



Expertise
and insight
for the future

Tuan Vu

Building a Food Application with Full Stack JavaScript

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

28 April 2020

Author Title	Tuan Vu Building a food application with full stack JavaScript
Number of Pages Date	35 pages 28 April 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of the degree program
<p>The main goal of the final year project was to develop a full-stack food application based on JavaScript. The idea of using only one programming language on both the client and the server has many advantages. It helps software developers to reuse components and resources.</p> <p>The application was built using React in the front-end development and Firebase, Node.js and Express in the back-end development. Node.js is the most popular JavaScript runtime environment at the moment and has no competitors that have as much as 10% of Node.js's community. Firebase was used as a database for the application and Express was used to build the server. On the front-end, React.js was used along with Redux for state management.</p> <p>After three months of the first development phase, the application could provide basic features such as choosing a restaurant and showing directions. The application was tested by a small number of real users and their feedback was positive. Based on the feedback collected, the application could move on to the next stage to get more features.</p> <p>The project can be perceived as a success because through this project, the web development team achieved all the initial objectives, which included developing a working application for food lovers and trying to meet specific technical and performance goals.</p>	
Keywords	Firebase, Express, React, Node, full stack, JS

Contents

List of Abbreviations

1	Introduction	1
2	Full stack JavaScript	3
2.1	Introduction to JavaScript	3
2.2	Advantages of full stack JavaScript	4
3	Firebase	6
3.1	Introduction to Firebase	6
3.2	Notable features	6
3.2.1	Firebase authentication	6
3.2.2	Real-time database	7
3.2.3	Cloud storage	8
4	Express	9
4.1	Introduction to Express.js	9
4.2	Middleware	9
4.3	Templating engines	10
5	React	12
5.1	Introduction to React	12
5.2	Outstanding features	13
5.2.1	Components	13
5.2.2	Virtual DOM	15
5.2.3	Lifecycle methods	16
5.2.4	JSX	18
5.3	Redux	18
6	Node.js	20
6.1	Introduction to Node.js	20
6.2	Node.js's performance	20
6.3	Node architecture	22
6.4	Node package manager (NPM)	23

6.5	Node.JS event loop	23
7	Implementation of the application	25
7.1	User interface	25
7.2	Database	28
7.3	Authentication and authorization	28
7.3.1	User signup	29
7.3.2	User authentication	31
7.4	Server	33
7.5	Directions API	34
8	Conclusion	36
	References	37

List of Abbreviations

API	Application Program Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
NPM	Node Package Manager
HTML	Hypertext Markup Language
JS	JavaScript
JSX	JavaScript XML
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
XML	Extensive Markup Language
UI	User Interface

1 Introduction

Since the first website was published in 1991, web development technology has grown tremendously and overcome huge obstacles. Current websites have evolved far beyond the first website, which had nothing except a little text and some links (Shontell 2011). This development has required web technology to expand exponentially to make modern websites load faster, be mobile-friendly, and provide easy and simple user experience. In general, web technology is used to improve the appearance and behavior of websites.

To meet the criteria set for modern websites, the combination of HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) and JS (JavaScript) which was common in the past is no longer adequate. Web developers now enjoy the emergence of sophisticated and time-nested stacks, such as MERN, to deliver rich user experience. The MERN stack comprises of four main web development technologies: MongoDB for data storage, Express for routing, React for the frontend and Node for the service layer. In the final year project discussed in the thesis, the styling of the website has support from CSS while data management and user authentication utilize the help of Firebase instead of MongoDB. In the limited scope of this thesis, only four main technologies as well as their relevant supporting libraries are discussed and explained thoroughly.

The primary goal of this final year project was to develop a food application for a startup. The application aims to help food lovers decide where to eat because the huge number of restaurants and the diverse cuisine they provide may make the decision annoying and time consuming. By providing a more comprehensive user experience and a new marketing strategy, the company hopes that this brand-new application can compete with other famous established food applications such as Resq, Wolt or Foodora. This goal required the web development team of Kunam Oy, the company commissioned to develop the application, to build a competitive application while keeping its budget at the minimum level due to the fact that startups usually cannot keep the flow of investment money for a long time. Therefore, the choice of the technology used in this project relied heavily on open-source software.

The thesis first explains the technology used to build the application, mostly each component of the MERN stack, except for Firebase replacing MongoDB. Then it provides a closer look into the deployment of the application.

2 Full stack JavaScript

2.1 Introduction to JavaScript

JS powers almost the entire Internet because it is one of the trendiest programming languages in the world right now. Its popularity is growing more rapidly than that of any programming languages at the moment and multinational conglomerates such as Walmart, Netflix and PayPal build their entire applications around JS.

At the beginning, JS was solely utilized inside browsers to create interactive web pages. When it first appeared in 1995, it was referred to as a toy language. Since then JS has grown thanks to its enormous community support and investment by leading technology firms such as Google and Facebook. (Haverbeke 2018.) Nowadays, JS allows building fully developed web or mobile applications as well as video streaming services command-line tools and real-time networking applications such as chats or even games.

JS was initially designed to work solely in browsers, so every browser has a JS engine to execute JS code. For example, the JS engines in Firefox and Chrome are SpiderMonkey and v8 respectively. For the front-end web development, it allows having functionalities such as dragging and dropping, handling events when clicking. (Haverbeke 2018.)

Starting from 2009, JS can also run with Node outside of a browser. This invention let JS code be passed to Node for execution to build backend for the web and mobile applications (Haverbeke 2018). JS on the backend can access the database, manipulate data and send that back to the front-end through an API.

While ECMAScript is just a specification, JS is a programming language that conforms to this specification. An organization called ECMA is responsible for defining standards and managing ECMAScript specification. The first version ECMAScript was released in 1997. Then starting from 2015, ECMA has been working on annual releases of a newest specification. In 2015, they released ES2015 also known as ES6. This specification defines many new features for JS. (Balter 2019.)

2.2 Advantages of full stack JavaScript

The first advantage of building the entire application in JS is the fact that using a common language allows the web development team of Kunam Oy to understand the source code better. A common language erases the gap between the front-end and back-end team. Therefore, the development team can be combined into one instead of two, which lowers the cost and reduces the effort of maintaining the team. In addition, using the same language helps to save time through code reusing and sharing. The same utility such as libraries, templates or models can be used on the server and browser. Decreasing the number of lines of code is a great asset in terms of maintaining and refactoring the source code. (Bush 2016.)

Another advantage of utilizing JS is its popularity and wide usage. Looking for an experienced JS developer is relatively easy thanks to its huge number of followers. The 2019 Stack Overflow annual survey indicates JS is the most dominant programming language as 67.8 percent of all the respondents use it, which is equivalent to 23,000 developers on this website alone. Considering the fact that Stack Overflow users account for only 0.4 percent of over 20 million developers in the world, it is estimated that there should be over 10 million JS developers. (Stack Overflow 2019.)

Programming, Scripting, and Markup Languages

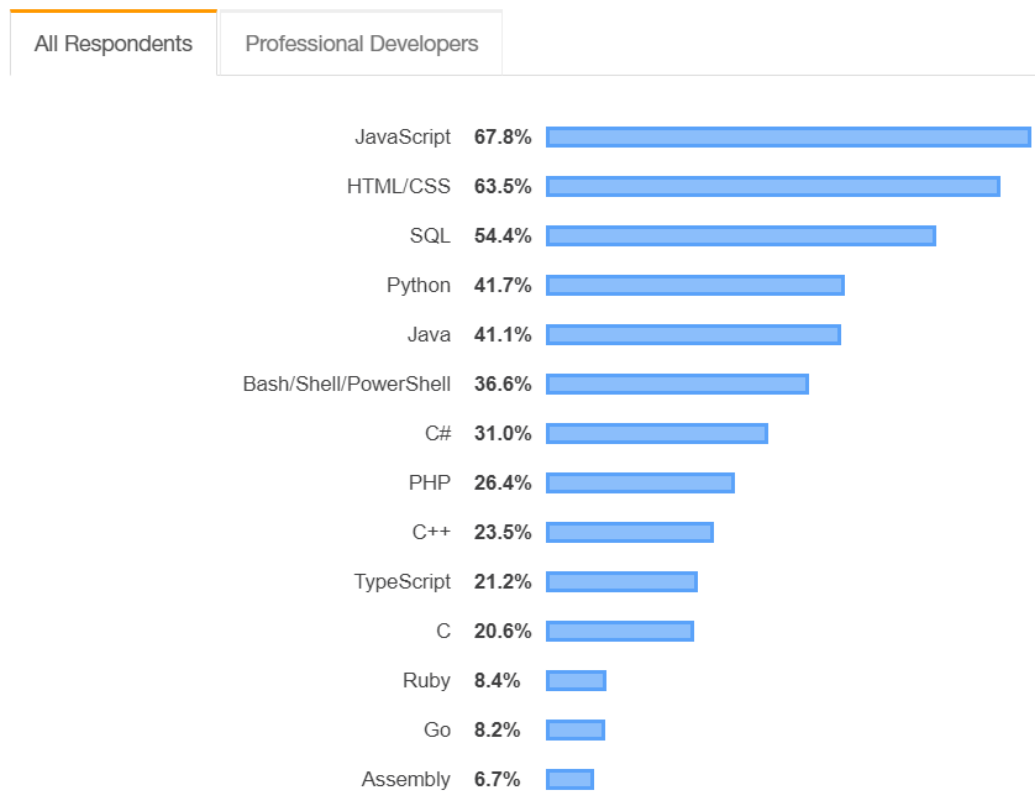


Figure 1. Most commonly used programming languages according to Stack Overflow users. Modified from Stack Overflow (2019).

Because of its powerful and evolving community, JS can provide an extensive knowledge base and high active community support. Its free and open source development tools are among the largest in the world, which means that JS projects do not need expensive licenses and subscriptions.

3 Firebase

3.1 Introduction to Firebase

In April 2012, Firebase, Inc., launched the beta version of Firebase called the Firebase Realtime Database. It was an API that synchronized data from Android, iOS or Web applications and stored it to Firebase's cloud. One notable feature was that it helped software engineers to collaborate in building real-time applications. After the beta launch, Firebase secured important investments to develop two more products in 2014, Firebase Hosting and Firebase Authentication. These products cemented the company's position as a mobile backend as a service. Seeing its potential growth, Google acquired the company in 2014. (Houssein 2017.)

Starting from 2016, Firebase has been marketed by Google as a way to generate income for application developers. Earlier in 2006, Google released AdMob, a mobile advertising platform. The combination of Firebase and AdMob provides more revenue from in-app advertisements, user experience improvement, fast scaling and monetization reports. However, Firebase is designed not only for mobile application developers to earn income. It can be used for other purposes, notably free services such as Cloud Messaging, Remote Config or Crash Reporting. (Google n.d.)

3.2 Notable features

3.2.1 Firebase Authentication

Nowadays, many web or mobile applications allow only authorized users to access their services to secure their data and identify users. For example, some Google applications require an account and password to log in. Because of third-party authentication APIs, the authentication process is getting even more complicated (Smyth 2017). Firebase was created with the goal of simplifying the authentication process by providing a convenient API which allowed users to log in from federated providers such as Facebook, Google, GitHub, Twitter etc. If Firebase is already manipulated by developers, the sign-in process does not need to be repeated as it is integrated with these providers (Moroney 2017).

Some standard functionalities such as finding a forgotten password are supported by Firebase. In addition, it also supports Smart Lock that allows credentials to be automatically saved to sign in.

3.2.2 Real-time database

Although the real-time database is a simple feature using an API, it is the main product of Firebase by overcoming the real-time obstacle of providing a reliable database system. By synchronizing new data as JSON in all platforms, Firebase creates one real-time database instance and notifies all connected devices to update the newest information and share the update to one another in milliseconds. When a device goes offline or a user loses their connection, data remains available on Firebase Realtime Database SDK servers and changes are stored into the local cache. In this way, when the device is connected again, it can update all the changes made during the offline period automatically. Allowing access to data from multiple devices imposes a huge risk of security. Accessibility and validation of data are defined and secured by Firebase Realtime Database Rules that is activated whenever the data is read or modified. The security rules are securely stored alongside the real-time database in the servers of Firebase. It is advisable to define user access hierarchy and structure it in Firebase. (Google 2019.)

No application server and no server maintenance or operations are required when using Firebase Realtime Database. It can be accessed easily from all common platforms with only a few lines of code because it is hosted in the cloud. This feature lets developers create good user experience without sacrificing the application's responsiveness.

Implementing Firebase Realtime Database can be done through a few steps. First, the application needs to be integrated with the Firebase Realtime Database SDKs. All clients will be quickly included through Cocoa Pods or Gradle. The next step is to reference JSON data to set data to create Realtime Database References. These references can be used to write data or listen out for changes. Finally, Offline Persistence and Firebase Realtime Database Security rules are enabled to provide security and offline availability. (Google 2020.)

3.2.3 Cloud storage

Storing data in the cloud benefits users by many useful features such as sharing data for in-app collaboration or accessing data from multiple devices. Normally, building cloud storage is not an easy task because many servers need to be set up to host the data and be kept running constantly. Dealing with networking issues, scalability and offline availability is time-consuming. Firebase eases this tedious task by offering Cloud Firestore.

Cloud Firestore has powerful querying and fetching capabilities that let developers structure their data in a sensible way. It is also completely up to developers to request and fetch data manually or let Cloud Firestore automatically fetch changes from the database in near real time. Users of Cloud Firestore also get Google Cloud Platform's powerful database infrastructure with features such as multiregional data replication for extra reliability, strong consistency and multi document transactions. (Google 2019.)

4 Express

4.1 Introduction to Express.js

Various frameworks exist for building web applications and web servers on top of Node such as Koa.js and Happy.js. The most popular one is Express with over 10 million weekly downloads. Express.js or simply Express is a minimal, unopinionated and flexible framework to build web applications based on the Node eco system (npm, Inc. n.d.). Thanks to its advantages over other frameworks and popularity, Express has been claimed the indisputable standard server software for Node.js (Hahn 2016). Working only as a thin layer above base level web features of Node.js, Express helps to create single-page, multi-page, and hybrid applications. The same back-end logic can be applied in plain Node but working with Express gives higher securities and shorter lines of code making applications run faster and providing more access to third-party plugins.

In 2010, the first version of Express.js was created and released to GitHub by a prolific open source contributor named TJ Holowaychuk. Four years later in 2014, the ownership of Express was transferred from Holowaychuk to StrongLoop, an American startup with a strong foundation on Node.js (StrongLoop n.d.). IBM then bought StrongLoop in 2016 and put the Express project under the Node.js Foundation Incubator Program to maintain its stability and further enhance its competitiveness. (Krill 2016.)

4.2 Middleware

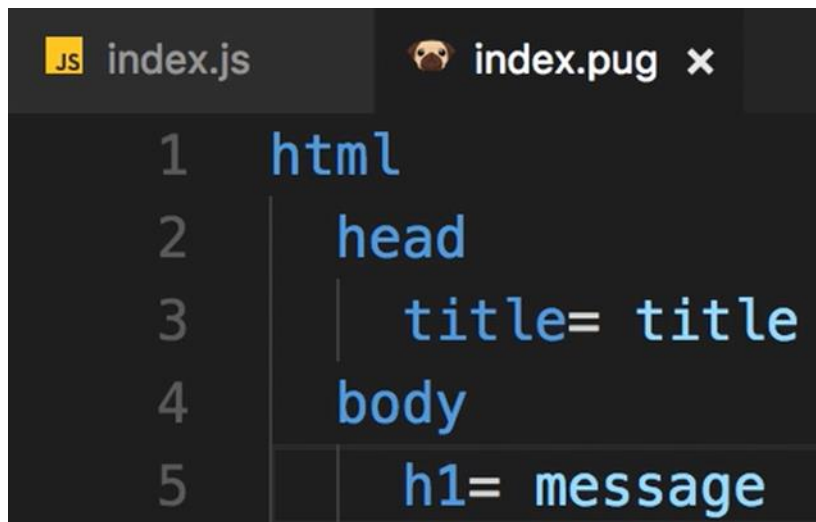
One of the core concepts in Express is middleware or the middleware function. On runtime, when a request is received on the server, it will go through a request processing pipeline. This pipeline includes one or more middleware function(s). Each middleware function either terminates the request-response cycle by returning a response object to the client or passes its control to another middleware function. In Express, every route function is technically a middleware function because it takes a request object and returns a response to the client. Express includes a few built-in middleware functions. Custom middleware functions can also be created at the front of the request processing pipeline. (npm, Inc. n.d.)

One of Express's built-in middleware is the `JSON()` middleware. It reads a request and if there is a JSON object in the body of a request, it will parse the body into a JSON object and then it will set the `req.body` property. Another similar middleware function is called `urlencoded()`. When this function is called, it will return a middleware function to parse incoming requests with URL encoded page load. `static()` is a built-in middleware commonly used to serve static contents. All the static assets such as CSS, images, readme text file, etc. are put inside a folder and this folder is later passed into the `static()` function. The name of the folder is not necessarily included in the URL because the static files are served from the root of the site. (npm, Inc. n.d.)

A list of useful third-party middleware can be found on the Express.js' official website. Among them, `Helmet` will help secure the application by setting many HTTP headers. `Morgan` is another helpful middleware to help log HTTP requests. When `Morgan` is installed, it will be logged to console whenever there is a request to the server. The form in which a request is logged and the place where it is returned can also be easily configured with `Morgan`. Adding `Morgan` to the working features will impact the request processing pipeline, so its usage in production should be done with caution. (StrongLoop, IBM 2017.)

4.3 Templating engines

When there is a need to return HTML markups to the client, various templating engines can be used with Express such as `Pug`, `Mustache` and `EJS`. Each templating engine has a different syntax when generating a dynamic HTML and returning it to the client. In this final year project, `Pug` was chosen because of its widespread use and cleaner syntax than that of regular HTML. There is an example below of using `Pug` to define a template without using opening and closing elements.



```
JS index.js  pug index.pug x
1  html
2    head
3      title= title
4    body
5      h1= message
```

Figure 2. An example of defining a template by Pug.

5 React

5.1 Introduction to React

React is a library based on JS and it is used to build fast and interactive UIs. It was initially developed by an employee of Facebook named Jordan Walke in 2011. At that time, Facebook Ads Org oversaw building client-side applications with the traditional MVC model, two-way data binding and templates. These applications were built by a simple logic: views would listen out for changes on the models and then respond to the changes by updating themselves manually. However, as the number of features increased, the applications became more complicated, ran slower, and required more people to maintain them as a result. Later, the engineer team of Facebook realized that even a small change deep in the structure of one application could cause the whole application to re-render. Internal cascading updates slowed down the team and wasted a lot of their work because engineers could not keep track of all the changes and reasons why they happened. The React team attempted to solve this problem by embedding HTML inside JS. Under the hood, they introduced XHP into the PHP stack. It took a piece of code such as small XML fragments inside PHP code and turned these declarations into function calls. One of the most important benefits of this approach was that it prevented cross-site scripting attacks. In addition, XHP allowed software developers to create composite components. These components not only sent out markup necessary to render a view but also all style, presentation information and JS code needed. It meant that React wrapped an imperative mutation of API with a declarative one that favors immutability. This approach made the code predictable by allowing developers to understand all the possible states of any component when opening a JS file containing that component. This powerful feature of React made its applications more reliable and maintainable and increased its popularity. (Occhino 2015.)

Currently React.js is among the most widely used JS libraries for building UIs. As shown in the figure below, the number of React downloads in the last 6 months stably has tripled the total amount of downloads of all other popular frameworks such as Angular and Vue (npm trends website 2020).

Downloads in past 6 Months ▾

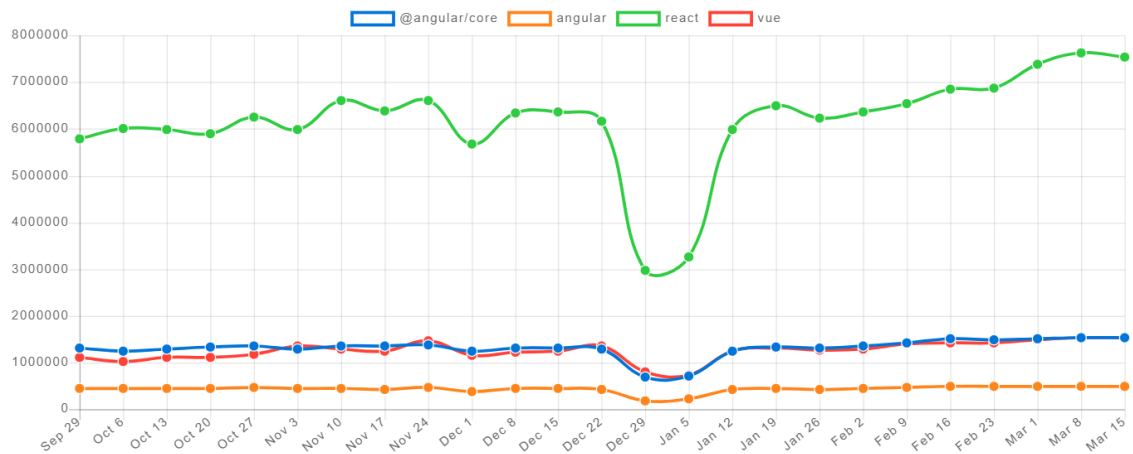


Figure 3. Number of downloads of some popular front-end frameworks. Modified from the npm trends website (2020).

5.2 Outstanding features

5.2.1 Components

At the core of all React applications are components. Simply speaking, a component can be understood as a piece of UI. To build an application with React means to build several independent, isolated and reusable components and then compose them to build complex interfaces. Every React application consists of at the minimum one root component called `App.js`. This component acts in place of the whole application and includes all other child components, meaning that every React application can be understood as a tree of components. This tree component concept can also be found in other popular frameworks such as Angular 2 or higher. In terms of implementation, a component is usually built as a JS class that has some states and a `render()` method. The state here is the data that will be displayed when the component is rendered. The `render()` method is in charge of describing what the UI should look like. (Facebook, Inc. 2020a.)

Inside `React.js`, components are mostly declared by class-based components, known also as stateful components or container/smart components, because they hold values through their state and pass data to child components via props. However, when a component has only one single method and no other helper methods, event handlers and no

state, a function can be used to declare the component. They are called stateless functional components or presentation/dumb components.

```
class Parent extends Component {
  constructor() {
    super()
    this.state = {
      books: [],
      favoriteAuthors: []
    }
  }
  render() {
    return (
      <div>
        <Books books={this.state.books} />
        <FavoriteAuthors favoriteAuthors={this.state.favoriteAuthors} />
      </div>
    )
  }
}
```

Figure 4. An example of a stateful component

Figure 4 shows a stateful component called 'Parent'. Parent has its own state defined in the constructor to hold its internal values and props to pass data to its children. In contrast, figure 5 shows the BooksList component that has only one method and no state or props. Booklist can be defined as a stateless component.

```
const BooksList = ({books}) => {
  return (
    <ul>
      {books.map(book => {
        return <li>book</li>
      })}
    </ul>
  )
}
```

Figure 5. An example of a stateless component

While stateful components observe changing data, stateless components always render the same thing or whatever delivered to them via props. Stateful components are used when data changes dynamically. A user's list of favorite movies or scoreboards serves as an example of this. The tree component can be structured to have one parent component to keep track of all data and pass data to its children stateless components.

State is a special property of the React components. It is an object that includes any data that a component needs. The method `setState()` is used to tell React that the state of a component is going to change. React then schedules an asynchronous call to the `render()` method to return a new React element. In virtual DOM, React will inspect the old and the new component tree to determine what elements in the virtual DOM are modified. It will reach out to the real browser DOM and update the corresponding changes to match what is inside the virtual DOM. Only modified components are updated in the DOM. (Facebook, Inc. 2020b.)

Every React component has a property called props. Props is basically a plain JS object that includes all the attributes set in a component. When passing something between the opening and the closing tag of an element, it automatically creates children props in props. Children are special props and are essentially React elements. (Facebook, Inc. 2020a.)

The difference between props and state can be confusing for React beginners. Props include data that is given to a component while state includes data that is local or private to that component. Other components cannot access that state. The state is completely internal to the component. In addition, props are read-only. Because React will not allow any change to any property of the props object, input to a component cannot be changed inside the component. If input needs to be modified during the life cycle of the component, it is necessary to get that input and put in the state. (Facebook, Inc. 2020a.)

5.2.2 Virtual DOM

This `render()` method produces a React.js element that is a simple plain JS object that maps with a DOM element. It is not a real DOM element, just a plain JS object, that acts

as a representative for that DOM Element in memory. In that way, React keeps a light-weight version of the DOM in memory which is also referred to as Virtual DOM. (Facebook, Inc. 2020c.)

Unlike the browser or the real DOM, Virtual DOM is cheaper to generate. When a state of any component is changed, a new React element is generated. React.js then makes a comparison between this element and all of its children to the previous ones. It realizes what has been changed and updates only a part of the real DOM to keep the real DOM in sync with the virtual DOM. This means that when building an application with React, programmers can avoid dealing with the DOM API in browsers, unlike vanilla JS or jQuery. Programmers just need to simply modify the state of a component and React will automatically update the DOM to match that state. This is the reason why this library is called React, because when the state is changed, React will react to the state changes and update the DOM. (Facebook, Inc. 2020c.)

5.2.3 Lifecycle methods

Components in React go through a few phases in their lifecycles. There are a few special methods that can be added to a component and react will automatically call these methods. These methods are referred to as lifecycle hooks. They allow programmers to hook into certain moments during its lifecycle and do something. The lifecycle methods of React could be divided into three groups in accordance with the three phases of a component's lifecycle as the diagram below shows.

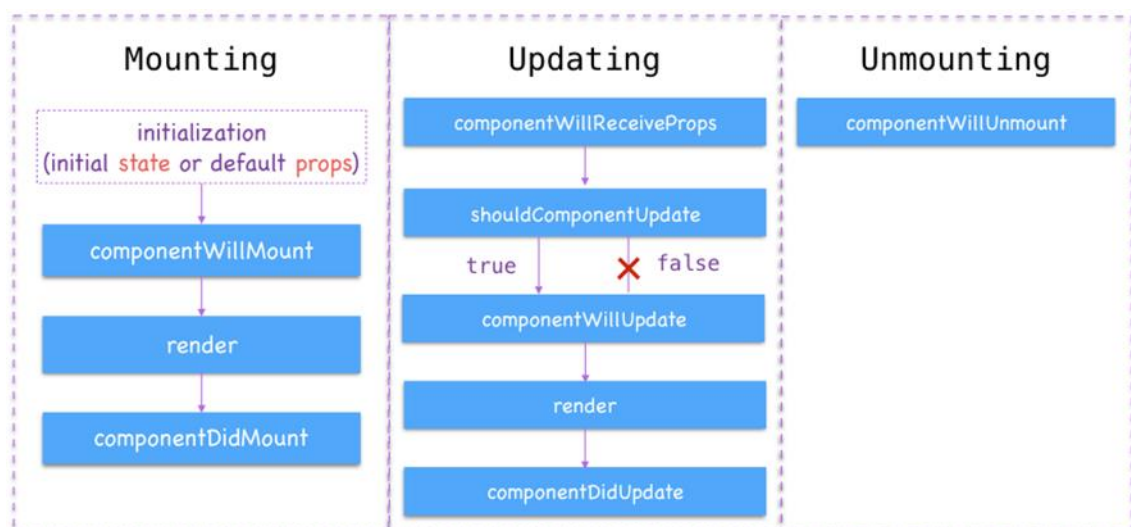


Figure 6. The life cycle of a React component (Codevoila n.d.).

The first phase is the mounting phase when an instance of a component is created and inserted into the DOM. In the mounting phase, there are three lifecycle methods. React will call these methods in order. `Render()` is called when a component is rendered, which returns a React element, which represents the virtual DOM. After that React gets virtual DOM and renders it in the actual browser DOM. When a component is rendered, all its children will be rendered recursively. `componentDidMount()` is called after a component is rendered into the DOM, and it is a perfect place to get AJAX called to get data from the server. When a component is mounted, the component is in the DOM. (Facebook, Inc. 2020b.)

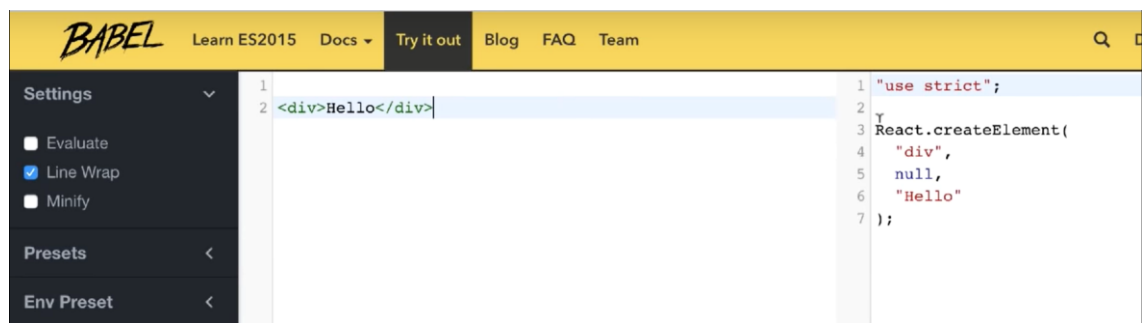
The update phase is next when the props or the state of a component is changed. In this phase, two lifecycle hooks, `render()` and `componentDidUpdate`, are mostly used. Changing the state of a component or giving it new props will schedule a call to the `render()` method so the application is going to be rendered, which means all its children are going to be rendered as well. The entire component tree being rendered does not mean the entire DOM is updated. When a component is rendered, a React element is returned, updating the virtual DOM. React will then look at the virtual DOM. It also has a copy of all virtual DOMs. That is why the state should not be updated directly, so that there can be two different object references in memory. We have the old virtual DOM and the new virtual DOM. Now React will determine what is changed. Based on that it will update the real DOM accordingly. `componentDidUpdate()` is called after a component is updated, which means there is new state or new props, so programmers can compare the new state to the old state and new props to the old props. Furthermore, if there is a change, programmers can get an Ajax request to get new data from the server. If there is no change, there will be no Ajax request. This is an optimization technique. (Facebook, Inc. 2020b.)

The last phase is the unmounting phase when a component is removed from the DOM such as when we delete a component. `componentWillUnmount()` is called just before a component is removed from the DOM. This is the opportunity to do any kind of clean up:

the listener that was called can be removed before the component is removed. Otherwise, there is a risk of memory leaks. (Facebook, Inc. 2020b.)

5.2.4 JSX

JSX stands for JS XML. React uses this HTML-like syntax to extend ECMAScript and allows this XML/HTML-like code to coexist with JS/React code. These HTML-like syntaxes ultimately get transformed into lightweight pure JS objects via Babel.JS which is understood by browsers. Like XML, JSX tags have a tag name, attributes and children. JSX is not mandatory when writing React applications. However, without JSX, it is painful to work with React. As shown in the figure below, the code using JSX on the left is clearer and more elegant than the pure JS code on the right.



The screenshot shows the Babel online playground interface. The top navigation bar includes 'Learn ES2015', 'Docs', 'Try it out', 'Blog', 'FAQ', and 'Team'. On the left, there is a 'Settings' sidebar with options for 'Evaluate', 'Line Wrap' (checked), and 'Minify'. The main editor area is split into two columns. The left column shows JSX code:

```
1
2 <div>Hello</div>
```

 The right column shows the equivalent pure JS code:

```
1 "use strict";
2
3 React.createElement(
4   "div",
5   null,
6   "Hello"
7 );
```

Figure 7. An example of code with and without using JSX

5.3 Redux

When the application grows larger and has several big components, managing individual component's state is difficult and messy. In that case, a state manager will become handy and Redux is one of them. Any frameworks can work with Redux such as React, Angular or Ember. Redux has three main foundations:

1. Store: holds all the states of the application.
2. Action Creator: plain JS objects describe what happened.

3. Reducers: explains what to do when the state changes. (Banks and Porcello 2017.)

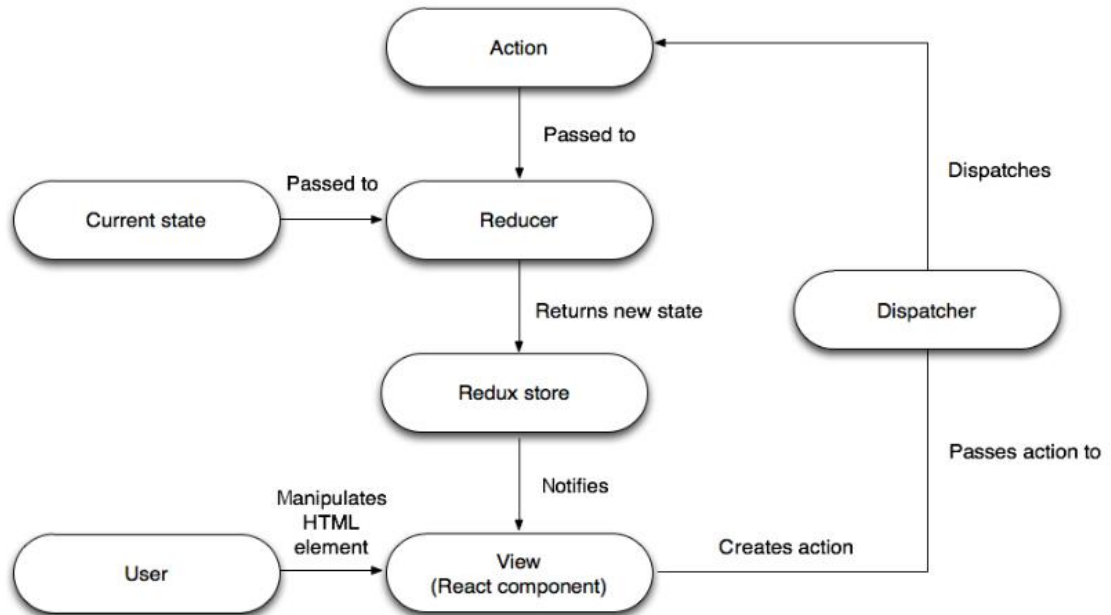


Figure 8. Redux Data Flow (Geary 2016)

As the figure above shows, the Redux store includes the state and then sends the state to View which is basically a React component. Inside the component, for example, there may be a button that is clicked or a form that is going to be submitted. When the button is clicked, an action is created and passed to Action creators, which then dispatches the action to the Store. Reducers are pure functions that describe how the state of the application should change with regard to that action. The reducers respond with a new state. The state is immutable so it cannot be modified so it is recreated as a new state. In turn, the new state is sent down to a component and the component will react accordingly to that state. (Fink 2016.)

6 Node.js

6.1 Introduction to Node.js

Node.js is an open source and cross-platform server environment that takes Google Chrome's V8 engine to execute JS code outside of a browser (Cantelon, ym. 2014). Node is often utilized to create back-end services, also called APIs. Node is most suitable for developing real-time, data-intensive, super-fast and highly scalable back-end services which powers the client applications.

Additionally, it is not difficult to learn to use Node.js and it could be utilized for prototyping and agile development. Huge multinational conglomerates such as Pay Pal, Netflix, Uber and Walmart are using Node.js in their production (Nag 2017). In addition, JS is used in Node applications so programmers with knowledge of JS in the front-end can easily reuse the knowledge and convert to a full-stack developer. In that case, JS is utilized in both front-end and back-end. The codebase can become more consistent and cleaner. Finally, Node.js has the largest collection of free and open source frameworks and libraries available to be used, so whenever the development needs more features or building blocks, there will be always some free libraries that can be utilized.

6.2 Node.js's performance

Although Node was developed later than many of its competitors, Node is not an all-around perfect option to replace other alternatives. In comparison to its other competitors, the speed in which Node.js performs depends on the structure of the program. The same situation also happens with benchmark tests, when the performance of Node is up to the types of operations used in the tests.

In 2013, PayPal conducted a benchmark test between Node.js and Java, the most popular programming language for decades, to decide which one is more suitable for their service. The criteria for the test was to have two separate teams to create the same functionality. The Java-based team used Java together with Spring while the other utilized a Node-based build with Express (PayPal Engineering 2013). The figure below shows the results.

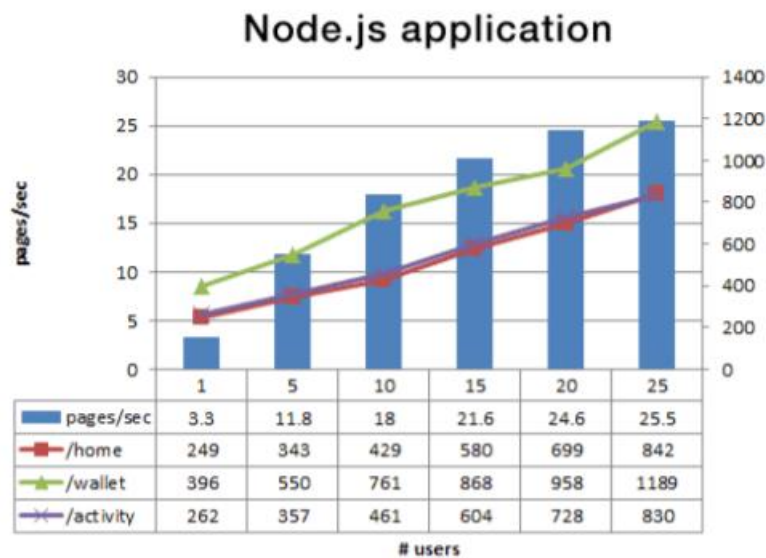
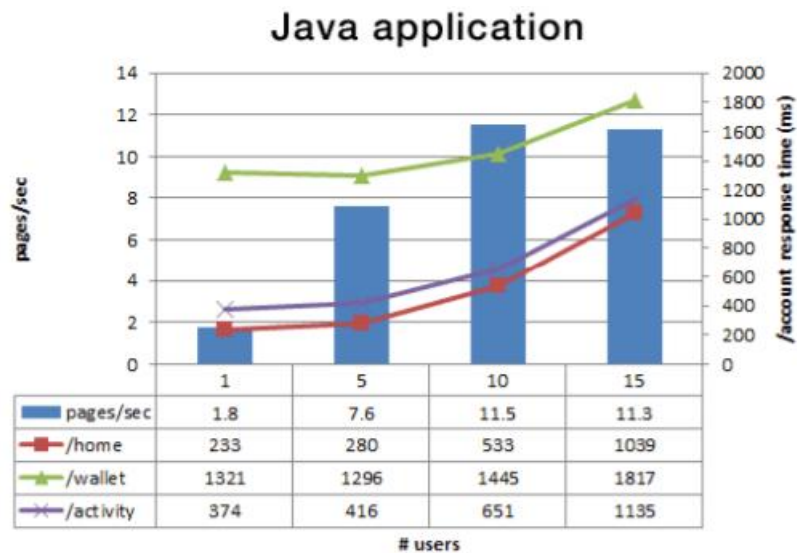


Figure 9. Results of the PayPal Node.js and Java benchmark test (PayPal Engineering 2013)

The results clearly show that Node.js increased the requests per second two-fold compared to Java. Given the condition that the concurrency kept increasing, the gap only grew larger and more noticeable. In addition, PayPal also revealed that Node helped decrease the average response time by 35%, which resulted in the fact that the page was served 200 milliseconds faster (PayPal Engineering 2013).

6.3 Node architecture

Before the coming of Node.js in 2009, JS was solely utilized to develop applications that ran inside a browser. Each browser had a JS engine that took JS code and converted it into machine code, which a computer could understand, examples being Chakra for Internet Explorer, SpiderMonkey for Firefox and V8 for Chrome. Because of this variety of engine, JS code could behave differently from one browser to another. In 2009, Ryan Dahl, the creator of Node.js, came up with a working solution. He felt that it would be better to run JS outside of a browser. Therefore, Dahl took the V8 engine from Google, which was the fastest JS engine at the time, and embedded it inside a C++ program and named the combination Node.js (Dahl 2009). With the same logic as a browser, Node.js acts like a runtime environment for JS code and carries a JS engine which can run JS code. However, it also has certain objects providing an environment for JS code. The objects are different from the environment objects in a browser. Instead of having the document objects, Node contains other objects that give more interesting capabilities such as fs (used to work with filesystems) and http (lists the requests). (Cantelon, et al. 2014.)

Node.js also uses non-blocking architecture, which lets a single thread handle multiple requests. Whenever a request comes, Node.js uses this single thread to handle the request. It is not necessary for the thread to wait for the database to return data. During the time the database is running the query, the single thread serves another client. If the database prefers the result, it will put a message into an event queue. Node.js is constantly keeping an eye on the queue in the background. Whenever there is an event in the queue, Node will take it out and process it. The design of this architecture turns Node.js into a good option for creating applications which include a sizeable amount of disk or network access. The applications can serve more clients without the need to deploy more hardware. That is the reason why the Node application is highly scalable. In contrast, Node is not advised to be used for CPU-intensive applications such as video encoding or an image manipulation service. These applications require several calculations executed by the CPU and a few of them to connect the file system or the network. When the single thread is serving a client that needs a high amount of calculation, other clients must wait. (Cantelon, et al. 2014.)

6.4 Node package manager (NPM)

Every Node application has a `package.json` file that includes metadata about the application. This includes the name of the application, its version, dependencies, etc. NPM is used to download and install 3rd-party packages from the NPM registry. NPM is basically a command-line tool as well as a registry for third-parties libraries that can be added to a Node application. There are currently about half a million building blocks on NPM and all of them are free and reusable code. These building blocks can be easily added to a Node application. All the installed packages and their dependencies are stored under the `node_modules` folders. This folder should be excluded from the source control. Node packages follow semantic versioning: `major.minor.patch` (npm, Inc. n.d.)

Before any Node package is installed to an application, a file called `package.json` must be created. `Package.json` is a JSON file that includes some basic information about the application or project such as its name, its version, its author(s), the address of GitHub repository and its dependencies, etc.

6.5 Node.JS event loop

Typically, in the computer system, input and output are the most expensive operation. The system means to wait for the I/O operation to be completed and it is called blocking I/O and synchronous programming. They involve waiting time, which is usually the performance bottleneck of the system. Node is a very high-performance server. What makes Node.JS so popular and different from others is that it has an event loop. It is a design that makes the execution fast. (Cantelon, et al. 2014.)

First, Node performs a check to determine whether it should run an iteration of the event loop. The condition is that whenever there are one of three types of operations - pending timer operations, pending operating system tasks and pending execution of long running operations - the event loop will start a new iteration as shown in the figure below. (Perry 2018.)

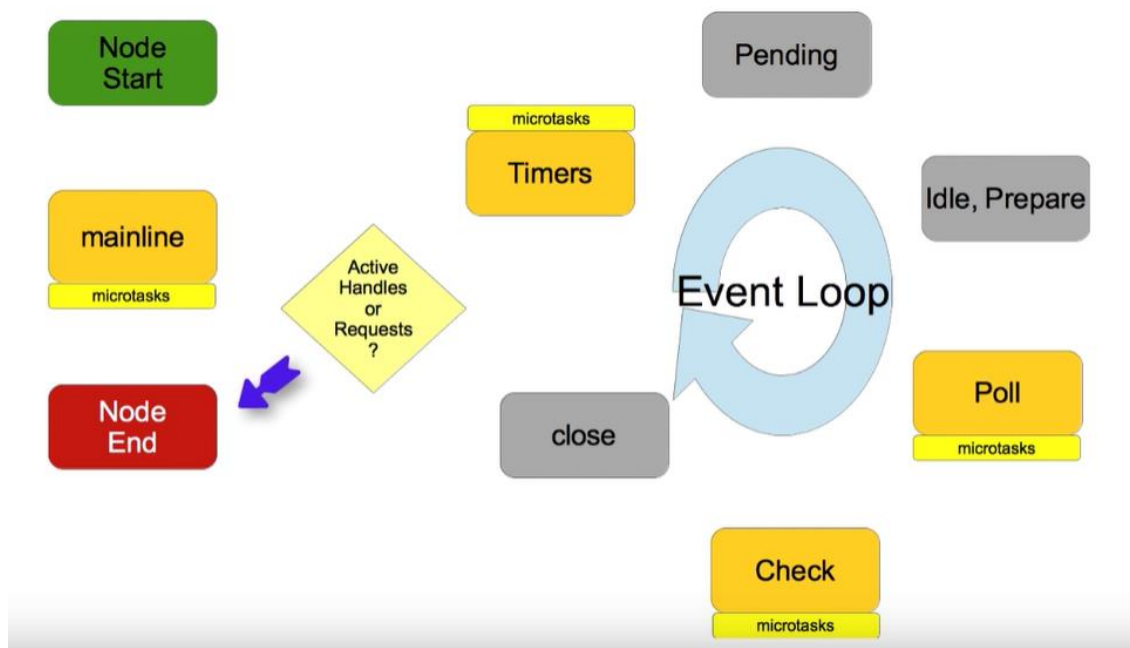


Figure 10. The event loop of Node.js (Perry 2018)

There are five steps that an event loop takes at every iteration. The first phase is called timers. By looking into its inner collection of pending timers, it will determine if there is any timeout ready to be called when its timer has expired. However, technically speaking, when timers are executed is controlled in the poll phase. Secondly, it finds if there are any pending OS tasks such as disk, network and child processes and does more management internally with those events and then calls the ready callbacks. Then it momentarily stops to wait for new events such as a new timer or OS task completion. Fourthly, it goes on with the `setImmediate()` queue and calls any related functions that are ready. Finally, there is an internal phase where 'close' events are created to clean up the state of the application or open sockets. (Perry 2018.)

At the end, it uses the reference counter to determine whether it should exit or keep going. Every time an operation starts, one reference is added to the reference counter. Then when the callbacks come, one reference is subtracted. If the reference counter is zero, then the event loop will exit. (Perry 2018.)

7 Implementation of the application

7.1 User Interface

The interface of the What to eat application was developed entirely by React. When beginning to develop the React application structure for UI, the first step was to divide the application into many smaller parts. Splitting the application into multiple components has more advantages than creating one large component that only serves one purpose. A good application structure with well-splitting components boosts the rate of reusing and repeating components, eliminates code repetition, and provides better unit testing. As shown in the figure below, the main page of the application's UX design is separated into components.

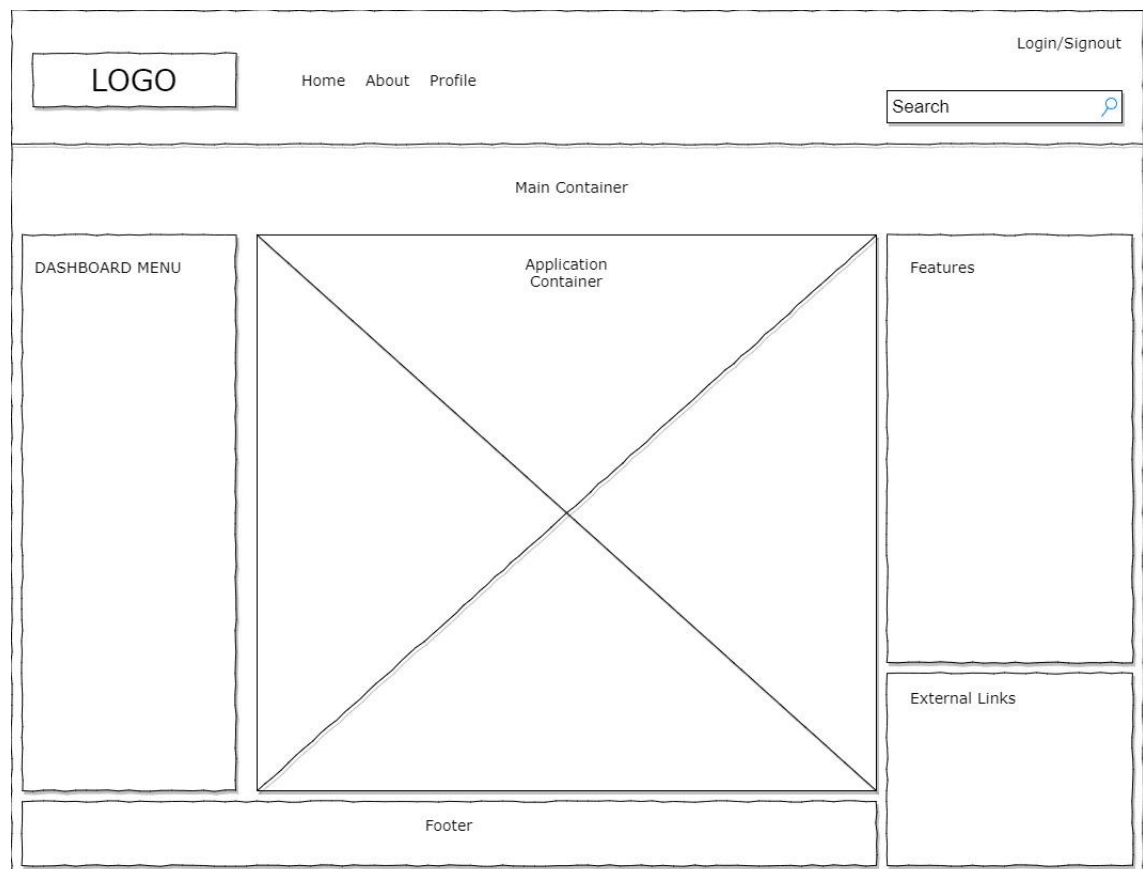


Figure 11. Design of the application's main page

In a React application, the largest component, usually called App.js, appears as the first container. It contains all other components and attaches them to the root node of DOM . In the What to eat application, App.js wraps all child components as shown in the diagram below.

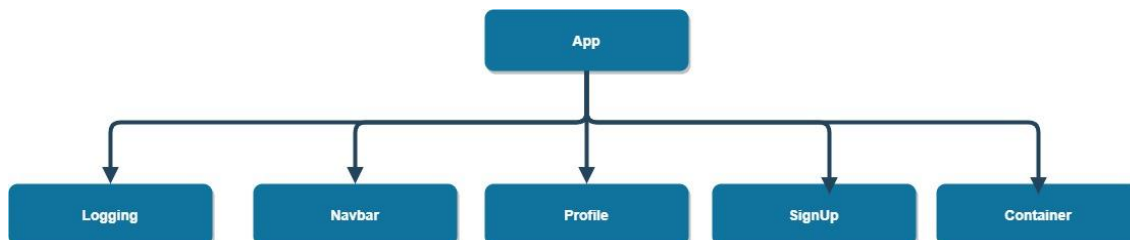


Figure 12.A tree diagram illustrating the React components used in the interface

After considering the application's functionality, the following interfaces were built:

- The login/signup page containing a form for users to fill in their username and password.
- The profile page keeping basic details of users and allowing them to edit the information.
- The main page offering a range of restaurant choices. Users will be offered a pair of choices and they keep picking until there is only one choice left.

The client side of the application is contained in the 'views' folder. From this folder, HTML templates are rendered to the browser and information is displayed to the client. The figure below shows the structure of the 'views' folder. It is divided into sub-folders containing each component of the application.

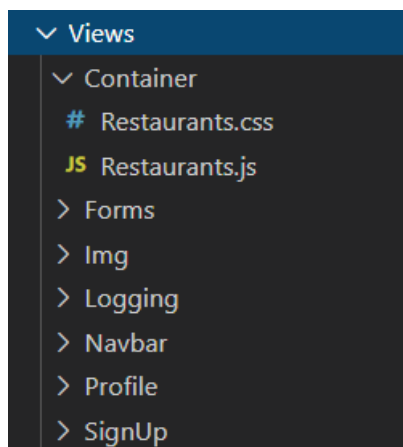


Figure 13.Views folder

Inside the Container folder, the restaurant.js file is in charge of rendering the content of the main page. The figure below shows the dashboard menu of the application.



Figure 14. Screenshot of the dashboard menu.

The restaurant.js file will offer a restaurant choice whenever users click the 'next option' button. After finalizing an option, users can click either two directions button and then the application will show the directions from wherever the users are to the chosen restaurant thanks to Google API. In addition, the users can save their choice for later by the 'Add to favorites' button.

7.2 Database

The application uses the Cloud Firestore database, which is a managed service from Firebase. For initializing and connecting to the database, Firebase is imported to the `base.js` file and then it is automatically included in the bundled JS file. By typing `firebase.initializeApp()` and providing configuration from the Firebase console, React is now ready to get to work with Cloud Firestore.

Cloud Firestore is a flexible and scalable NoSQL cloud database for web, mobile and server development from Firebase and Google Cloud Platform. Through real-time listeners, Cloud Firestore stores and keeps data in sync across client applications. It also provides offline assistance to both mobile and web applications. This feature allows engineers to build responsive applications that work continuously despite network latency or Internet connectivity. Smooth integration with other products from Firebase and Google Cloud Platform is also offered by Cloud Firestore.

7.3 Authentication and authorization

Authentication is the process of identifying if the user is who they claim they are. When a user is logging in, their username and password are sent to the server and the server authenticates them. Authorization is determining if the user has the right permission to perform the given operation. In the What to eat application, the goal is that only authenticated users can perform operations that modify data. If the user is anonymous or they are not logged in, they can only read data from certain endpoints. If the user wants to create their own list of favorite choices or update their preference, they have to be authenticated first. As an additional security feature, only admin users can delete data. This is the second level of authorization.

The application uses the solution from Firebase Authentication for authenticating. Backend services, SDKs, and UI libraries to authenticate users are all provided by Firebase Authentication. A wide range of authentication methods from passwords, phone numbers to popular federated identity providers such as Facebook, Google and Twitter are supported.

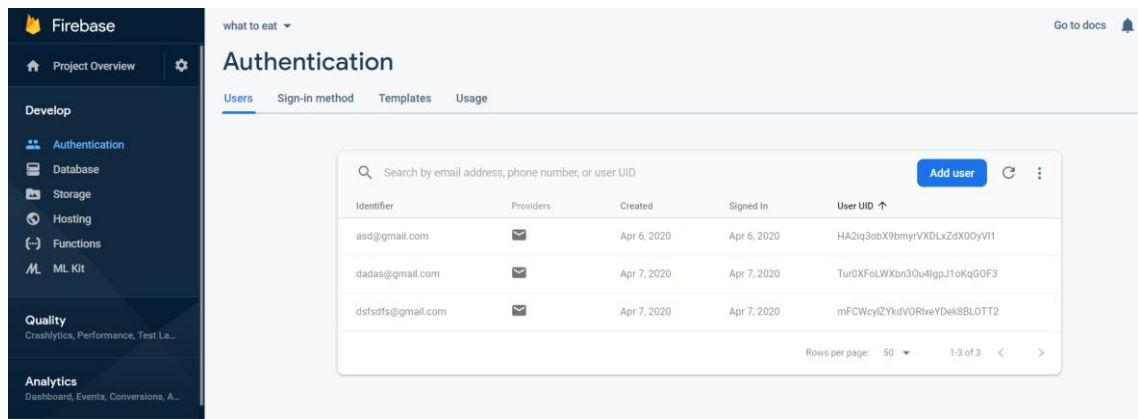


Figure 15. The interface of Firebase Authentication used in the application.

The figure above shows the user-friendly interface of Firebase. Users' usernames and passwords are displayed clearly and can be edited without any difficulty. By designing this attractive and easy-to-use front-end application, Firebase eases the work of a database developer and makes data management more efficient.

7.3.1 User signup

First, a new user needs to create an account to start using the service. The sign-up form requires users to submit their email and password. Since the form stores only two values, it is built by a simple stateless component, illustrated in the figure below.

```
const SignUpView = ({ onSubmit }) => {
  return (
    <div>
      <div className= { styles.form }>
        <h1 className= { styles.header }>Sign Up</h1>
        <form onSubmit={onSubmit}>
          <label>
            Email
            <input
              className= { styles.input }
              name="email"
              type="email"
              placeholder="Email"
            />
          </label>
          <label>
            Password
            <input
              className= { styles.input }
              name="password"
              type="password"
              placeholder="Password"
            />
          </label>
          <button type="submit" className= { styles.signup }>Sign Up</button>
        </form>
      </div>
    </div>
  );
};
```

Figure 16.A stateless component being used to create the sign-up form.

An event is attached to the sign-up button to handle new users. The method `createUser-WithEmailAndPassword()` allows users to create new accounts and also to log in.

```

class SignUpContainer extends Component {
  handleSignUp = async event => {
    event.preventDefault();
    const { email, password } = event.target.elements;
    try {
      await app
        .auth()
        .createUserWithEmailAndPassword(email.value, password.value);
      this.props.history.push("/");
      alert("Created user " + email.value);
    } catch (error) {
      alert(error);
    }
  };

  render() {
    return <SignUpView onSubmit={this.handleSignUp} />;
  }
}

export default withRouter(SignUpContainer);

```

Figure 17. Handling of sign-up event

7.3.2 User authentication

Another form for log in was built to verify login attempts to and from Firebase. Similar to the sign-up form, the log-in form was also created by a stateless component.

To log a user in with an email and password, the `signInWithEmailAndPassword()` method can be called. This method will sign in an existing user and return a promise that can asynchronously resolve the user. Since it is a promise, it will resolve the user's data only one time.

```

class LogInContainer extends Component {
  handleSignUp = async event => {
    event.preventDefault();
    const { email, password } = event.target.elements;
    try {
      await app
        .auth()
        .signInWithEmailAndPassword(email.value, password.value);
      this.props.history.push("/");
    } catch (error) {
      alert(error);
    }
  };

  render() {
    return <LogInView onSubmit={this.handleSignUp} />;
  }
}

export default withRouter(LogInContainer);

```

Figure 18. Handling of log-in events

To monitor the authentication state, a different method named `onAuthStateChanged()` is used in `componentDidMount()`. `componentDidMount()` is a great place for the real-time database. If the listeners are set up in `componentDidMount()`, an initial set of data is allowed to be set and the data is asynchronously downloaded from the real-time database and then is rendered to the DOM.

The method `onAuthStateChanged()` takes in a callback and this callback fires off every single time the authentication state changes. A user logging in or logging out will always trigger this callback function. In the case that a user logs in, the user parameter will be populated with the current user's information. If the user logs out, the user parameter will be null.

```
componentDidMount() {
  app.auth().onAuthStateChanged(user => {
    if (user) {
      alert("Logged in as " + user.email)

      this.setState({
        authenticated: true,
        currentUser: user,
        loading: false
      });
    } else {
      this.setState({
        authenticated: false,
        currentUser: null,
        loading: false
      });
    }
  });
}
```

Figure 19. Implementation of the onAuthStateChanged() method

7.4 Server

When the Express module is loaded, it will return a top-level function and it is named 'express' by convention. This function returns an object of the Express type named 'app'. This app object has several useful methods such as get(), post(), put() and delete(). The get method is used to implement a couple of endpoints that respond to an HTTP request. This method takes two arguments. The first argument is the path or the URL. The second argument is a callback function that will be called when there is an HTTP request to the endpoint defined in the first argument. The callback function should have two arguments: req(request) and res(respond). Whenever there is an HTTP get request to the hello page, the res object will return a 'Hello from Express' message.

```
const express = require("express");
const path = require("path");

const app = express();
const port = process.env.PORT || 3002;

// API calls
app.get("/hello", (req, res) => {
  res.send({ express: "Hello From Express" });
});

if (process.env.NODE_ENV === "production") {
  // Serve any static files
  app.use(express.static(path.join(__dirname, "client/build")));

  // Handle React routing, return all requests to React App
  app.get("*", function(req, res) {
    res.sendFile(path.join(__dirname, "client/build", "index.html"));
  });
}

app.listen(port, () => console.log("Listening on port ${port}"));
```

Figure 20. Implementation of the server

As shown in the figure above, to listen on a given port, `app.listen()` is called.

7.5 Directions API

After choosing the restaurant, the route from the client's current position to the restaurant of their choice will be shown. To implement this function, Google's Directions API is used to show directions between locations. Many types of transportation are provided such as public transit or cycling, walking and driving.

The directions API can be accessed through an HTTP interface. The requests are constructed as a URL string. Text strings or latitude/longitude coordinates can be used to identify the locations, along with the API key. The request can be simply tested by entering the URL directly into a browser. If the URL is correct, after calculating directions, the API will return the most efficient route.

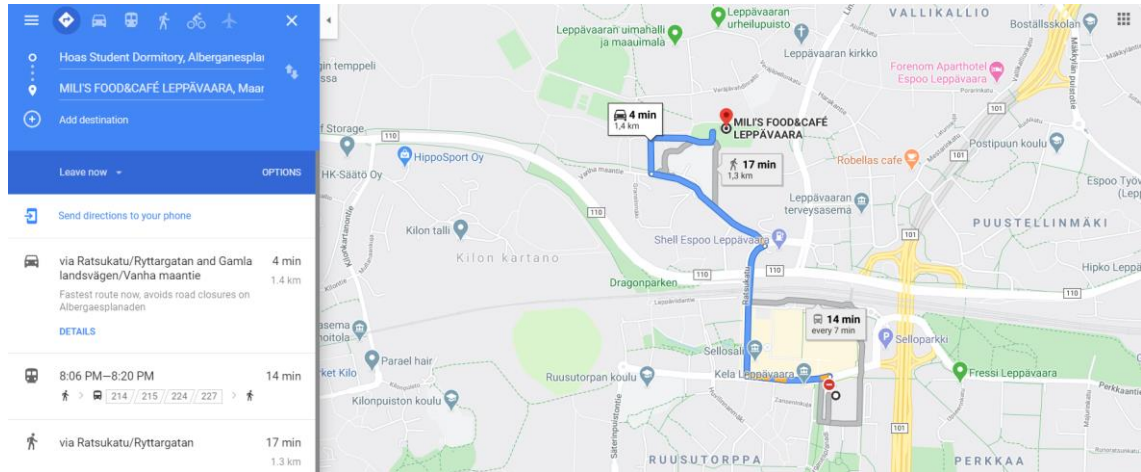


Figure 21. The directions API showing directions from the current position of a client to a restaurant of their choice.

When showing the directions, the applications offers two choices for clients. If the current position is provided, the directions will be shown as in the figure above. However, if the current position is not provided or cannot be located, the origin will be left empty for clients to fill in on their own.

8 Conclusion

The main objective of this final year project was to develop a food application based on full stack JS. Therefore, it gives technical details about the application and analyzes each component of the stack under the hood. Other relevant plugins are also mentioned and explained throughout the thesis. A great deal of time was spent to explore how full stack JS works. In particular, each component of the full stack technology used in the project was studied: Firebase, Express, React and Node were carefully examined. The advantages and disadvantages of each component were analyzed and compared to other alternatives.

JS was chosen for this project because it is one of the best languages for building web applications because of its large ecosystem. JS alone can provide all the components necessary for web development such as Node.js and Express.js for the back end, React and Angular for the client side, and MongoDB or Firebase for the database. In addition, a wide variety of open source libraries and frameworks from Node.js offers more choices for web developers and lets them expand the Node.js services easily. Moreover, the evolution of JS with new features and ECMAScript added every year demonstrates that it has a bright future ahead.

After the first development phase, the What to eat application could provide basic features such as choosing a restaurant and showing directions. The application was tested by a small number of real users and their feedback was positive. The application is at the beginning stage and still under development. Based on the feedback collected, many features may be added in the future. The application satisfies the goal set for it in the beginning. The idea behind the application is great and really solves a problem for food lovers. In the future, hopefully it will get the attention and picks up some decent investment to make it a successful product.

References

- Balter, Leo. 2019. "ECMAScript® 2019 Internationalization API Specification." *Ecma International*. June. Accessed April 28, 2020. <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-402.pdf>.
- Banks, Alex, and Eve Porcello. 2017. *Learning React: Functional Web Development with React and Redux*. Newton: O'Reilly Media, Inc.
- Bush, Eric. 2016. *Full-Stack JavaScript Development: Develop, Test and Deploy with MongoDB, Express, Angular and Node on AWS*. New York: Red Sky.
- Cantelon, Mike, T. J. Holowaychuk, Marc Harter, and Nathan Rajlich. 2014. *Node.js in Action*. New York: Manning Publications Co.
- Codevoila. n.d. "React.JS Tutorial: React Component Lifecycle." Accessed April 20, 2020. <https://www.codevoila.com/post/57/reactjs-tutorial-react-component-lifecycle>.
- Dahl, Ryan. 2009. *Node.js introduction*. November 7 and 8. Accessed April 20, 2020. https://www.jsconf.eu/2009/video_nodejs_by_ryan_dahl.html.
- Eloff, Erik, and Daniel Torstensson. 2012. *An Investigation into the Applicability of Node.JS as a Platform for Web Services*. Linköping: Department of Computer and Information Science, Human-Centered systems. Linköping University, The Institute of Technology.
- Facebook, Inc. 2020a. *Components and Props*. Accessed April 28, 2020. <https://reactjs.org/docs/components-and-props.html>.
- Facebook, Inc. 2020b. *State and Lifecycle*. Accessed April 20, 2020. <https://reactjs.org/docs/state-and-lifecycle.html>.
- Facebook, Inc. n.d. "Versioning Policy." *ReactJS official website*. Accessed April 20, 2020. <https://reactJS.org/docs/faq-versioning.html>.
- Facebook, Inc. 2020c. *Virtual DOM and Internals*. Accessed May 15, 2020. <https://reactjs.org/docs/faq-internals.html>.
- Fink, Gil. 2016. "Redux Data Flow with Angular 2." Accessed April 20, 2020. <https://www.slideshare.net/gilfink/redux-data-flow-with-angular-2>.
- Geary, David. 2016. *Introducing Redux*. July 18. Accessed April 20, 2020. <https://developer.ibm.com/tutorials/wa-manage-state-with-redux-p1-david-geary/>.
- Google. n.d. *AdMob Official Website*. Accessed Apr 28, 2020. <https://admob.google.com/home/>.

- Google. 2019. "Firebase Documentation." *Firebase*. Accessed April 28, 2020. <https://firebase.google.com/docs/>.
- Google. 2020. *Firebase Realtime Database*. Accessed May 19, 2020. <https://firebase.google.com/docs/database>.
- Hahn, Evan. 2016. "Express.JS in Action." New York: Manning Publications.
- Haverbeke, Marijn. 2018. *Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming*. San Francisco: No Starch Press.
- Houssein, Yahiaoui. 2017. *Firebase Cookbook*. Birmingham: Packt Publishing.
- Krill, Paul. 2016. "Node.JS Foundation to Shepherd Express Web Framework." Accessed Apr 20, 2020. <https://www.infoworld.com/article/3031686/nodeJS-foundation-to-shepherdexpress-web-framework.html>.
- Moroney, Laurence. 2017. *The Definitive Guide to Firebase*. Washington, USA: Apress.
- Nag, Poulomi. 2017. "How Are 10 Global Companies Using Node.js in Production." *To the New*. November 6. Accessed April 20, 2020. <https://www.tothenew.com/blog/how-are-10-global-companies-using-node-js-in-production/>.
- npm trends website. 2020. "Angular Core vs Angular vs React vs Vue." Accessed April 28, 2020. <https://www.npmtrends.com/@angular/core-vs-angular-vs-react-vs-vue>.
- npm, Inc. 2014. "What Is Npm?" Accessed April 20, 2020. <https://docs.npmJS.com/about-npm/index.html>.
- npm, Inc. n.d. *Express Introduction*. Accessed April 28, 2020. <https://www.npmJS.com/package/express>.
- npm, Inc. n.d. "npm Documentation." Accessed Apr 28, 2020. <https://docs.npmjs.com/>.
- Occhino, Tom. 2015. "React.JS Conf 2015 Keynote - Introducing React Native." Accessed April 20, 2020. <https://youtu.be/KVZ-P-ZI6W4>.
- PayPal Engineering. 2013. "Node.js at PayPal." Accessed April 28, 2020. <https://medium.com/paypal-engineering/node-js-at-paypal-4e2d1d08ce4f>.
- Perry, J Steven. 2018. "Learn Node.JS, Unit 5: The Event Loop." Accessed April 20, 2020. <https://youtu.be/X9zVB9WafdE>.
- Shontell, Alyson. 2011. "FLASHBACK: This Is What The First-Ever Website Looked Like." *Business Insider*. June 29. Accessed April 20, 2020. <https://www.businessinsider.com/flashback-this-is-what-the-first-website-ever-looked-like-2011-6?r=US&IR=T>.
- Smyth, Neil. 2017. *Firebase Essential - Android Edition. Electronic book*. eBookFrenzy.

Stack Overflow. 2019. "Developer Survey Results." Accessed April 28, 2020.
<https://insights.stackoverflow.com/survey/2019#technology>.

StrongLoop. n.d. "TJ Holowaychuk Passes Sponsorship of Express to StrongLoop."
Accessed April 20, 2020.
<https://web.archive.org/web/20161011091052/https://strongloop.com/strongblog/tj-holowaychuk-sponsorship-of-express>.

StrongLoop, IBM. 2017. *Express Middleware*. Accessed April 28, 2020.
<https://expressjs.com/en/resources/middleware.html>.