Högskolan Kristianstad
291 88 Kristianstad
044 250 30 00
www.hkr.se

Independent project (degree project), 15 credits, for the degree of Degree Bachelor of Computer Science (180 credits) with a major in Computer Science

Spring Semester 2020

Faculty of Natural Science

Creating custom APIs to solve shortcomings in existing JavaScript Front-End Frameworks

Ivan Marosan and Yousef Namaz

Author

Ivan Marosan and Yousef Namazi

Title

Creating custom APIs to solve shortcomings in existing JavaScript Front-End Frameworks

Supervisor

Dawit Mengitsu

Examiner

Eric Chen

Abstract

The technology of creating web-based applications has been expanding and making better user interfaces that can support many different focus groups, such as building a website that is flexible and easy-to-use for children and/or the elderly. Another expansion has been in the field of storages of data and connectivity. These technologies, however, have a steep learning curve for new developers. With the rise of these new techniques, a unique contemporary set of skills have been introduced, the Front-End, which is a layer that involves the presentation aspect of an application.

React.js is a popular JavaScript framework for front-end development. It has become the most framework for user interface design. In today's market there are several other frameworks that one can utilize for front-end solutions such as Angular v2 and Vue.js. The results of this thesis is the improvement on creating a developer friendly custom creation of unique elements. By following proper processes when using the API, one can learn and create their own custom API to their own liking. The API created is a basic GUI addition to React, which enables a Log-In interface that can be easily manipulated by following the thesis, and/or by tracing the steps in the code. This is useful for those that want to have a template GUI API.

Keywords

JavaScript, React.js, API Creation, Front-end Framework, User Interface

Table of Contents

Dictionary

Here's a dictionary of abbreviations that appears in this thesis work

API - Stands for **A**pplication **P**rogramming **I**nterface. A set of functions that allows access/features of an operating system, application, class or other services to work with one's own application.

.js - The file extension for **J**ava**S**cript files. Indicates that a particular file or framework uses and runs JavaScript code.

ACM - Stands for **A**ssociation for **C**omputing **M**achinery. A non-profit professional learned society group. Known for their conferences and publications.

IEEE - Stands for **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers. A non-profit association for electronic and electrical engineers.

DOM - Stands for **D**ocument **O**bject **M**odel. It's what allows scripts to make dynamic HTML objects on a web page.

HTML - Stands for **H**yper**T**ext **M**arkup **L**anguage and it's the standard markup language for displaying documents as web pages.

MIT License - A permissive software license which gives minimal requirements in how code can be redistributed. Originated from **M**assachusetts **I**nstitute of **T**echnology

KB - Stands for **K**ilo**B**yte. A measuring unit for bytes. In this case 1KB = 1000 Bytes

Spaghetti code - An informal and pejorative term that means unstructured and generally difficult code to understand or read.

GUI - Stands for **G**raphical **U**ser **I**nterface and it is a graphical representation of an interface

NPM - Stands for **N**ode.js **P**acket **M**anger and is utilised to manage dependencies and packets within Node.js.

JSX - Stands for **J**ava**S**cript XML and is a JavaScript extension utilised within React.js

1 Introduction

This chapter lays the foundation for our thesis, describing the background, the motivation and finally the research questions

1.1 Background

JavaScript is one of the most popular languages out there when it comes to front-end development. There are several factors that contribute to its popularity for the front-end. One of these factors is that it's a dynamic language compared to Java or C who are all statically typed languages. This allows web pages to only run the code when it's initiated rather than have to wait for the compiler to compile all the data before running the web page. Because of this there are several frameworks for JavaScript. Some of the popular frameworks are Angular v2, React.js, and Vue.js. In an article by Doernhoefer [1] due to the popularity of JavaScript and with so many companies having their own implementation of it. There's hardly any standard for the language since there's so many different ways to code JavaScript such as following ECMAScript or utilizing frameworks. As can be seen with the popularity of various frameworks mentioned earlier which are specifically designed for user interfaces.

Despite JavaScript's popularity in front-end development, it has seen an increased usage as it does in back-end development. This is thanks to Node.js. Because of this many JavaScript frameworks such as React utilize Node.js for their server environment [2]. The reasoning to why it's popular is simple because of how JavaScript as a language is designed. Since JavaScript does not require any other special environments for both client side and server side applications like Java has with its Java for client-side and Java Enterprise Edition for server-side. Because of this one can utilize the same language both front-end and back-end. In an article written by Scott Frees, [3] they describe the fact that Node.js makes it possible to have JavaScript run on the back-end means that one can utilize their knowledge of JavaScript front-end development without much difficulties of learning another set of environments.

1.2 Motivation

React.js is one of the many JavaScript front-end frameworks that has recently risen in popularity. As a result of its increase in usage, questions of whether the React library, and it's easy-to-master learning curve, could be manipulated in ways to improve upon it. Problems that could occur during the process of improvement could be learning how the React core works, what could be tempered with, and how one can add their own custom

API to improve on existing features that the library offers.

The purpose of this thesis is to make a custom-made API to improve upon the React core. These improvements are going to strive to make React.js more intuitive to use on the programming side, rather than the user-experience. As such this thesis is going to assume that the reader has no prior knowledge of React.js and how it works. However, this thesis is going to assume that the reader is at least familiar with JavaScript and similar languages, since React.js is a library that's built upon JavaScript front-end development standards utilizing ECMAScript 6 as its standard. Another assumption that this thesis takes into account, is that the reader is at least familiar with the concepts of APIs.

1.3 Research Questions

Question 1:
What methods are utilized for API creation?

Question 2:
How can creating new API towards existing frameworks help developers in their work and development of front-end applications?

1.4 Limitations

Due to the broad scope of effor it takes to improve a library, there are certain things that have to be limited. For instance this thesis is not going to strive to improve the overall security of React.js. JavaScript as a programming language has been known for having various security concerns. In an article by Marchand de Kerchove [4] due to the complexity of web browsers and web standards, this has presented a large and an unexplored area for developers in the field of web applications. As such, with the limited knowledge of the field. This thesis is not going to improve upon security flaws that could potentially be in React.js due to the sheer difficulty of understanding security concepts in JavaScript as well as implementing them in a reasonable time.

Another limitation would be the shortage of time to study and examine the entirety of the React library. Due to this, the focus of this thesis is going to be on the improvement of the interface part of React.js. The reasoning for this is because React.js is primarily used to make user-interfaces [2]. Especially since popular websites like Facebook and Netflix utilizes React.js for their interface. However, this thesis is not about the design aspect of websites, but rather the development of said interface, such as making it easier for a

developer to implement React.s components or make the creation of such components easier for the developer. If successfully implemented it would make utilizing React.js to make interfaces more intuitive and increase the overall efficiency of interface development.

2. Methodology

This chapter is dedicated to the methods that were utilized for this thesis.

This study has been divided into two parts. The first part is literature review, and it's related work and the other part is the proposed method that was utilized for this thesis. More detail will be found at respective sections. The creation of this API falls into a method of science known as case study. The purpose is to investigate an instant or a case where the framework to see how it performs in a given field of work such as performance, interface, memory management, and so forth. The purpose is to obtain knowledge on how it works, why it works the way it does, can this part be used for something else, and etc. The goal for this thesis is to study React.js and afterwards trying to improve upon the framework with our own API. Finally the research questions will be answered with the help of information gathered from the literature study.

2.1 Literature review

The main portion of this study has been utilizing various databases to gather and collect information about React.js, JavaScript and API creations. The databases that were utilized were primarily ACM Digital Library which was provided by Högskolan Kristianstad. However, other databases were used to find information. These were SpringerLink and DiVa, once again provided by Högskolan Kristianstad. Outside of the databases provided by Högskolan Kristianstad; Google Scholar was used as well, which is Google's own search engine for academic text and research thesis which is linked to several academic databases, making it easier to find published peer-reviewed text from other databases such as IEEE. Another part of our literature review was utilizing React.js own documentations. Since these documentations are first-hand sources of how React.js works and behaves. The words that were utilized for finding this information were the following: JavaScript, React.js, API, and creating API. These keywords were chosen for the following reason: *JavaScript* was chosen as a word because React.js is a JavaScript front-end library. Therefore getting the understanding on the possibilities and limitations of JavaScript is important if we want to develop and implement our API. *React.js* was chosen because it is the framework of this thesis. The other two search words *API* and *creating API* goes hand in hand with each other. One was chosen to get a better understanding of what is an API. The other word was chosen to understand the process and mentality required to develop and maintain an API.

The search for information was conducted by looking through the databases while utilizing the words to find relative information. The process afterwards involved filtering through articles and papers to find relative information. The approach utilized was

reading the title and the abstract to see if relative information could be extracted from the articles. The process for this involved reading the articles and taking down notes in a separate document where information was sorted for each section in the article in following order: title, author, introduction, method, results, discussion, and finally relevancy for this thesis. The purpose of this was to keep information separated from each other and make it easy to access without having to go through the whole article. The research questions were answered by utilizing this methodology for literature review.

2.2 Related Work

API creation is a major task. Therefore a part of the literature review was to find theses and articles, and documentations that covers API creation. In an article by Henning[5] there's an underlying problem with trying to take shortcuts or other time saving methods in the creation of an API. Such shortcuts will lead to small problems accumulating over time and in turn degrade the API to become barely functional to what it's striving to improve upon. In order to make a good API there are many equations that one must think of. In Henning's article there are four major points that one must have in mind when creating API. These four points are: to provide functionality, to be minimal without hindering the caller, the need to understand context and design from the perspective of the caller, and the final point is documentation, the API should be documented, way in advance before implementing the API. Work conducted by Stylos, Jeffrey [6] strengthens the need for strong documentation during the development process for API creation. Even proposing various techniques that one could utilize for making quality documentation such as providing placeholder space for documentation utilizing JavaDoc, or utilizing their own proposed solution.

In another article by Dimianidis[7] highlights the various improvements that have been made in the process of web services over the years, and how this improvement contributes to more refined software standards such as various methods of development such as Specification Driven Development and Behaviour Driven Development. Such standards when applied will improve the overall quality of a web service. Furthermore, in their work they describe that with some proper techniques and tools, one could manipulate a RESTful API in such ways that it increased the performance for various web services that utilized the API without having to implement a new function or method.

2.4 Proposed strategy for API creation

The following strategy is a summarization of the various key points learnt on how to tackle the API creation. First point is *“What are we trying to solve?”*. This point is meant to highlight what the purpose of the API is. This is the underlying problem of the framework that needs to be improved, if it needs improvement. The second point is *“Has it already been implemented before?”*. The purpose of this point is to highlight the fact that there’s no point in trying to develop and implement a strategy that already has been implemented before. Therefore, consideration towards other methods, implementations and functions must be taken in consideration before starting so that the API doesn’t become redundant. The third point is *“Can we improve the library in this field of area? Why, why not?”* This point is to see if it’s possible to improve in a given field such as memory management or efficiency in rendering. However, one must take in consideration that there are some areas that can’t be improved upon due to various challenges or difficulties it can pose. Such as, not enough knowledge in a given field or the fundamental design of the library is changed to an extreme that it becomes its own thing rather than an improvement on something that already exists. The final point to highlight is *“Why can’t this library be improved?”* The purpose of this point is a safety net. It is meant to highlight the fact that a given idea may not be feasible to improve due to various difficulties or limitations. Such as it not being possible to develop and make something that is better at this current time with the technology that exists or that the field might get an update during the research and development process that it might change how the library handles various fields that may make the improvement obsolete due to the fact that the previous methods or library may become deprecated.

3. Literature Study

This chapter contains the information gathered during our literature study, utilizing the methods described in the previous section.

3.1 JavaScript

In the words of Mark Doernhoefer [1], JavaScript is a scripting language which is used on the client-side. While the server-side is using programming languages such as Perl, PHP, Ruby, or Python. JavaScript, alongside with HTML and CSS, enables a reciprocal webpage format that is generally viewed in modern websites. In the early 1990s, before JavaScript was created, websites consisted of only texts and colors with no purpose other than being a poster or an advertisement. Brendan Eich developed JavaScript at Netscape, a computer services company, which enables programmers to manipulate web pages' DOM, to exchange data between clients and servers. Some examples of unique interfaces that are driven by JavaScript are web page menus, launching pop-ups, or validating HTML forms, etc.

Despite the functionality and dynamic nature of JavaScript, there are some complex issues that can't be overlooked when working with said language. In a proceedings by Saba Alimadadi [8] they mention that JavaScript has some challenges when one is developing applications for it. This has to do with the fact that JavaScript is a weakly-typed language which will cause errors if the developer is not having this in mind. Especially as Alimadadi[8] mentioned when one is working on more complex events and event handlers. In their proceeding they mentioned 3 areas of problem with JavaScript that causes errors in one shape or another. These errors come from how JavaScript handles the DOM object with events. One of these problems mentioned in the proceeding is that when an event handles a DOM it will put it up the DOM hierarchy tree. However, because of this it can fire off other events as it is climbing up thus causing remote code execution.

3.2 React.js

React.js is a JavaScript library used to make user-interfaces, developed by Facebook[9] and released to the general public in 2013 utilizing a MIT License[10]. The purpose of React.js was to handle the complexity between business logic and the current state of the application. How React.js accomplishes this is by simply re-rendering the whole application and does a diff to see what states have changed. When a diff has been seen by React.js it will update it accordingly to display it to the user. Another aspect of React.js is that it treats the user's code as a black box, this in turn allows React.js to

have any structure to it, however, in the end React.js will always return to the user a virtual DOM. The purpose of this virtual DOM is to see if there's something new that needs to be rendered. Since React will always keep a track of its current representation it will always do a diff with this virtual DOM to see if there is any difference and render the interface accordingly. This form of data-flow is called one-way data flow and it's how React.js handles all forms of data [11].

Another key feature of React.js is that it follows the latest standard by ESCM International of JavaScript which is JavaScript ES6 (which from here on will be referred as ES6 for simplicity). Beyond that, React.js supports other methods that have evolved beyond what ES6 normally allows for, such features are async/await and object rest/spread properties. As such it's supported universally by all modern web browsers. However as stated in the design documents, one must utilize a set of polyfills for older web browsers such as Internet Explorer.[12]

React when compared to its competition of Angular v2 and Vue.js is in between them when it comes to overall performance, such as response time and memory management. In a study conducted by Xing YongKang [13] React.js performance was not much better than the more robust and advanced framework of Angular v2, especially considering that Angular v2 was a much larger framework handling about 143 KB of data volume while React.js only handles 43KB of data volume. While at the same time was overshadowed by Vue.js in both response time and memory management despite Vue.js only handling 23KB of data volume. In this study they conducted they noticed that React.js was the only framework that utilized specifically one-way data flow. Other than Angular v1, both Vue.js and Angular v2 allowed both one-way and two-way data flow. However, at the same time they noticed that React.js is more stable and offers stronger technical support compared to the other two.

3.3 API

An API is a software library containing one or more functionalities to a user's interface made by a third-party software. It is used to expand further upon a software's utilities. Several authors[6] have mentioned that there are difficulties in maintaining good APIs. However, there are plenty of other APIs where one could easily learn to design more appropriately structured APIs. The difference between a good API and a bad one is that a well-constructed API, although difficult to make, can work without any hindrance to the user, that it can call requests for performing tasks, easy-to-remember methods, and well documented. On the other hand, a bad API is made without testing implications of the

methods made or any compensation to check the user's development, such as a method that overrides arguments[5].

Furthermore an API must fulfill two aspects to be considered an API. These two aspects are that it easen up the development cycle for the developers, by providing a set of documentation on what the purpose of the API is and how the API works. The other key aspect of an API is its usage plan. This is due to the fact that API:s are meant to be applied in real-life situations. As such it's becoming more crucial to find a usage for one's API before starting on developing one. Another key aspect about API is the need to limit it's activity to some extent. This is not in large due to security concerns, but from a performance perspective. For an API to be successful, it needs to have certain restrictions or limitations to it in form of quotas or rates. To only allow an X amount of requests at any given time to reduce performance degradation [14].

APIs are used in many different platforms. For instance, creating a link between an interface and an operating system, sending/receiving assets between the client-side interface and the web server using a Web API, or using different protocols to manipulate resources that are remote. In an article about the compliance of API protocols, it is suggested that during the creation of an API, one must ponder upon the challenges around expressiveness, subtyping, inheritance, and aliasing[15]. It is expressed that during development, it is essential for the developers of the API to document, or comment, on their protocols. This is necessary for users to easily identify the right protocols. Aliasing, to check whether the user has examined the protocols or not, is also another essential topic. It is difficult to see the references made for the protocols that are ready for activation when using APIs.

When it comes to developing an API there are two methods of software development that one have to consider. These are two are the waterfall-style method[16] and agile software development. Each of these have their own benefits and drawbacks that will be elaborated on. The first method is known as the waterfall-style method, it is a style of method that views software development as a very linear process where each process is designed into various stages/phases. With this in mind one cannot move on from one phase to the next unless the previous one is 100 % completed. This has natural advantages over the agile style approach since first and foremost the important part is first establishing the requirements for the project before moving onto to design, then implementation, then verification and finally maintaining the project once its finished. However despite this linear approach to it, it does bring some disadvantages to it. One of those disadvantages is simply that with the waterfall-style approach one does not cater or take needs of customer's or client's. Instead one is locked to their model until completion which makes it difficult to make changes when changes arise. Especially in a larger scale project where groups have to be synchronized with each other.

The other method for software development is known as agile software development[17] and it's a flexible way of developing software compared to waterfall. Since instead of following a linear progression, agile is based on iterations and feedback from clients/customers. With this it makes it easier to develop software for an intended group due to the sheer flexibility it offers. In fact, some companies have used agile over waterfall due to the limitations that waterfall poses on development[16]. However, despite the fact that agile does offer flexibility in form of client/customers need and how software development goes with the style of pair programming, there are some disadvantages to agile that does not make it the fully optimal choice. One of the main disadvantages of agile is that it is client/customer based, which makes it difficult to use when one does not have a client that provides feedback. This also plays into another problem with agile since requirements and design is solely on the basis of the client/customer, there is no real base design to work with other than picking what language one wants to utilize.

3.4 Improvements

When it comes to finding improvements for any particular software or framework one must establish what's considered a weakness prior to development. In an article by K.K Aggarwal [18], they mention how quantitative studies in order to measure and quantify how improvements shall be made. Some of the examples they mention are testability and maintainability. In their experiment they utilised the average number of live variables and compared them with the average life of a variable to conclude on how efficiently a software was utilizing its code. In this study they concluded that one thing that when it comes to calculating weakness in software is that the formulas used are usually static ones. They only count for the given formula rather than the scope of the software. Such as the mathematical formulas don't take in account for example things like dependencies that could otherwise fix the weakness in the given software.

In an article by Brad A. Myers and Jeffrey Stylos [19] they mentioned when it comes to improving on APIs and moreover that utilizing an API of some sort can help improve on existing software or framework if the said API follows the standards of API creation: development and implementation. In their article they mentioned that with the right API the development of software does not only becomes easier for the developers due to less time having to be spent on developing their own methods, but also it will help the developers to make error-free software without limiting or hindering their own scope if the API provides sufficient enough methods for the developer. This in turn allows for more testing to be done on the software, less spaghetti code, faster deployment of said software, and importantly increased performance of said software or framework.

Furthermore, they mentioned the one key advantage of making an API is that it's easier to share it around. This in turn allows for multiple softwares or frameworks to use the API as a foundation, and with this allowing people to share their knowledge and understanding of the API and applicable software utilizing the API.

When it comes to finding the right methods for finding shortcomings there are several dimensions one has to think of. In an article by Robert Watson [20], Microsoft themselves utilize a 12 dimension view when it comes to improving on the .NET framework. These 12 dimensions all fill in various needs for developing improvements, and these dimensions cover a wide range from documentation to development, and in some instances covers both of them. With these 12 dimensions one can find the suitable level or field that one is trying to improve upon. Be it the abstraction level, the work-step unit, or something like the overall consistency of the framework. Each of these dimensions covers a specific need and desire for improvement.

3.5 Research Questions - Preliminary

Here are the preliminary answers to the research questions gotten from the literature study and review .

3.5.1 What methods are best utilized for API creation?

When it comes to finding a method for API creation there exist many ways to approach it. Because of this there doesn’t exist one singular best way to approach API creation. But several methods, since it all is dependent on what one wants to develop. As an example mentioned by Watson[18] when it comes to evaluating such things as weakness and improving on said weakness there exists 12 different cognitiative dimensions that one has to look into, and each of these dimensions covers different areas. For instance as mentioned by Watson you have two dimensions that share the same field where they look at i.e. code. But covers different aspects of it, such as abstraction level and consistency of code. Another note to add that looking into all 12 dimensions for improvement is a very time consuming process and thus is not efficient at all. So in conclusion, there does not exist a singular best way or method to find shortcomings in software/frameworks, instead there exists many different dimensions and one has to take into consideration *what* one is trying to improve upon and on what dimension it is applicable in.

Another thing as mentioned in the literature review an API creation has a lot of steps to it. Especially since one important aspect of API creation is the need for design documents to show how and why the API was designed the way it is [6]. Since without proper structure and cohesiveness an API itself will fail to fulfill the demands that it is supposed to meet, and these are that it shouldn't hinder the user of the API when they are developing their application.

3.5.2 How can creating new API towards existing frameworks help developers in their work and development of front-end applications?

Creating a new API is a challenging work, however in the studies has shown that an API can improve reliability in any software or framework. As mentioned previously, a good API reduces the risk of error-prone code, while at the same time improves reliability and performance, another boon that an API provides is that when it has good documentation it is easy to share it around, thus allowing more developers to utilize the API and thus increase the knowledge and application of said API.

4. Case Study

This chapter is dedicated to the case, the tools, and setup that were utilized for the creation of the API.

4.1 The Case

Before being able to implement the API, one must address the issue on what is it that's going to be improved. In this case study it is going to be improving on GUI creation. The reasoning for this is due to functionality of React.js. Because React.js is strictly used for developing user interfaces. However, due to how React.js works and functions, making an interface requires the calling on several components in order to make a suitable interface. In order to improve on this; the improvement is going to be to make a premade interface that one can call on that will make the interface for the programmer without the need to make it manually.

4.1.1 Reasoning

The reason why implementation of GUI components were chosen are in fact due to various factors. One of them is as mentioned previously is that React.js is primarily utilized for GUI components. Therefore that improving, and providing GUI components is something that works well within the React.js environment. Secondly due to the scope of React.js and due to the time limitations of the thesis work, there exists a limited number of options to what one can research/experiment on. Thirdly is that GUI creation has proved difficulties for developers in other areas, not just React.js. In a conference by Lutteroth and Weber [21], they discussed the effect of constraint-based GUI layout such as the benefits it brings to web development. However, due to the fact that doing this manually is an arduous task for many developers due to the inherent complexity that one needs to work with. Therefore they proposed a model for aiding in making complex GUI elements and layouts.

4.2 The Setup

One thing that needs to be established is that React.js on itself does not work on its own, but it contains components that one can utilize for user interface creation. There are two important package managers that one needs before developing for React.js. These two package managers are NPM and yarn. For this set up we decided to utilise JetBrains WebStorm as our IDE. The reasoning for this is because that WebStorm was

specifically made for developing with JavaScript, and offers various tools and functions that just makes development more efficient by allowing installing NPM and yarn within the IDE without needing to use the command line externally. The version of React we used is v16.31.1 and as mentioned, WebStorm streamlines this by allowing one to make a new React App project within the IDE without the need to import it via NPM.

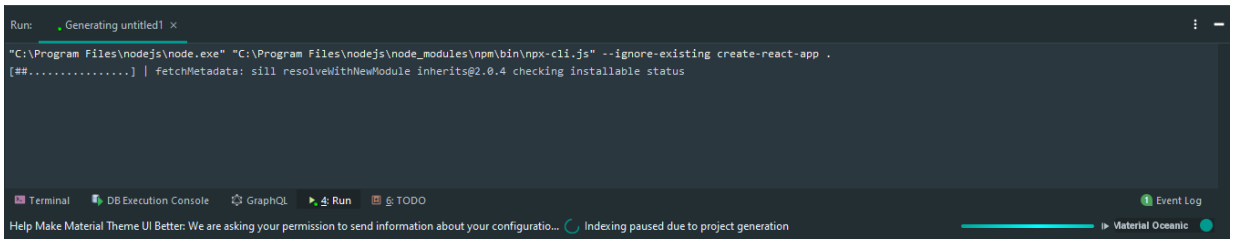


Figure 1: WebStorm initializing and installing React.js

Another feature that is utilised is JSX. JSX is a useful extension that produces components which could easily manipulate HTML elements which will provide aid in creating user interfaces. One thing that must be stressed is that JSX elements are immutable thus we need to be extra careful when utilising them since once they are made they can't be changed.

```
const element = <h1>Hello, world!</h1>
```

Figure 2: A JSX implementation, a static variable with HTML element as value

The purpose of using JSX instead of JavaScript is due to its simplicity and easy-to-understand methods that can help new developers to write and understand code to build well-structured projects and web applications. With only JavaScript, creating large nested HTML documents using JavaScript syntax would make the process harder, as well as making the overall syntax difficult to read and understand[22].

```
const foodName = 'Deluxe Pasta'
const element = <p>Please, try our {foodName}</p>

ReactDOM.render(
    element,
    document.getElementById( elementId: 'root')
);
```

Figure 3: Image showing a declaration of a variable and then used into JSX using curly braces.

```
function formatFood(food) {
    return food.foodType + ' ' + food.foodName;
}

const food = {
    foodType: 'Deluxe',
    foodName: 'Pasta'
};

const element = (
    <h1>
        Please, try our {formatFood(food)}
    </h1>
);

ReactDOM.render(
    element,
    document.getElementById( elementId: 'root')
);
```

Figure 4: Image showing normal JavaScript function embedding into HTML element

4.2.1 Installation

There are two things that one needs before installing and implementing the API. These two things are: Firstly one needs an Integrated Development Environment that supports JavaScript, and secondly one needs React.js and all the necessary setups for it such as NPM and yarn. After this initial setup it is as simple as downloading the API from GitHub and utilizing it within one's project by importing the necessary directory from the API into their own JavaScript files where one wants to utilize the API. Documentation can also be found in the *components* directory if one needs proper help.

4.2.2 Functions and implementations

The functions that were custom class components of various GUI elements. The things that were designed and implemented were primarily focused on a login page. These things include various things necessary for a basic login page, such as custom buttons for potential login, and the other for registering an account. These components are easy to call upon since all the developer has to do is to import them and thus they are readily available for them to use. Also was custom text boxes which provide an ability for the developer to implement text boxes which will accept strings for which they can utilize for their own purpose. Just as with the buttons, a developer just has to simple import these components to make use of them. The third and final component is a layout which one

can use to set up a layout to utilize the other components such as our API or different HTML components that one desires to use.

5 Results

This chapter presents the results that were generated during the thesis work.

5.1 Designing the API

When it comes to designing the API a lot of time went into the semblance of structure. Such as structuring our components into their own directory, documenting what our each class does and what each attribute does, and finally having some sort of cohesion both in looks and code. As such the key importance for success was discussing what was needed to do, what was unnecessary, and what could be done to improve on the existing work. This can be noted in a work by Johan Fredriksson[23] that specifying the top-level design is important, especially in a field of area that’s lacking or is underdevelopment. Another key aspect of this design mentioned by Fredriksson was making sure that there was proper documentation which naturally played a big role in the design of the API.

5.2 Class components

There are two ways of handling the creation of custom components in React.js. These are making a function of the component, which functions just like a normal JavaScript file that accepts props as a valid argument and returns a React element. Needless to say, the function method of making custom components, function however, works just like a normal JavaScript file and thus brings with it the shortcomings that JavaScript has such as that one cannot set the state of one's current component, it is limited to how the function is written. Another limitation with the function component is that functions exclude lifecycle hooks within React.js. Needless to say this limits the creation of what a component can actually do in React.js. However, functions still have the benefit if one writes smaller projects due to less code being needed and therefore is easier to deploy.

The other method is creating a class for the component akin to how one would do in Java by extending to React.components. Class components have the benefit over a function component is that it features various qualities that overall makes the component more efficient to utilize such as mentioned previously the ability to set the state of a component and respecting the lifecycle hooks within React.js. This was much more desired in our implementation since it eliminates the problems that the function method has. Since the importance of being able to set the current state of a component was highly desired.

```
class CustomFoodTitle extends React.Component {
    render() {
        return <h1 style={{color: 'blue'}}>Food</h1>
    }
}
```

Figure 5: A custom React element, for creating a food title, made by using Class component that is extended to React.Component API

5.3 Props

A prop is a HTML attribute in React.js that is used to pass on information that manipulates HTML elements in the React component. This gives custom components unique attributes to them, that if one calls upon the component it will have the prop attached to it. This gives components a unique way to identify from each other. Another aspect of props that one can utilize is the ability to allow the user of the interface to change it's value. There's also a part of React.js that's called *defaultProps* which enables one to define a new standard to a component. This allows one to have a own custom and exclusive view for said component instead of relying on the normal default value of the component. We utilized this to set our own custom components which allowed us to call for them in other .js files.

```
static defaultProps = {
    borderRadius: '5px',
    width: '20%'
}
<input style={{borderRadius, width}}/>
```

Figure 6: Setting default properties for a custom component using **defaultProps**

5.4 Implementation of the custom components

By using a class component that extends React.component, we are able to enter in a render function which allows us to return a HTML element of the desired component. This is where the class component comes into play. Because with the render function it allows the class component to be visible for the user of the interface. This is handled by the render function, one important aspect of this function is the ability to relate to the props of a component. This is done by setting a variable and calling the proper props to said variable. With this in mind one can manipulate the DOM to their own likings. In this case we utilized this function to be able to manipulate the DOM in such a way that we can change the property of the components in real time. Refer to figure 1-4 in the appendix to see how it works in practice and the end result of it.

5.5 The API

As mentioned previously in this thesis, by using class components and with the help of defaultProps we are able to construct an establishment. This allows developers that include the API in the projects to build a technique that allows them to build their own custom-made components. In this API we have made a login page that features the functionality of a login page. This login page is what one would expect of your normal login page, it has two text fields where one can enter their username and password respectively and buttons for logging in and registering an account. With this API one can call upon the custom login page (refer to appendix figure 6) as a scene. This will shorten the line of codes that developers would require to implement due to already being customly made. Due to this, all that the developer has to do is to call upon the custom HTML element in their React app, and that will reveal a custom login as shown in appendix figure 9.

Another aspect of the API is that it utilizes proper structure and documentation. Proper structure of itself is a simple concept that's not too difficult to understand. What it means is that each of the components and files are located in their own directory. This is to make finding the right component or file easier instead of having everything in one single directory. Secondly is documentation, since one important aspect of an API is that it is properly documented. Not only does it help with developers trying to understand how it works and why it works like it does, but it also ensures that the API is preserved. Since one huge issue as mentioned previously in the literature review is that without proper documentation it not only becomes increasingly difficult to understand and use, but also be much harder for the original developers to use since human memory is not reliable enough to store all the information needed.

The key advantage of the API versus without it is that it saves a lot of developers time in the creation of graphical components. While at the same time makes it easy to modify them to suit their own personal needs as can be seen in appendix figure 7 and 8. There's a lot of busy work that one has to do to make graphical components. With this API one has to just make one single line of code to call upon it, for it to display when one is running it. As seen with figure 7 when you're making a call to this simple line of code you will invoke the layout of appendix figure 8. This is the same result as one would do if one made the process of appendix figure 7 and 8. Another thing to take in account is that the API has also predefined methods in it. That one can write their own code directly if one wants to make a certain function behave in a certain way, without having the need for specifying each individual component yourself. Another feature of this API that makes it appealing is that the components themselves are reusable. Thus if one

```
<CustomLogIn />
```

Figure 7: Custom Log-In element to call upon the Log-In GUI

just wants to utilize a button or the layout itself, it is fully possible to just call upon them. As one can see in appendix figure 3 where just the layout and a button is called upon and it provides the results that are in appendix figure 4.

The API can be found here https://github.com/Youzur/ThesisProject and all that the user has to do is download it and move the component directory to their desired project. There's a test app that allows one to see how it looks and how it is implemented.

5.6 Research question - definitive answers

5.6.1 What methods are utilized for API creation?

As mentioned in the literature review part of this thesis work. There exists several ways to implement an API. As mentioned previously there are several ways to approach this. For instance as Watson[20] mentions that one can look and analyze at 12 different dimensions on where one can develop and improve upon existing API. With this in mind one must also look at what approach one is going to utilize for development, as mentioned by Cusamano[16] there exists two popular ways for software development which are waterfall and agile approach. Each of these provide their own advantages and disadvantages for developing an API. Such as having a deadline to which one completes certain tasks, setting priorities, and setting up meetings, discussion and etc. The approach that one chooses depends entirely on how structured one wants to work. With waterfall favouring a top down design, where requirements and design takes precedents, while agile favours a more developing focus design where requirements are made at the end of a meeting. One thing that both agile and waterfall however requires when it comes to API design is that there exists a strong documentation.

3.5.2 How can creating new API towards existing frameworks help developers in their work and development of front-end applications?

A new API creates opportunities for developers. These opportunities include a variety of things and one of them is that a good API doesn't limit their ability development skills and programming abilities. An API can make it easier for a developer to make applications by providing the methods, functions and components readily available for the developers. This in turn means that the developers themselves don't have to develop these methods, functions and components on their own. Instead they can just make a single call to the API to get their needed methods, functions or components at their own leisure. Another thing to note is that with a good API, with good

documentation; developers themselves can learn to make an API on their own. With this they can develop their own necessities if needed.

6 Discussion

The aim of this thesis was to create a custom API to ease the burden of GUI creation by having the developers make a call to the API. This started out with a literature study on the various topics related to the subject of React.js. Such as what language React.js utilized, what is React.js and finally what is an API and what's important to know when making one When it came to the development of the API various approaches were considered during the course of the thesis.

Two approaches were discussed when trying to implement the API. These were waterfall-style development and agile development. Each of these provides advantages and disadvantages to the style of API creation. In the end a hybrid was utilized giving the best of both worlds. That is having a clear cut plan and style of development and having a more loose and agile approach in the coding/development part. This provided some benefits such as having a structure in how the API is meant to look like while at the same allowing for a more free form and pair programming style approach that agile offers. Naturally, this comes at a cost, since neither waterfall-style and agile development gets fully utilized to their capability. An example of this that was disregarded would be the development of story cards that's common in agile development where one writes down to *who* the intended software is for. Another example from the waterfall-style approach is that only the starting nodes from it were utilized which are *Requirement, Design,* and *Implementation* (refer to appendix figure 10), the other two were disregarded since it clashed a bit with the agile style approach. The clash is due to how agile development keeps implementing after requirements that get added on after each iteration. This in turn makes it a bit difficult to evaluate and then finally maintaining the software as waterfall style wants one to do.

When it comes to the API itself the setup was as mentioned previously to have components and files separated into their own directory. The advantages of this method outweighs the disadvantages if one would just have it in one big directory. Since it gives the API structure and greater clarity on where each of the components and files are by the virtue of providing cohesive structure. The main disadvantage of this structured hierarchy is that one must specify the pathing to the specific file. However, this is just a small burden on the developers of the API rather than for the developers that want to use the API. Since it is an underlying structure that they most likely won't notice when they are working on their projects.

One of the key advantages of this API over traditional methodology as mentioned in results is that it reduces the time a developer needs to implement graphical components

inside of React.js. Since as mentioned previously the components themselves are individually reusable. This helps with both learning and utilizing the API. A such example would be in a company that develops web applications using React.js to build single page applications. With this API all they have to do is make a call to the individual components. Unlike the previous way where they have to specify first a CSS(Cascading Style Sheets), the graphical part of web development, file if they want to customize their style for the component. As well as coding the button itself as can be in appendix figure 7 and 8. With just the API all they have to do is specify what each of the components does, how it works and therefore go on from there. This makes the programming aspect easier, since they can readily and easily modify for example the button to suit their own needs both graphically and functionally. Another aspect of this API is for companies to learn and adapt their own styles, properties, and techniques which can be studied from this thesis and documentation, in order to have a standard protocol that employers can follow in the company.

6.1 Evaluation

The advantages of utilizing this API is that it reduces the time needed for developers to make their own React components by already providing existing components that they can utilize, and even modify to serve their needs, thus reducing the overall time the developers need to make GUI components while at the same time being extendable by allowing developers to modify the initial components however they see fit, while at the same time they can add their own components to the API. The main disadvantage of this API is that it does not improve upon performance such as speed of computation or response time, due to the API being focused on GUI components, this is something that the developers themselves have to address when they are utilizing the API by implementing their own methods.

6.2 Limitations

Due to the time limitations, it wasn’t possible to study every inch and corner of React.js. This is why the authors of this thesis decided to limit it to the improvement of GUI creation only. The reason for this is because React.js as mentioned previously is mostly just used for the creation of user interfaces for the front-end. Another factor to why it was something as simple as making a graphical API over something else, is due to the limited knowledge that the authors had of API creation at the start of the thesis. Therefore a lot of time was needed to learn the proper discipline, structure, and process that is API and the creation of API and making a complex and intricate API was not feasible.

Another limitation that has to be done was that certain features and functions couldn't properly be implemented. These were features that are database related. This is due to largely not being relevant to the thesis. But also that a database method/function is not possible to implement due to that database design is separate from each webpage and software. Thus, it is not feasible to implement a method that involves databases without breaking the intended software for the API even if it's a graphical one. This is especially apparent when there exists different approaches to databases such as relation databases like MySQL or non relation ones like NoSQL.

Another limitation that happened was the outbreak of COVID-19 during the process of thesis work, which put a hamper on the API creation. This is due to the fact that the authors live in different cities from each other and therefore did not want to risk the aversion of potentially infecting one and another. Therefore communication and work was mostly done via the assistance of online tools. This includes Discord for voice and screen sharing capabilities. So the authors could see what one and the other did, especially when it came to the coding part to emulate pair programming. For the documentation writing, instead of passing a document through Google Drive, Dropbox or something like GitHub the authors instead utilized Google Docs as their preferred writing tool. This provided the authors with a much needed assistance in writing since the document is shared online, and therefore is accessible at any given time no matter the location in the events that they couldn't meet up.

7 Conclusions and Future work

7.1 Conclusion

The purpose of this thesis was to implement a custom made graphical API that improves on GUI creation for React.js. As established in the preliminary, there exists no single best way for finding shortcomings. There exist several methods and depending on the context and what one is trying to improve one must look at the different dimensions as mentioned previously. With this, one must first develop a strategy on their own when it comes to tackling an API creation and think of first *what* one wants to improve and then work from there. After one has found what it is that needs improvement one must think of also *how* one is going to implement this. This process of itself requires a lot of design process that one needs to think about, especially since one can't just jump into implementation.

We also established methods that produce the outcome of creating an interface that is reliable in ways that could benefit the users when importing it to their projects. However limited it may be, this thesis and documentation could support the developer(s) in creating a custom API that can be used in companies and/or for personal use. Such methods include creating class components, which is done by using React.js' own Top-Level API, by importing React.Component framework that service in building a HTML element that has its properties transformed to one's liking, and another method is by implementing a JavaScript functionality that accepts props to access the changing implications. How this improves the standard of a framework is due to the fact that it makes things readily available for a developer. All they have to do is get the API and make a call to it and it will make the GUI components for them, without the need of having to manually make them. This in turns makes development of graphical interfaces easier and reduces the need of extra work on the developers part.

7.2 Future work

Work on other utilities besides improvement of GUI. There's much that one can learn from React.js and as such there exists many other fields and areas that one can strive to improve upon. One can improve upon aspects like security, performance, or even developing more GUI-based API since an API is an open-ended term that allows one to explore and study new possibilities for a project, software or even framework. Because of this open ended nature of what an API is. There will always exist work that one can

do to improve upon existing features, software and frameworks. Thus there is future work readily available. The key reasoning is to think of *what* it is that one is striving to improve upon.

8 References

[1] Doernhoefer Mark. *JavaScript. SIGSOFT Softw. Eng. Notes 31,* 4 (July 2006), 16–24. DOI:https://doi-org.ezproxy.hkr.se/10.1145/1142958.1142972 (Accessed 2020-03-18)

[2] React design documents: *Create a new React app* (Online source) 2018 (Last modified 2020-03-10) https://reactjs.org/docs/create-a-new-react-app.html (Accessed 2020-03-19)

[3] Frees Scott . *A place for node.js in the computer science curriculum*. J. Comput. Sci. Coll. 30, 3 (January 2015), 84–91. (Accessed on 2020-03-19)

[4] Marchand d.K Florent , Noyé Jacques, and Südholt Mario *Aspectizing JavaScript security*. In Proceedings of the 3rd workshop on Modularity in systems software (MISS '13). Association for Computing Machinery, New York, NY, USA, 2013. 7–12. DOI:https://doi-org.ezproxy.hkr.se/10.1145/2451613.2451616 (Accessed on 2020-03-18)

[5] Henning Michi. *API Design Matters*. Queue 5, 4 (May 2007), 24–36. DOI:https://doi-org.ezproxy.hkr.se/10.1145/1255421.1255422 (Accessed on 2020-03-20)

[6] J. Stylos, A. Faulring, Z. Yang and B. A. Myers, "*Improving API documentation using API usage information,"* IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Corvallis, OR, 2009, pp. 119-126.

[7] Dimanidis Anastasios, Chatzidimitriou Kyriakos C. , and Symeonidis Andreas L. *A Natural Language Driven Approach for Automated Web API Development: Gherkin2OAS*. In Companion Proceedings of the The Web Conference 2018 (WWW '18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2018 1869–1874. DOI:https://doi-org.ezproxy.hkr.se/10.1145/3184558.3191654 (Accessed on 2020-03-20)

[8] Alimadadi Saba, Sequeira Sheldon, Mesbah Ali, and Pattabiraman Karthik. *Understanding JavaScript event-based interactions*. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). Association for Computing Machinery, New York, NY, USA, 2014. 367–377. DOI:https://doi-

org.ezproxy.hkr.se/10.1145/2568225.2568268 (Accessed on 2020-03-28)

[9] React design document: *Getting Started* (Online source) 2018 (Last modified 2020-03-09) https://reactjs.org/docs/getting-started.html (Accessed on 2020-03-21)

[10] Facebook, React's github repository [Online] https://github.com/facebook/react (Accessed on 2020-03-21)

[11] Hunt Pete, O'Shannessy Paul, Smith Dave, Coatta Terry. *React: Facebook's functional turn on writing Javascript*. Commun. ACM 59, 12 (December 2016), 56–62. DOI:https://doi-org.ezproxy.hkr.se/10.1145/2980991 (Accessed on 2020-03-21)

[12] React. *Design Documents: Supported Browsers* [Online] 2018 (Last modified 2020-01-16) https://create-react-app.dev/docs/supported-browsers-features (Accessed on 2020-03-23)

[13] Xing Yong Kang, Huang Jia Peng, and Lai Yong Yao. *Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development*. In Proceedings of the 2019 11th International Conference on Computer and Automation Engineering (ICCAE 2019). Association for Computing Machinery, New York, NY, USA, 2019. 68–72. DOI:https://doi-org.ezproxy.hkr.se/10.1145/3313991.3314021 (Accessed on 2020-03-20)

[14] Gamez-Diaz Antonio, Fernandez Pablo, and Ruiz-Cortés Antonio. *Governify for APIs: SLA-driven ecosystem for API governance*. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019). Association for Computing Machinery, New York, NY, USA, 2019. 1120–1123. DOI:https://doi-org.ezproxy.hkr.se/10.1145/3338906.3341176 (Accessed on 2020-03-28)

[15] Bierhoff Kevin *API Protocol Compliance in Object-Oriented Software* Institute for Software Research School of Computer Science Carnegie Mellon University Pittsburgh 2009.

[16] Cusumano MA, Smith SA. Beyond the waterfall: *Software development at Microsoft*. 1995

[17] Gelperin David. *Exploring agile*. In Proceedings of the 2008 international workshop on *Scrutinizing agile practices or shoot-out at the agile corral* (APOS '08). Association for Computing Machinery, New York, NY, USA, 2008. 1–3. DOI:https://doi-org.ezproxy.hkr.se/10.1145/1370143.1370144 (Accessed on 2020-05-02)

[18] Aggarwal K. K., Singh Yogesh, and Kumar Jitender C. *Computing program weakness using module coupling*. SIGSOFT Softw. Eng. Notes 27, 1 (January 2002), 63–65. DOI:https://doi-org.ezproxy.hkr.se/10.1145/566493.566497

[19] Myers Brad A. and Stylos Jeffrey. *Improving API usabili*ty. Commun. ACM 59, 6 (May 2016), 62–69. DOI:https://doi-org.ezproxy.hkr.se/10.1145/2896587 (Accessed on 2020-04-16)

[20] Watson Robert. *Applying the Cognitive Dimensions of API Usability to Improve API Documentation Planning*. In Proceedings of the 32nd ACM International Conference on The Design of Communication CD-ROM (SIGDOC '14). Association for Computing Machinery, New York, NY, USA, Article 24, 2014. 1–2. DOI:https://doi-org.ezproxy.hkr.se/10.1145/2666216.2666239 (Accessed on 2020-04-16)

[21] Lutteroth Christof and Weber Gerald. *End-user GUI customization*. In Proceedings of the 9th ACM SIGCHI New Zealand Chapter's International Conference on Human-Computer Interaction: Design Centered HCI (CHINZ '08). Association for Computing Machinery, New York, NY, USA, 1–8. DOI:https://doi-org.ezproxy.hkr.se/10.1145/1496976.1496977 (Accessed on 2020-06-07)

[22] React. *Design document Introducing JSX* (Online source) 2017 (Last modified: 2020-03-09) https://reactjs.org/docs/introducing-jsx.html (Accessed on 2020-04-17)

[23] Fredriksson Johan, Tivoli Massimo, and Crnkovic Ivica. *A component-based development framework for supporting functional and non-functional analysis in control system design.* In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (ASE '05). Association for Computing Machinery, New York, NY, USA, 2005. 368–371. DOI:https://doi-org.ezproxy.hkr.se/10.1145/1101908.1101972 (Accessed on 2020-04-22)

9 Appendix

```
class CustomButton extends React.Component {

    static defaultProps = {
        color: 'blue',
        backgroundColor: 'lightblue',
        border: '1px solid black',
        borderRadius: '12px',
        width: '80%',
        padding: '10px',
        margin: '10px',

        type: 'button',
        onClick: handleClick()
    };
    render() {
        const { color, backgroundColor, border, borderRadius, width, padding, margin, type, children } = this.props;

        return (
            <button className={'customButton'} type={type}   style={{ color, backgroundColor, border, borderRadius, width, padding, margin }}
                {children}
            </button>
        )
    }
}
```

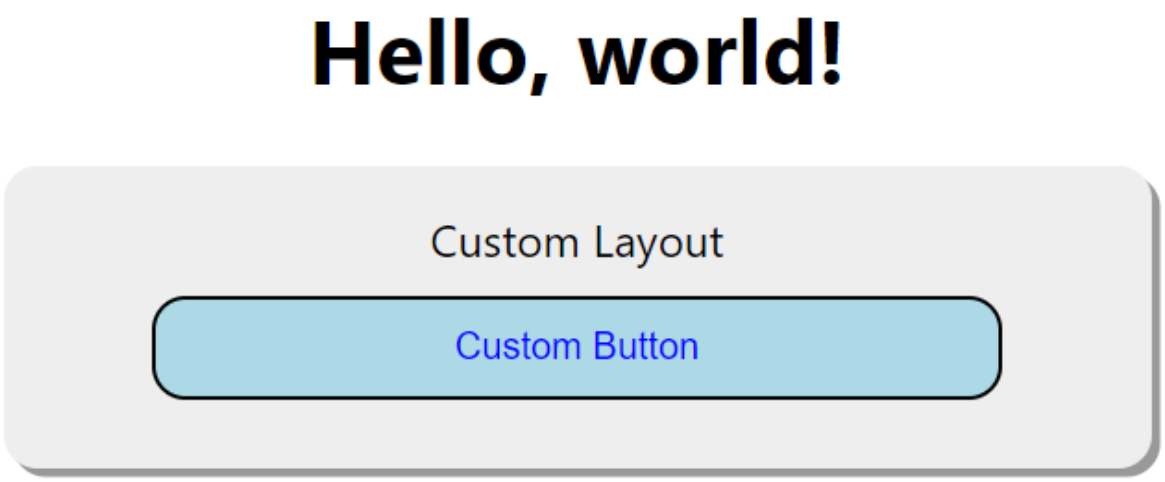
Appendix Figure 1: A custom button with its set properties using **defaultProps**, rendering a HTML element, **<button>**, to which it contains props arguments that handles the desired changes.

```
class CustomLayout extends React.Component {
    static defaultProps = {
        backgroundColor: '#eee',
        borderRadius: '12px',
        width: '30%',
        height: '100%',
        padding: '15px',
        margin: 'auto',
        boxShadow: '3px 3px #999'
    };

    render() {

        const { backgroundColor, borderRadius, width, height, padding, margin, boxShadow, children } = this.props;
        return (
            <div style={{backgroundColor, borderRadius, width, height, padding, boxShadow, margin}} >
                {children}
            </div>
        );
    }
}
```

Appendix Figure 2: A custom layout

```
function App() {
  return (
    <div className="App">
        <h1>Hello, world!</h1>
        <CustomLayout>Custom Layout
            <CustomButton>Custom Button</CustomButton>
        </CustomLayout>
    </div>
  );
}
```

Appendix Figure 3: Adding the custom components



Appendix Figure 4: Result of implementation

```
class CustomTextField extends React.Component {
    static defaultProps = {
        borderRadius: '5px',
        width: '45%'
    }

    render() {
    const {borderRadius, width} = this.props;
        return (
            <input style={{borderRadius, width}}/>
        );
    }
}
```

Appendix Figure 5: A custom textfield

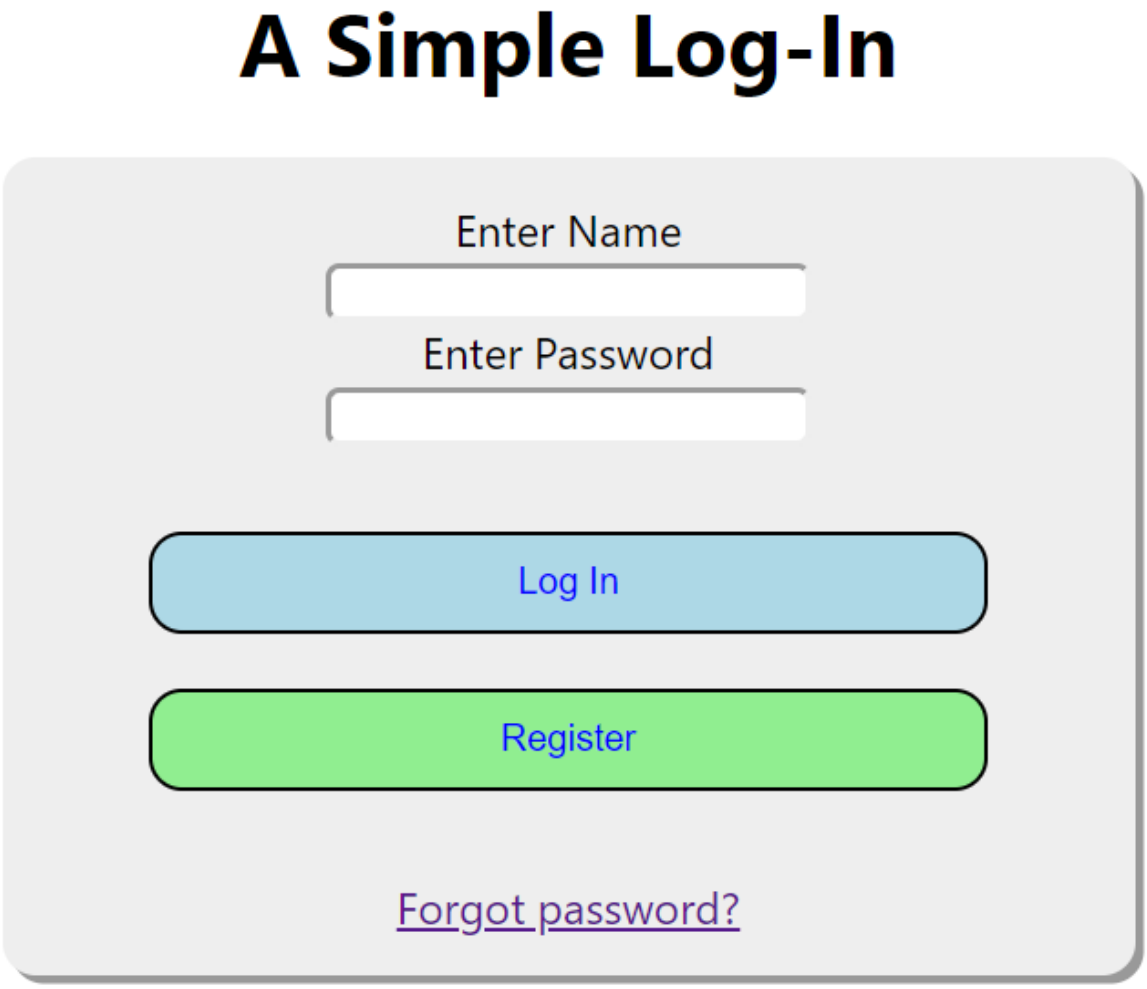
```
function App() {
  return (
    <div className="App">
        <h1>A Simple Log-In</h1>
        <div className={'customLayout'}>
            <label>Enter Name</label>
            <br/>
            <input className={'customTextField'} type={'text'}/>
            <br/>
            <label>Enter Password</label>
            <br/>
            <input className={'customTextField'} type={'text'}/>
            <br/>
            <br/>
            <button id={'loginButton'} type={'submit'}>Log In</button>
            <button id={'registerButton'} type={'submit'}>Register</button>
            <br />
            <a href={'https://en.wikipedia.org/wiki/Self-service_password_reset'}>Forgot password?</a>
        </div>
    </div>
  );
}
```

Appendix Figure 7: Creating the log-in page without the use of the API

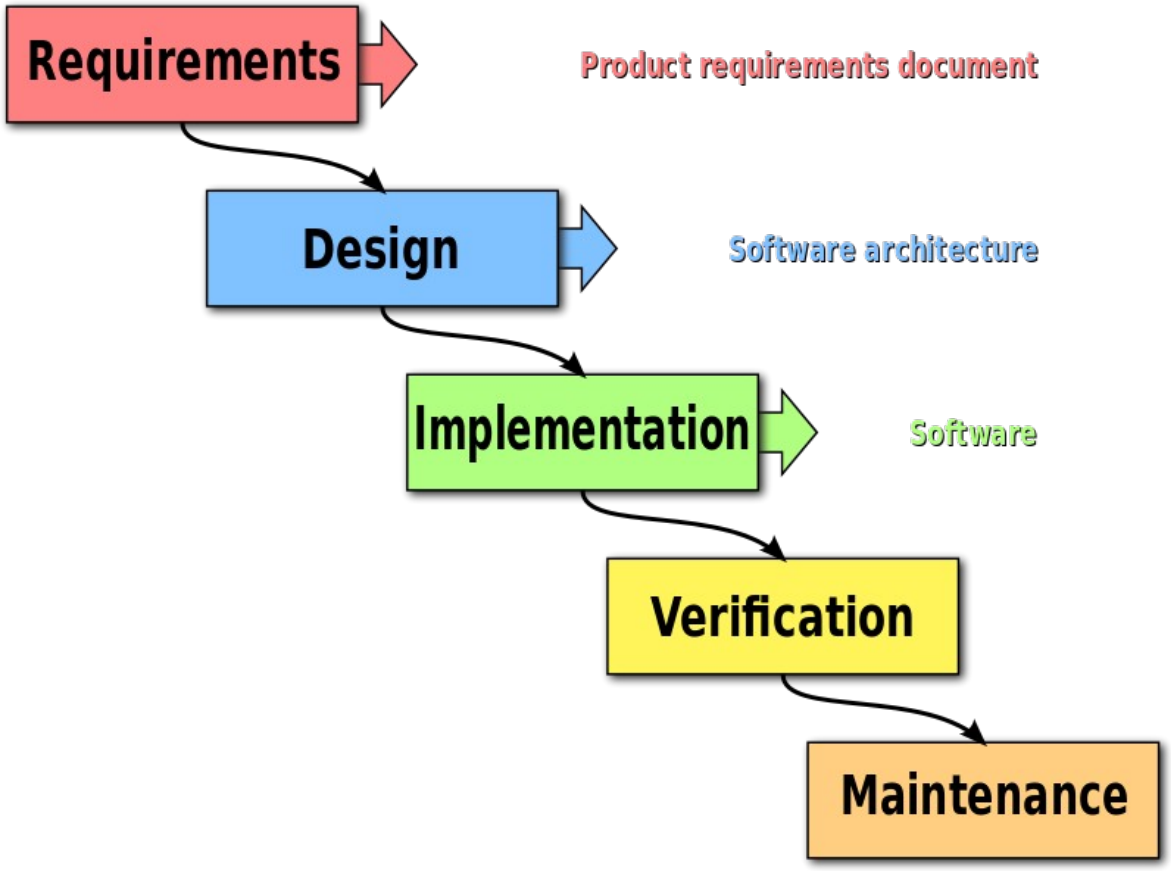
```css
.customLayout {
  background-color: #eee;
  border-radius: 12px;
  width: 30%;
  height: 100%;
  padding: 15px;
  margin: auto;
  box-shadow: 3px 3px #999;
}

.customTextField {
  border-radius: 5px;
  width: 45%;
}
```

```css
#loginButton {
  color: blue;
  background-color: lightblue;
  border: 1px solid black;
  border-radius: 12px;
  width: 80%;
  padding: 10px;
  margin: 10px;
}

#registerButton {
  color: blue;
  background-color: lightgreen;
  border: 1px solid black;
  border-radius: 12px;
  width: 80%;
  padding: 10px;
  margin: 10px;
}
```

Appendix Figure 8: CSS file of the application without the use of the API



Appendix Figure 8: View of the custom Log-In component.



Appendix Figure 9: Waterfall-style approach by Peter Kemp / Paul Smith - Adapted from Paul Smith's work at wikipedia, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=10633070