taskUpdate="taskUpdate" :task="task.taskDescription" />
      <DeleteTask :taskId="task._id" @taskDelete="taskDelete" />
    </li>
  </ol>
</template>
<script>
import DeleteTask from "./DeleteTask";
import UpdateTask from "./UpdateTask";
export default {
  name: "ShowTasks",
  props: ["isTaskAdded"],
  components: { DeleteTask, UpdateTask },
  data() {
    return {
      tasks: [],
      isTaskEdited: false,
      isTaskDeleted: false,
      updatedTaskData: "",
      deletedTaskId: ""
    };
  },
  watch: {
    isTaskAdded(newValue, oldValue) {
      if (newValue) {
        this.tasks = [...this.tasks, this.isTaskAdded];
      }
    },
    isTaskEdited(newVal, oldVal) {
      if (newVal) {
        const index = this.tasks.findIndex(
          x => x._id === this.updatedTaskData._id
        );
        if (index !== -1) {
          const updatedTasks = this.tasks.slice();
          updatedTasks[index] = this.updatedTaskData;
          this.tasks = updatedTasks;
        }
      }
    },
    isTaskDeleted(newVal, oldVal) {
      this.tasks = this.tasks.filter(task => task._id !== this.deletedTaskId);
    }
  },
```


ShowTasks.vue
client > todo-mevn > src > components > ShowTasks.vue > {} "ShowTasks.vue" > script > watch
},
created() {
fetch("http://localhost:5000", {
method: "get",
headers: {
"Content-Type": "application/json",
Accept: "application/json"
}
})
.then(response => response.json())
.then(body => (this.tasks = body));
},
methods: {
taskUpdate(val) {
this.updatedTaskData = val;
if (val !== "") {
this.isTaskEdited = true;
} else {
this.isTaskEdited = false;
}
},
taskDelete(val) {
this.deletedTaskId = val;
this.isTaskDeleted = !this.isTaskDeleted;
}
}
};
</script>
<style scoped>
ol {
width: 50%;
margin: 22px auto;
text-align: left;
padding: 0 !important;
}
li {
padding: 22px 11px;
width: 100%;
border-bottom: 1px solid #ccc;
}
</style>

Appendix 7 – Screenshots from the backend application created with Node.js and Express

index.js

server > index.js > ...

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const app = express();
const config = require("./config/index.json");

// connect to the database and load models
require("./models").connect(config.dbUri);

//json bodyparser
app.use(bodyParser.json());

//enable cors
app.use(cors({ exposedHeaders: ["Location"] }));

const tasks = require("mongoose").model("tasks");

//create tasks
app.post("/", function(request, response) {
  const newTask = request.body;
  const newTaskData = new tasks(newTask);
  newTaskData.save(err => {
    if (err) {
      response.status(500).end();
    } else {
      response.status(201).end();
    }
  });
});

//get tasks
app.get("/", function(request, response) {
  tasks.find({}, (err, task) => {
    if (task) {
      response.status(200).json(task);
    } else if (err) {
      response.status(500).json(err);
    }
  });
});

```


index.js
server > index.js > ...
40 });
41
42 //update tasks
43 app.put("/:id", function(request, response) {
44 const taskId = request.params.id;
45 tasks.findOneAndUpdate({ _id: taskId }, request.body, err => {
46 if (err) {
47 return response.status(400).json({ errors: "task id not found." });
48 } else {
49 return response.status(204).end();
50 }
51 });
52 });
53
54 //delete tasks
55 app.delete("/:id", function(request, response) {
56 const taskId = request.params.id;
57 tasks.findByIdAndDelete({ _id: taskId }, err => {
58 if (err) {
59 return response.status(400).json({ errors: "task id not found." });
60 } else {
61 response.status(204).end();
62 }
63 });
64 });
65
66 const port = 5000;
67
68 app.listen(port, () => console.log(`server is running on port ${port}`));
69