Tampere University

Hoang Khoa Nguyen

FRONT-END DEVELOPMENT OF AN EVENT-MANAGEMENT WEB APPLICATION: ADOPTING COMPONENT-BASED ARCHITECTURE WITH USABILITY HEURISTICS

Bachelor of Science Thesis
Faculty of Information Technology and Communication Sciences (ITC)
Supervisor: Prof. Zheying Zhang
April 2023

Abstract

Hoang Khoa Nguyen: Front-end development of an Event-management web application: adopting component-based architecture with usability heuristics
Bachelor of Science Thesis
Tampere University
Bachelor's Programme in Science and Engineering
April 2023

The fast-paced world nowadays might be challenging for university students to keep track of and attend their events. For that reason, this thesis focuses on front-end development with a user-centric approach of a web service aimed at helping university student to manage their personal and social events. The platform uses component-based architecture to build user interfaces while emphasizing designing great user interfaces and user experiences. This thesis will explore the principles of user experience design and its application to web development. It will also discuss different web architectures and technology stacks available to design and implement a friendly front-end. With its detailed development strategy, this thesis will greatly assist aspiring software developers aiming to create a user-friendly web application front-end.

Keywords: Web Development, Frontend Technology, Component-based architecture, User Experience design, Social Media, React.js

Preface

I would like to express my gratitude and appreciation to Prof. Zheying Zhang, for her support and valuable insights that helps me with writing this thesis. Also, I would like to thank An Nguyen - for his effort and hard-working attitude during the implementation process of this system, Anh Pham - for his contribution in the planning phase of our HTI.110 course. This project would not be possible without all the help and support I received.

Tampere, 24th April 2023

Hoang Khoa Nguyen

Contents

LIST OF SYMBOLS AND ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interfaces
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
JS	JavaScript
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VDOM	Virtual Document Object Model

LIST OF FIGURES

List of Programs

1 Introduction

In recent years, where the internet and online services have exploded in popularity, web applications have revolutionized how individuals and companies interact with technology. These applications are accessed via a web page and provide a wide range of services and functionality, making them accessible to any user with a web browser and internet connection. Users can use these system without the need to install software or go through complicated setup procedures, thus simplifying the process of using computer applications. These advantages make web-based apps applicable across various industries - primarily social media platforms, where people are connected and require fast access anywhere.

The user interface is critical to ensuring a positive user experience in any software product, mainly web-based services. Because the front-end is where users directly engage with the program, a responsive, intuitive, and appealing interface will ensure the service receives more traffic than a slow, unattractive one. As a result, while establishing a web application, organizations must invest resources in designing and developing an excellent front-end.

This thesis explores the development of a social media platform that encourages university students to participate in events organized by the school, with a specific focus on applying usability heuristics in front-end development. The thesis will discuss component-based architecture using Reactjs for implementing web application projects.

This thesis is structured as follows. Chapter 2 briefly introduces the background of component-based architecture, Reactjs, CSS libraries, and the strength of using them when building user interfaces. Chapter 3 discusses the importance of user experience in software applications and how to help users have a great time using software products. The design and implementation of the application prototype will be discussed in Chapter 4. Chapter 5 reflects the implementation results of the prototype and presents future development plans for the service. Finally, Chapter 6 will summarize the content of this thesis.

2 Background

Many system structures are available for a software development team to design and build a new application, each with its benefits and drawbacks. Thus, careful consideration must be taken when selecting an optimal method to establish an efficient infrastructure. These factors include end-user environment, website performance under load, or the device used to access the web page, which all significantly impacts the website's design and implementation process. Of all the available methods, the component-based approach has advantages to building a highly scalable system. This chapter will provide an overview of the component-based architecture for developing a web-based application, introduce some JavaScript Component-based frameworks, and explain how to utilize those frameworks with a CSS library.

2.1 Component-based architecture in software development

Because of its benefits, the component-based approach is one of the more popular software architectures nowadays. Component-based architecture can be considered in large-scale, complex systems where the top priorities are code reusability, scalability, and maintenance. With its design philosophy and implementation method, component-based architecture helps engineers and developers deliver highly scalable software products that are easy to test and maintain.

2.1.1 What is Component-based architecture?

In software engineering, Component-based development is building software by breaking down the application into smaller, reusable components. These components can then be combined to create a more extensive product [1]. By adopting this architecture, developers can reuse particular components of the system. [2].

Component-based web architecture operates on the concept of components - self-contained units comprising logic, styling, and markup. An example of a component-based system is depicted in Figure 2.1. A component in this system implements one or more interfaces and satisfies specific contract duties. A component-based system consists of several components, each of which serves a distinct purpose in the system and is described by an interface. The component model is a collection of component types, their interfaces, and a description of the permissible interaction patterns between component types. The component framework includes various deployment

and runtime services to support the component paradigm [3]. Developers can extend a component without changing the existing internal code or modifying existing parts by combining different component interfaces with the contracts they satisfy, creating a new component model with different functionalities.

This architecture makes it easier to manage, maintain, and evolve a more complex system [4], as each component can be updated and modified independently of the rest of the application, and adding or removing different components generally has minimal impact on the structure of the product. Compared to a traditional monolithic system - in which the functionality of an application is built into a single, large code base - a modular approach such as component-based architecture could potentially reduce the complexity and vulnerability of such implementation [5] [6].

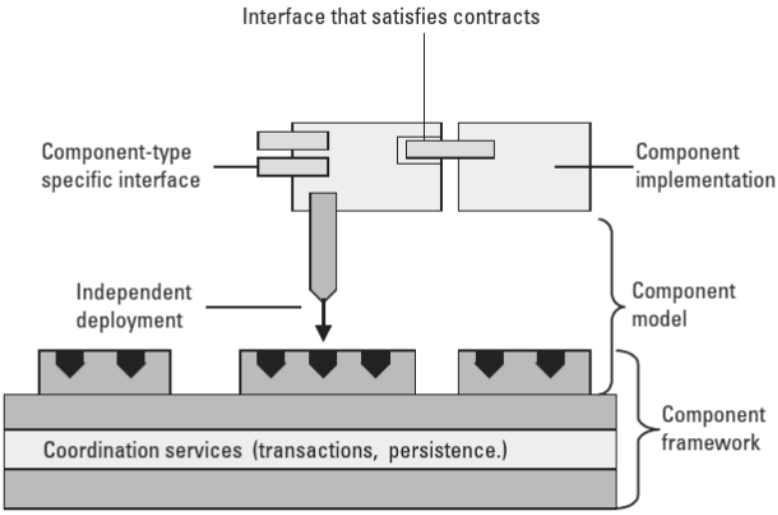

Figure 2.1 *Example of a component-based system. [3]*

In conclusion, component-based web architecture follows the foundation principles of modular design, which develop products by integrating independent modules. Each component has a well-defined interface that produces its inputs and outputs, making using the same component in multiple places possible. This practice results in a more structured and organized code base that is easier to maintain over time [4]. To build the application, the developer can freely choose and connect multiple components to create a combination that serves the purpose of the software.

2.1.2 The benefits and issues of using Component-based architecture in software development

Using the component-based architecture to build a software product has multiple advantages. These benefits come from the implementation method of the system: defining different components and connecting them like building bricks. For this

reason, it is easier and faster to build functionalities for the system by combining multiple parts. The Component-based architecture has many profits [6], including:

- **Improved code reusability**. The design of each component aims for modularity and self-containment, enabling it to operate simultaneously in various parts of an application or even in multiple applications. Without the need for modification of each use, this can aid development time and lower maintenance costs.
- **Extensibility**. Different components can be combined to create new behavior. They can also extend the behavior of another component while keeping its own, creating a different interface for the software to use.
- **Encapsulation**. Each component is completed on its own and is self-contained. They only expose their functionality through the interface while enclosing the internal data manipulation process.
- **Independence**. Components have minimal dependencies on other components and can operate based on the input data in different environments and contexts.
- **Better scalability**. A component can be freely added or removed from a system to scale the application up or down as needed without impacting other parts of the application.
- **Easier maintenance**. Because self-contained components can be easily tested and maintained without impacting the rest of the system, component-based architecture can lead to faster bug fixes and easier updates.

Although component-based architecture offers many advantages, it also has potential drawbacks. These flaws also came from the design principles of the architecture itself and must also be considered before deciding to use this design approach. These issues including [6] [7]:

- **Complexity**. While it is true that component-based architecture makes significant and complex systems easier to manage, dividing the infrastructure into too many pieces can backfire. Managing many components can increase complexity within software systems that use a component-based architecture, making the system harder to understand, maintain, and modify.
- **Overhead**. Component-based architecture may introduce additional overhead regarding performance and memory usage. These overheads result from each component type having its runtime environment and may need to communicate with other components via message passing or other mechanisms.

- **Dependency management**. Components may depend on other components or external libraries, leading to versioning issues and other complexities, making it challenging to manage the dependencies between components and ensure they are all compatible.
- **Integration challenges**. Integrating components from different vendors or teams may prove challenging, primarily when differing assumptions or interfaces regarding their usage exist. With proper communication and documentation, combining different parts implemented by different teams could be more accessible.
- **Testing**. While it is easier to provide unit tests for individual components, the whole component-based system can be challenging to test due to its complexity and the potential for interactions between components. This complication makes it difficult to ensure that the system works correctly and that all components function as expected.

Component-based web architecture is a powerful method of building web applications that provide many benefits over traditional, monolithic approaches. By breaking down the user interface into smaller, reusable components, developers can build more scalable, maintainable, and modular web applications that are easier to manage and modify over time. However, like any other technology, the component-based approach to software engineering also has its issues along with the benefits it provides. With different projects varying in scale and purposes, understanding potential problems and deciding whether their benefits can overcome the drawbacks is essential.

2.2 JavaScript frameworks/library in a nutshell

In order to understand the adoption of different JavaScript (JS) frameworks and libraries in the modern web, it is necessary to learn the primary method and elements of web development. This section will explain the traditional approach to building a website, the issues of that method, and the creation of different JS frameworks and libraries to address those problems.

2.2.1 The basic elements of web development

In traditional web development, developers can combine Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS), and ”vanilla” JS - the plain, unaltered version of JS - to create web pages and applications. HTML provides the structure

and layout of a web page, while CSS is used to style and format the page [8] [9]. Vanilla JS adds dynamic functionality to a web page, such as interface updates, form validation, and interactive elements.

A universal language that all computers may theoretically understand is required to distribute knowledge globally - thus, HTML is created as the publishing language used by the World Wide Web. It enables the authors to publish documents with different content elements, e.g., headings, texts, tables, and lists, or retrieve online information with a button click through hypertext links [8]. Developers can mark up their works with structural, presentational, and semantic information. HTML is one type of markup language used to provide content for a website, and it uses a tag system to display its content, with an example shown in Program .1.

In combination with HTML, web authors can use CSS to style their content. CSS provides far richer document looks than HTML ever did, even at its peak presentational extent [9]. Using CSS, developers can freely alter the styling of otherwise plain HTML components, including text, background, heading, and border, by changing their color, capitalization, decoration, spacing, or display [9]. For example, to change the styling of the heading of Program .1, the CSS code can be used as shown in Program .2. CSS opens up new possibilities for developers to decorate their web designs.

JS, or "language of the web browser", is one of the most popular programming languages in the world [10]. It controls the behaviors and adds dynamic functionalities to HTML, allowing developers to create interactive web experiences. With JS, web authors can access and control the Document Object Model (DOM) - an API for HTML documents. The DOM can be used to construct, explore the hierarchy, and edit the content of a web page by adding, removing, or changing nodes and content inside the DOM tree [11].

Moreover, developers can use JS to handle different web events customers perform on the site, or make Asynchronous JavaScript And XML (AJAX) requests. Website events can be user interactions, e.g., button click, menu toggle, hovering, or from the web page itself, e.g., page loading or errors. Events are the glue that holds the user and the Web page together; they allow pages to become interactive and responsive to what the user is doing [12]. AJAX is a technique used in web development that allows data retrieval from a server asynchronously without requiring a page refresh. By exchanging small amounts of data behind the scene with the server, AJAX eliminates the need to reload the entire web page whenever the user requests a change. By doing this, AJAX can help websites improve their interactivity, speed, and usability [13].

Developers can use a combination of these languages and techniques (HTML, CSS, "vanilla" JS, Event handling, and AJAX) to create a web application without any additional libraries or frameworks. While this web development method is still widely used, it can take time and effort to build an interactive web page using this approach. For example, manipulating the DOM to update the web interface using vanilla JS can be tedious and time-consuming because the DOM requires a lot of code to navigate and update. This task can be tedious because of the complex nested structure of the DOM and how it is poorly specified [11]. Moreover, making AJAX requests using vanilla JS can be complicated, as it requires a considerable amount of boilerplate code and is confusing to use by default [12].

2.2.2 The introduction of Javascript frameworks and libraries

There are JS libraries and frameworks to address these challenges. By adopting a JS library, developers use pre-defined code to build the functionality and aesthetic of a web application. Developers often classify libraries by their functionality, or they may be utilized to build application skeletons - often known as frameworks - that help build websites more efficiently. JS frameworks contain utilities, functions, and high-level abstractions tested across several platforms and browsers. In practice, JS frameworks are used for various tasks, including graphic design, graphing and dashboards, animation, and event management [14]. They give a more intuitive and straightforward interface for working with the DOM and making AJAX requests, saving developers time and simplifying the development process.

One of the crucial advantages of adopting a JS framework is that it can save developers significant time. Developers can adopt pre-existing, well-tested, and optimized components instead of writing code from scratch to achieve a specific task, allowing them to focus on the unique characteristics of their project rather than writing boilerplate code. Another benefit of using a JS library is that it can help enhance a project's overall performance and maintainability. JS frameworks and libraries also provide built-in debugging and testing tools, making it easier to identify and fix issues or add new features to a project over time.

Many different JS frameworks and libraries are available, each with its features and capabilities. For this thesis's scope - the front-end development of a web application - only front-end frameworks and libraries are discussed. Therefore, present below are some of the most wanted front-end frameworks and libraries, based on the 2022 Developer Survey by StackOverflow [15]:

- **React**. A JavaScript library for building user interfaces [16]. It is developed and maintained by Facebook and is widely used for building complex web applications.

- **Vue**. A JavaScript framework for building user interfaces [17]. It is known for its simplicity and flexibility and is often used for building small to medium-sized web applications.
- **Angular**. A JavaScript framework for building web applications [18]. It was developed and maintained by Google and is known for its powerful two-way data binding and dependency injection features.

React is the most popular front-end JS library [15], followed by Angular and Vue. They are powerful tools that help developers create robust and maintainable web projects more efficiently. With many frameworks and libraries available, each with advantages and disadvantages, choosing the right one for a project will depend on the specific requirements and use case.

The popularity of React is due to a combination of its features and the developer community that surrounds it. As mentioned above, the traditional approach to building a website is using a combination of HTML, CSS, and vanilla JS for event handling and DOM manipulation. This approach of directly manipulating the DOM is inefficient because the page's content is constantly changed, and the CSS related to those content must also be updated. Thus, updating the website using vanilla JS through the DOM is complicated and tedious. Furthermore, when the DOM is updated, the whole page is re-render, resulting in unnecessary re-rendering and compromising performance.

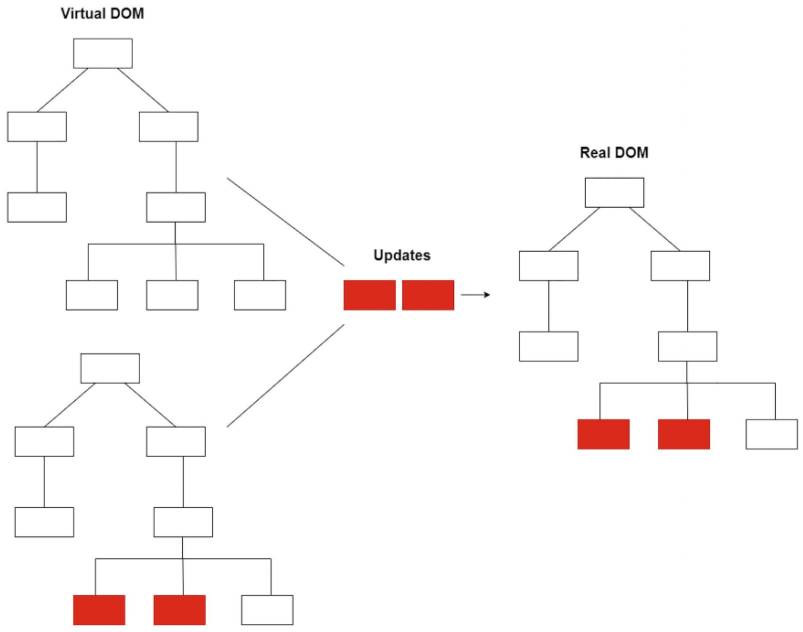

Figure 2.2 *React virtual DOM only re-renders the UI component whose data changes. [19]*

Resolving these disadvantages is where React's virtual DOM (VDOM) shines. The VDOM is a lightweight "virtual" representation of the actual DOM, stored in the

browser memory and synced with the ”real” DOM. To improve the performance of updates, the VDOM calculates and only re-renders parts of the UI that need updating, rather than re-rendering the entire page [20], allowing React to efficiently update only the necessary parts of a user interface, as shown in Figure 2.2. This selective re-rendering is particularly useful for building large, complex web applications with many dynamic elements that constantly change when the users interact with the web page.

Another reason why React is popular is its flexibility. React is a library, not a framework, which means it does not dictate how an application should be structured; it just provides a way to build reusable UI components [21]. This modularity allows developers to use their preferred approach and architecture for building web applications. React is also relatively small and easy to learn, enabling developers to build applications quickly [21]. For this reason, React can be used in various architectures, but the most common implementation is component-based. Through React, developers can quickly build a system of interface components and combine them to create a complex system with incredible customizability and functionality, as discussed in Section 2.1.

With a large and active community of developers, React has many resources available for learning and troubleshooting. React was developed and maintained by Facebook, and many big companies, such as Facebook and Netflix [22], widely use it. This diversity has led to a large ecosystem of third-party libraries and tools that integrate seamlessly with React, such as Next.js, Gatsby, and Create React App. Using these tools that have been tested and verified, developers can invest more of their time and effort into building and implementing functionalities and less into testing debugging.

React is currently the most popular JS library due to its ability to handle complex user interfaces, flexibility, and large and active community of developers. While other popular JS frameworks and libraries have advantages, the VDOM and flexibility of React make it a powerful tool for building reusable and efficient user interfaces. Additionally, the large ecosystem of third-party libraries and tools that integrate seamlessly with React makes it a versatile and easy-to-use option for web development.

2.3 CSS framework

When building the front-end of a web application, after developing the page structure with HTML comes CSS, which can be used to determine the appearance and layout of the user interface. Nowadays, there are multiple methods of writing CSS, ranging from using ”vanilla CSS” - or CSS in its purest form, without any preprocessor or libraries - to using CSS frameworks and libraries. Vanilla CSS is lightweight, fast,

and flexible since there is no additional overhead from a preprocessor or framework. The most significant advantage of writing CSS from scratch is that developers can have complete control of the styling of the webpage [9] and implement any design they want to - as long as they can ”translate” their design to CSS. Web engineers can write and manage CSS code without relying on a third-party library, which leads to cleaner and more efficient code - given that they are competent with the language.

This complete control over the decoration of the page is also where vanilla CSS has its issues. As the application grows to a bigger scale, styling the interface using only vanilla CSS can be challenging [23], as the code can become repetitive and lack modularity for different components. Maintaining a consistent look across the web application is more complicated with vanilla CSS, especially when dealing with frequently changing, dynamic components. Moreover, vanilla CSS has limited functionality and a steep learning curve, making it harder for beginners to use the language efficiently. Missing some advanced features such as variables and functions, vanilla CSS makes it more challenging for developers to style and design dynamic components and conditionally render.

In order to overcome the drawbacks of using vanilla CSS, multiple CSS libraries and frameworks were born. In recent years, using CSS frameworks has become increasingly popular to streamline the development process and improve the overall quality of web applications. CSS frameworks are collections of pre-written CSS code that can add specific functionality and design elements to a web project. These frameworks provide pre-defined classes and styles that can be easily added to HTML elements, allowing developers to quickly and easily apply consistent styles to their web pages instead of spending time writing CSS code from scratch [24].

One of the main benefits of using a CSS framework is that it can help improve a web project's overall design and consistency. Many CSS frameworks include pre-defined classes and styles that can be used to create visually appealing and consistent layouts and design elements. An example of this can be seen in Figure 2.3, which shows different pre-built Header components in Bootstrap. These pre-defined elements can be beneficial for developers who are not experienced in design or who are working on a project with a tight deadline. Using a pre-defined framework also ensures that the resulting code is more consistent, reliable, and maintainable [25].

Moreover, CSS frameworks can also play a key role by providing a distinctive visual style across the application. A CSS framework can help keep the design uniform across different pages and devices, ensuring the layout and typography align with pre-defined rules. This consistency helps to create a sense of coherence and brand identity, which can positively impact overall user satisfaction.

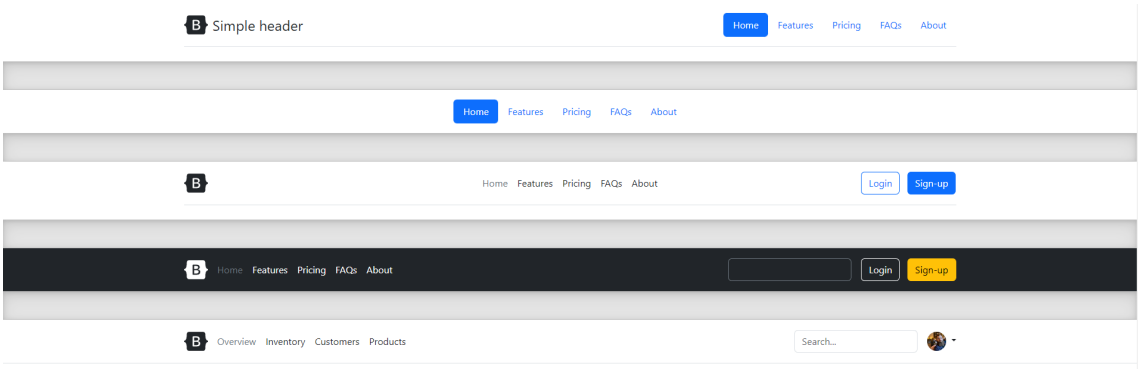

Figure 2.3 *Bootstrap pre-built Header examples [26].*

Another critical advantage of utilizing a CSS framework in the development process is the ability to implement responsive design more easily [24], which adjusts the user interface layout based on the size and orientation of the device. Responsiveness is critical for delivering a good user experience on various devices, from desktop computers to smartphones. Using a CSS library also makes it easier to implement accessible designs considering the needs of users with disabilities. Accessibility can include high-contrast color schemes, alternative text descriptions, and keyboard-friendly navigation.

Another benefit of using a CSS framework is that it can help improve a web project's performance and maintainability [25]. Many CSS frameworks include built-in optimization techniques, such as minification and compression, that can help reduce the CSS code's file size and improve the loading time of a web page. CSS frameworks also make identifying and resolving code issues easier by providing debugging and testing tools. With their pre-defined, reusable classes and design patterns, CSS frameworks can also help decrease code redundancy and improve code maintainability. Lastly, with large communities of developers and online support, it is easier for new developers to learn and use CSS frameworks effectively.

Many different CSS frameworks are available, each with its features and capabilities. Some popular CSS frameworks and libraries include [27]:

- **Bootstrap**. Developed and maintained by Twitter, Bootstrap is one of the most widely used CSS libraries. As a front-end toolkit, it provides a set of pre-defined classes and styles that can be used to create responsively and mobile-friendly layouts [26].
- **Foundation**. Developed by ZURB, this is another popular CSS library that provides a set of pre-defined classes and styles that is accessible and works with different platforms. Foundation makes it simple to create responsive websites and applications that look fantastic on any screen. Foundation is fully customizable, readable, adaptable, and semantic [28].

- **Bulma**. Bulma is a lightweight CSS framework with pre-defined classes and styles for creating responsive and modern-looking layouts. With its simple design system, straightforward syntax, JS-free, and quick customization, Bulma makes it easy for developers to create a functional design quickly [29].
- **Tailwind**. Tailwind is a utility-first CSS library that provides a set of low-level utility classes that can be composed to create complex designs. It provides pre-defined CSS classes that work straight from HTML without developers defining any additional CSS files [30].

CSS frameworks are powerful and can help developers greatly enhance their styling workflow efficiency while ensuring their design can work on multiple platforms, keeping the accessibility of different user groups in mind. With the wide selection of CSS frameworks and libraries available, each with its own set of benefits and drawbacks, choosing the one to use in a project or not using a CSS framework at all will rely on the particular needs and use case of the application.

2.4 Implementation with Chakra UI

Chakra UI is an open-source CSS library designed to help designers and developers create intuitive and accessible user interfaces [31]. It provides a comprehensive set of reusable and customizable components that can be easily integrated into a web-based application. The library is built on React; hence it was designed to work seamlessly with React components [31].

One of the critical strengths of Chakra UI is its focus on accessibility [31]. The library provides components that are fully accessible, meaning that they can be used by people with disabilities, such as users who are blind or have low vision. The library also supports screen readers, keyboard navigation, and other accessibility features. With these features, developers can easily design a web page accessible to anyone across multiple platforms and usable on different screen resolutions.

In terms of its uses, Chakra UI can be used to enhance the UX of a wide range of web-based applications. The library provides a pre-made styled system consisting of different components for various UI elements, such as buttons, forms, icons, and models. The components are designed to be highly customizable, allowing designers and developers to create interfaces tailored to their specific needs. In the case of this project, Chakra UI makes it easy to create a universal styling system of different UI components, which can then be applied to specific uses. This styling system will be discussed in more detail in Chapter 4.

Chakra UI is also designed to be lightweight and fast, making it ideal for applications that require high performance and low latency [31]. The library is optimized for performance and provides fast and smooth animations, transitions, and interactions. This powerful library provides designers and developers with the tools they need to create user-friendly and accessible web-based applications. With its focus on accessibility, customizable components, and high performance, Chakra UI is an excellent choice for designers and developers who want to enhance the UX of their applications.

3 User Experience in Web applications

User Experience (UX) is one of the most important aspects of modern software design. With good UX, customers are willing to spend time using the product, making it successful. According to Justin Mifsud, the founder of Usability Geek, a website with a bad experience will have 88% of its customers less likely to return [32]. This chapter will explain UX in detail - specifically in software and interface design - with its importance on the front-end. This chapter will also present the methodology for implementing great UX on the front-end.

3.1 What is User Experience?

UX is a critical part of web-based applications, as it determines how users perceive and interact with the application. In essence, UX includes all aspects - the needs, interactions, feelings, thoughts, and satisfaction - of the user engaging in some activity [33] [34]. In software engineering, UX is a broad term, and it can cover anything, from how well is the navigation of the service, how the content is provided, how responsive the interface is, or how ”good” the control of the software feels. The goal of UX design is to create an intuitive, usable, and satisfying interface for users [34].

The importance of UX in web-based applications is rooted in the changing nature of the digital landscape. Today, users are exposed to a wide range of digital experiences, and their expectations for high-quality products have risen accordingly. In this context, UX design has become a critical differentiator, as it can significantly impact the success or failure of a web app. Great UX can prolong the loyalty of users, and they will want to keep using the products. In response, the UX design field has evolved to aid the increasing demands for high-quality digital experiences from users [34].

The critical components of UX in web-based applications include usability, accessibility, and aesthetics. Usability refers to how easy the product is to use and how well it supports the goals and tasks of users [35]. Good usability means that the product is responsive and can perform according to the needs and commands of users. Accessibility refers to the ability of the application to accommodate users with disabilities and to provide equal access to all users [36]. Aesthetics refers to the visual design of the application and how well it supports the brand identity and the overall user experience.

3.2 The importance of UX on the front-end

As a result, the front-end of a web-based application plays a crucial role in determining the overall quality of the UX. Good UX is essential for the success of a web service, as it can significantly impact user engagement and satisfaction [34].

When a web app has poor UX, the consequences can be significant. Users may become frustrated, confused, or overwhelmed by the interface and abandon the application altogether. Poor UX can also negatively impact the reputation of the application and the brand, as users are unlikely to recommend an product that does not meet their expectations.

On the other hand, when a web service has great UX, the benefits are numerous. Users are more likely to be engaged and satisfied with the application and more likely to become loyal users. Great UX can also lead to increased user adoption, as users are likelier to recommend the product to others.

To achieve great UX in the front-end of a web app, designers must carefully consider the needs and goals of users and design interfaces that are intuitive, efficient, and aesthetically pleasing. Constructing a software product with a cohesive and distinctive aesthetic will help the service stand out from the crowd, which can impress the users and engage them to use the product. Developers must also take into account factors such as accessibility and usability, as these can have a significant impact on the overall UX. By designing software with great accessibility, the audience range of the application can be broadened drastically, for instance, to users that are color blind or have to use a screen reader.

3.3 How to implement good UX design on the front-end

To implement good UI and UX in the front-end of a web-based application, developers must consider several key factors and best practices. One of the guidelines for designing user experience is the 10 Usability Heuristics by Jakob Nielsen [37]. They are principles that designers can follow to develop intuitive UI, and specific usability rules do not confine them. Instead, usability heuristics are general guidelines developers can follow to help create accessible, efficient interfaces. The ten principles of Usability Heuristics are defined as follows:

1. **Visibility of system status**. By providing customers with immediate responses to their actions, the service can always keep users updated on what is happening. These system notifiers can be achieved by using multiple feedback

forms: pop-up notifications, indicators, and status messages when the users interact with different UI elements of the product.

2. **Match between the system and the real world**. The design should communicate in a language that users can easily understand by utilizing words, phrases, and concepts the user knows, arranging information naturally and logically. Avoid internal jargon and ensure that the users can understand and control the application naturally, e.g., use natural mapping and stimulus-response in the control design so that the user can understand the interface naturally [38].

3. **User control and freedom**. Users require a designated "emergency exit" to leave unpleasant actions without going through a lengthy procedure because they frequently make mistakes when doing actions. The design can assist users with an obvious way to exit the current interaction, such as a "Cancel" button - be sure that the exit is well-labeled and easily accessible.

4. **Consistency and standards**. Adhere to platform and industry conventions to prevent users from guessing whether various words, contexts, or actions indicate the same thing. According to Jakob's Law, people spend most of their time on other websites [39], which indicates that most users expect your site to function similarly to other familiar sites. Users' cognitive load can increase if they are forced to learn something new, different from their expectations, causing them to overthink and stress out.

5. **Error prevention**. Even though effective error messages are vital, it is considered best practice to avoid problems in the first place. Potential errors can be negated by eliminating error-prone situations, checking for user inputs, and providing users with a confirmation before committing to an action. Prevent errors by sanitizing user inputs, providing practical constraints and placeholders, and warning users.

6. **Recognition rather than recall**. Visible items, actions, and options can help reduce the user's memory load - they should not have to recall information from one interface section to the next. Assist users in identifying information in the interface rather than pushing them to memorize it by making field labels or menu items visible and easily accessible.

7. **Flexibility and efficiency of use**. By enable shortcuts - hidden from novice users - the design can cater to both naïve and experienced users. Enable users to customize frequently performed tasks and speed up the interaction. Provide personalization and customization so individual users can freely tailor their content and functionality.

8. **Aesthetic and minimalist design**. Exclude irrelevant or infrequently used

information from interfaces because they can compete with the relevant information, lowering their relative visibility. Focus the content and visual design on the fundamentals. Let no unnecessary items distract consumers from the information they require. Prioritize the information and features that will help users achieve their primary goals.

9. **Help users recognize, diagnose, and recover from errors**. Use straightforward, easy-to-understand language to display error messages and constructively offer a remedy to the problem. Employ standard error message aesthetics - bold, red letters - while avoiding using technical jargon when explaining what went wrong to users. Provide users with a solution - such as a shortcut - to solve the mistake quickly.

10. **Help and documentation**. The design should function in a way that does not need further explanation. However, if documentation is required to assist users in understanding how to fulfill their jobs, ensure the help documentation is easy to find. Make sure the documentation is available when the user needs it, and make a list of concrete steps to take.

By utilizing the 10 Usability Heuristics, the UX design of this platform will be discussed in more detail in Section 4.4 of this thesis. With excellent UX design, a distinctive brand identity, and aesthetics, the application can ”feel” good to use and will ensure the success and traffic of the product.

4 Implementation

The main scope of this thesis is to build a working prototype of a web application - Event Go - that focuses on encouraging university students to attend different social events. This chapter will provide the completed process of building the front-end of the platform from start to finish. Section 4.1 will discuss the original idea of building this application and the scope and goal of the project. The structure of the front-end setup will be explained in Section 4.2 and 4.3, followed by the UX design methodology and usability assurance on the front-end in Section 4.4. Then, Section 4.5 will cover the communication and data persistence between the front and back-end of the application. Lastly, the Section 4.6 covers implementing the styling system and interface components to build the completed system.

4.1 Scope, goal, and early stages of the project

The idea for this application started as the final project for a *Human-Technology Interaction* course at Tampere University - the *HTI.110 course, 2021 implementation* [40]. The final project of this course requires a basic sketch for a web-based service that aims to assist university students in managing different events. This web service functions similarly to a social media, where every student can create an account, become friends with other students, check out event recommendations from the platform, and view events to attend them.

The goal of this project is mainly for the development team to learn and apply their knowledge in software engineering to a ”real world” project and build a working web application. By developing a working prototype, the team can gather valuable knowledge not taught directly in university. The team hopes that this application can be further expanded in collaboration with Tampere University to make it a part of the digital system of the school. By doing so, the university can have its event managing social media site, where different events can be promoted and have students attending them, without having to manage multiple social media pages for event promotion, from Instagram to Facebook.

The initial ideas of the project started with pen-and-paper sketches. The sketches were done to define the basic functionalities of the pages and map out different essential interface elements and the interactions between users and the application. The pen-and-paper sketches were made so that everyone in the development team could freely present their ideas and discuss with everyone what approach should be

chosen to implement functionality, as shown in Figure 4.1. The team defines the system's application flow, navigation, and functionalities in these early pen-and-paper sketches. These sketches provide basic ideas of how the service works and user interactions.

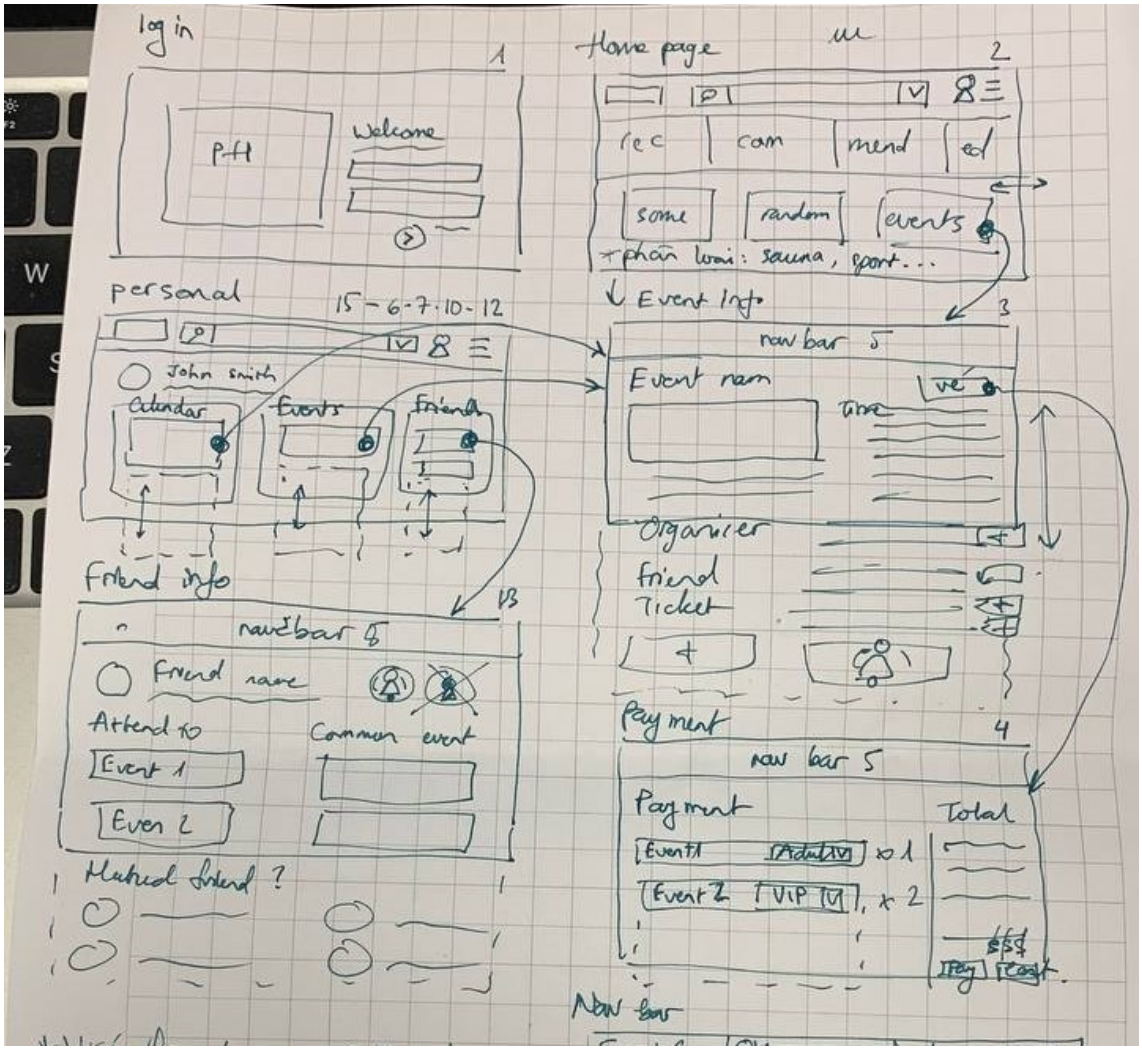

Figure 4.1 *Early pen-and-paper sketches of the web service.*

The team had three interviews with some Tampere University students to define the focus of end users in this kind of event-promoting social media platform. By gathering the interview results and analyzing them, the team can finalize their product plans and move to the next stage of development: defining the interactions and building prototypes.

The development team made the first wireframes of the application using Figma - a tool for teams to design and prototype product interface [41]. Figma was chosen because it is free to use, and the team can freely collaborate through a shared design board and design together in real time. Through this iterative process, the team carefully considered different elements in the design while following the 10 Usability Heuristics for designing interface [37]. By following this set of defined principles, the development team can ensure that each interface element is usable, allowing for

a smooth and intuitive UX.

Using the prototyping function of Figma, several low-fidelity wireframes of the service were made. These low-fidelity sketches act as a skeleton for the product that can be interacted with, thus setting up a base for further interface development to be built upon. Figure 4.2 shows the early stage of defining these low-fidelity prototypes. These low-fidelity mock-ups were based on the pen-and-paper sketches in Figure 4.1 and the interview results. They are easy and inexpensive to create, representing the application flow and page navigation, allowing developers to quickly iterate through different design options before committing to a final design.

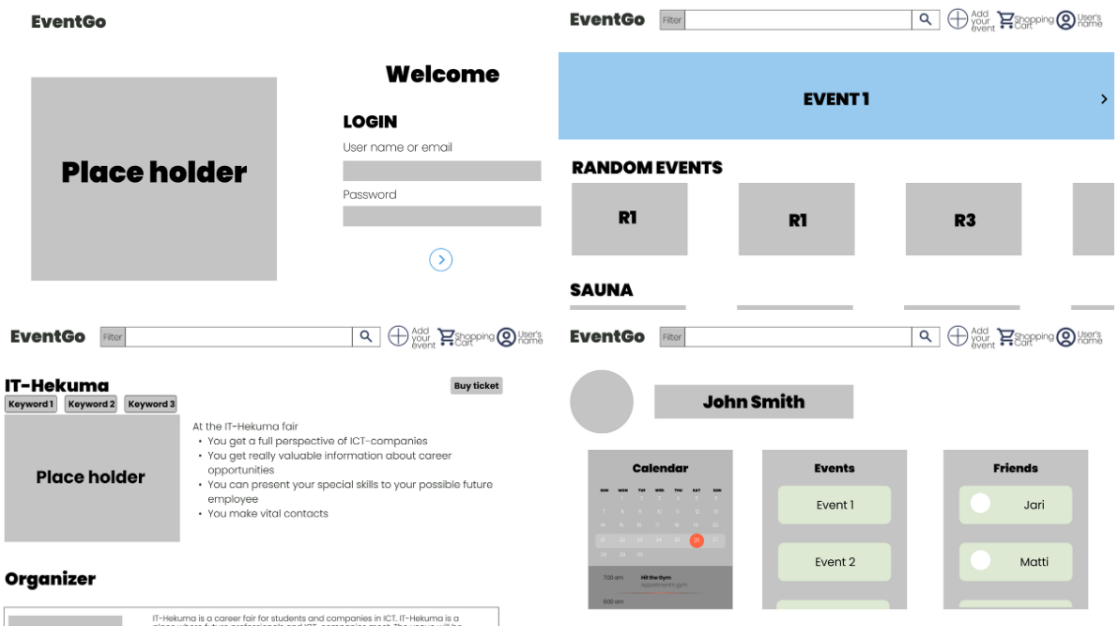

Figure 4.2 *Early low-fidelity sketches of the web service. [42]*

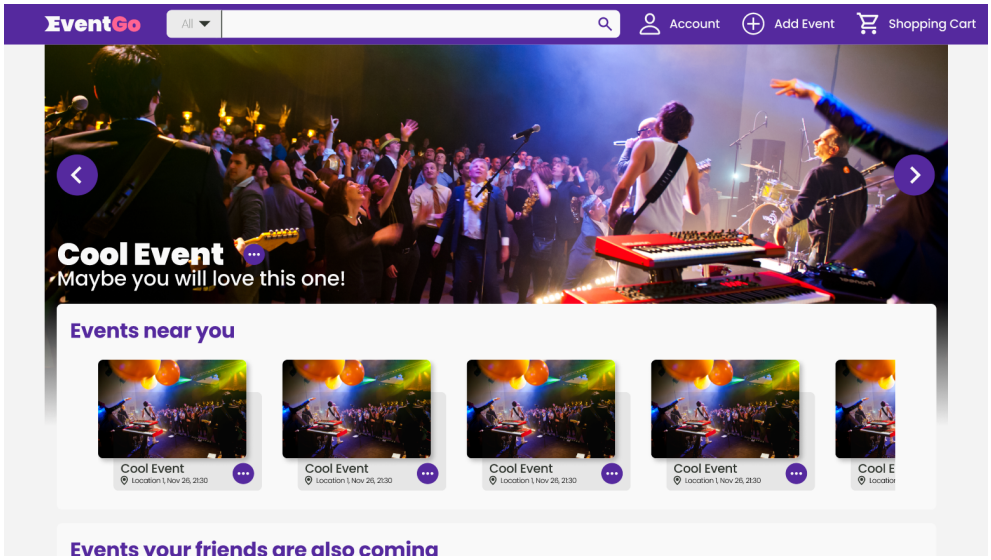

Figure 4.3 *Example of a high-fidelity page prototype of the web service. [42]*

After completing the low-fidelity sketches, the team tested the designs through a peer-review process. The testing procedures include usability, application flow, and

interface tests. Then, the group started to design and implement high-fidelity prototypes for the application - which portray the intended look and behaviors of the application, including styling, colors, and advanced functionalities of different components. Two prototypes were made, one for mobile, and one for desktop, along with most of the core interactions planned in the application. These high-fidelity prototypes are more detailed and realistic, shown in Figure 4.3, and act as an accurate design representation upon which the actual interface will be built in the latter stage of development.

The project is then divided into two major parts: the front-end and the back-end. For this thesis, Khoa Nguyen will be responsible for implementing the front-end of this web application, while An Nguyen will work on the back-end and database of the product. For this reason, even though the two work under the same project, the system scope, architecture, and technology usage will differ entirely. With the help of the defined high-fidelity prototypes, the front-end can be further divided into smaller pages and components to apply the component-based architecture to build it.

4.2 Overall system architecture

The next step after the initial planning is to decide on the architecture of the system and start building different elements of it. The general architecture of the web application composes of three main elements: database, back-end API, and client front-end. These components work in unison, handling data flow while displaying information to the users.

The database comprises multiple SQL tables linking together. In this project, the database contains information regarding different users, events, and the relationships between those users and events - users who are friends and users who go to events. Therefore, a relational database was chosen for the fast query, read, and write speed. Data from the database will then be called and handled through the back-end server, which returns several API endpoints for the front-end to call. From the front-end, the application can make AJAX requests to get and set different data, enabling various behaviors, such as event categorization, events and people suggestions, and modifying events or people information. The back-end server uses multiple technologies, for instance, Node.js and services from Amazon AWS. Moreover, the back-end is also responsible for handling login and sign-up behaviors, shopping cart checkout functionality, and data security. An Nguyen implements these elements and will be utilized when the front-end needs to read and write data later.

As stated above, this thesis will only focus on the implementation process of the front-end of this web application - building a solid UI and great UX. With a good UI

and UX, users can see and interact with data from the back-end while communicating with the database effortlessly. The front-end must utilize and combine different information the back-end API provides correctly and efficiently. The front-end must be able to refine and process the data it fetches.

4.3 Setting up Front-end structure

The first step in building the front-end of this application is choosing, planning, and setting up a suitable architecture - defining different pages and components needed to combine and build the finished application. As stated above, the web application will function similarly to social media; hence it will contain multiple pages, each serving a specific purpose. There will be multiple components that are reusable on each page. For that reason, the component-based architecture was chosen for the project, along with React as the front-end library. The project aims to create a working platform that primarily promotes social events to end users. Therefore, event displays and recommendations should provide a good overview and essential information regarding those events. The information should be shown in an accessible way, so users can straightforwardly understand the details of the events on display.

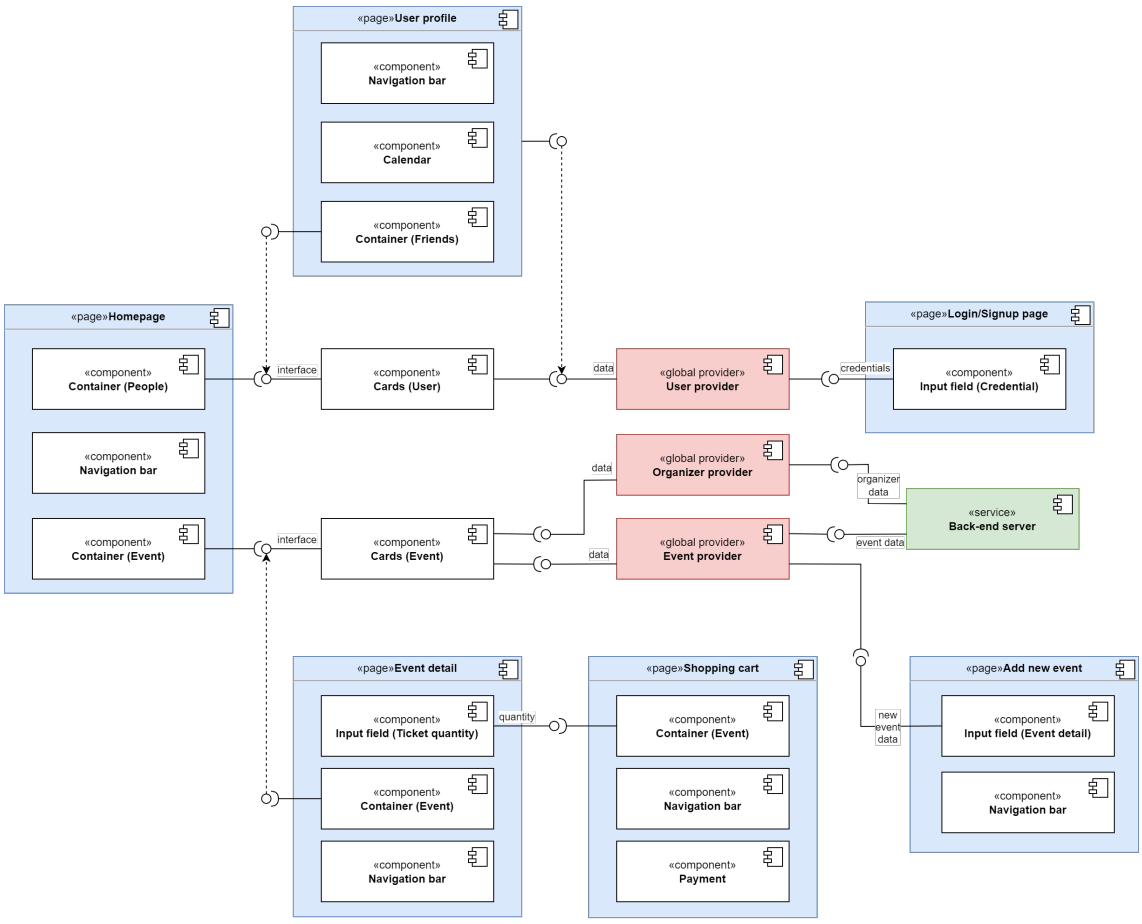

Figure 4.4 *Component diagram of the system.*

The website will have multiple pages that the users can navigate. As demonstrated in Figure 4.4 (pages are highlighted in blue), the service will have six main pages: Homepage, Add a new event, User profile, Shopping cart, Event details, and Login/Sign-up page, each composes of different components and functionalities. The data used as input for these pages is fetched from the back-end server (highlighted in green) through a series of providers (highlighted in red).

Major reusable components of the system include the navigation bar, different UI cards, containers, and input fields. These are the white components in Figure 4.4, and serve different purposes:

- **Navigation bar**: visible on almost every page (as shown in Figure 4.4) and always situates on the top. It helps users quickly and easily navigate to different places and services of the application. The implementation of this component will be discussed in Subsection 4.6.1.
- **UI cards**: a fundamental component of any interface design, these cards visually display different data. As illustrated in Figure 4.4, the Card component can show user or event information, which uses input data from different providers and produces a visual output interface. In this system, cards are usually used inside containers to display data in an orderly and cohesive manner. The implementation of this component will be discussed in Subsection 4.6.2.
- **Containers**: used to store the UI cards, these containers are responsive and can resize according to the screen resolution. As shown in Figure 4.4, the Container component uses the provided visual interface of the Card component and arranges those cards in a responsive grid. As a result, the content of these containers is based on what kind of cards they use as input. On this website, containers are usually used to store events and people cards. The implementation of this component will be discussed in Subsection 4.6.3.
- **Input fields**: can be configured to collect different user inputs, e.g., texts, numbers, or selections. As presented in Figure 4.4, these input fields can gather users' login credentials, event ticket quantity, or event detail. Other system components will consume the collected values, establishing the data flow between users and the front-end. The implementation of this component will be discussed in Subsection 4.6.4.

Combining the defined reusable components, developers can establish the structure proposed in Figure 4.4. The page system of this service is constructed as follows:

- **Homepage**: The service's main page, which the users will land upon navigating to the web application. The Homepage contains different event recommendations and event categories selection. There will also be an option for users to discover more people to befriend. In Figure 4.4, the containers on the Homepage use different event and user cards to display information.

- **Login/Sign-up page**: New users to the service can create their accounts here. After that, they will have full access to the functionalities of the service. Without login, users cannot access the database - in other words, they cannot save events to their calendars, add new events, or add new friends. As shown in Figure 4.4, this page utilized different input fields for user authentication, which logs users into the user provider for their data.

- **User profile page**: Display different information about a user - from the personal profile of the logged-in user, to the profiles of other users in this service. This profile contains basic information, such as name, profile picture, friends, and attending events. The ability to view the attending events of a user is the aspect that makes this web application unique. It can show the viewer who is attending an event and thus promote it. This page uses data from the user provider, as presented in Figure 4.4.

- **Event detail page**: Display information and additional details of an event, including the name, description, location, time, organizer, and event ticket. As illustrated in Figure 4.4, the event container of this page displays information from event cards while allowing users to purchase event tickets by the input fields. This page also shows which friends (if any) of the logged-in users will attend this event.

- **Shopping cart**: Function similarly to the shopping cart of e-commerce pages, this shopping cart contains tickets of events that the user of this session has added to their shopping basket from the event detail page, as represented in Figure 4.4. Users can checkout with their payment information and buy the tickets in this shopping cart.

- **Add new event page**: This page allows logged-in users to create events, with all the additional information regarding event details: date, time, location, and the possibility of inviting their friends to join the event. This new event will then get synced to the database of the system through the event provider, as shown in Figure 4.4.

With the six pages of the web application defined, each with its specific functionality and interaction, the development team can start designing and implementing the system's interface. As stated above, the front-end of this web application will mainly use React to build the interface, combined with Chakra UI for the styling system of

the components. This project will also utilize React Router to handle page routing and switching to and from different pages.

4.4 UX on the front-end

As discussed in the prior section, UX is one of the most critical aspects of building UI for web applications. By providing customers with good UX and a great time using the application, they will like it more and prefer using the product even in the future. Multiple systems can be utilized to build a good UX for the interface. For example, the 10 Usability Heuristics [37] lists the ten principles of interaction design by Jakob Nielsen, and the User Experience Honeycomb promoted by Peter Morville [43]. This project will use the 10 Usability Heuristics for designing the UX of the interface system.

With the 10 Usability Heuristics in mind, the UI can be implemented with good UX. Through the earlier tests and interviews, the developers can understand the needs and goals of end users. With this knowledge, the system can have interfaces that are easy to navigate, understand and support the task and goals of users. In combination with the 10 Usability Heuristics, some key functionalities are made for the system.

- This platform works for the most popular web-browsing platforms, e.g., mobile phones, tablets, and desktops. Every page is designed to keep different aspect ratios in mind, and the scaling will automatically update when used under different screen sizes. As shown in Figure 4.9 and Figure 4.5, the application automatically update different element layout and scaling as the screen gets bigger and smaller. This responsiveness ensures a great experience for users on different platforms.
- The product always notifies the user about their location in the system and provides feedback for users when they act. These identifiers can range from a page title, distinctive from other UI elements, to a notification pop-up when an action is completed. Examples of these functionalities are shown in Figure 4.6 and Figure 4.7. These functionalities implement the **visibility of system status** heuristic.
- The language of the service is the language of the users, without any technical terms and complex phrases that are hard to understand. The user should be able to use the product without wondering what a specific element means. Using the language of users implements the **match between the system and the real world** heuristic.

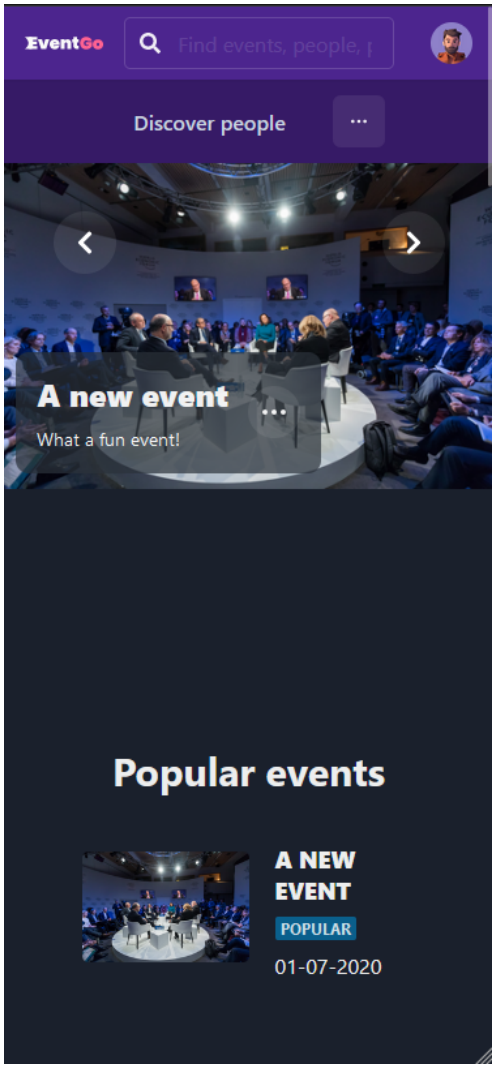

Figure 4.5 *Homepage, and platform navigation bar, on a mobile device.*

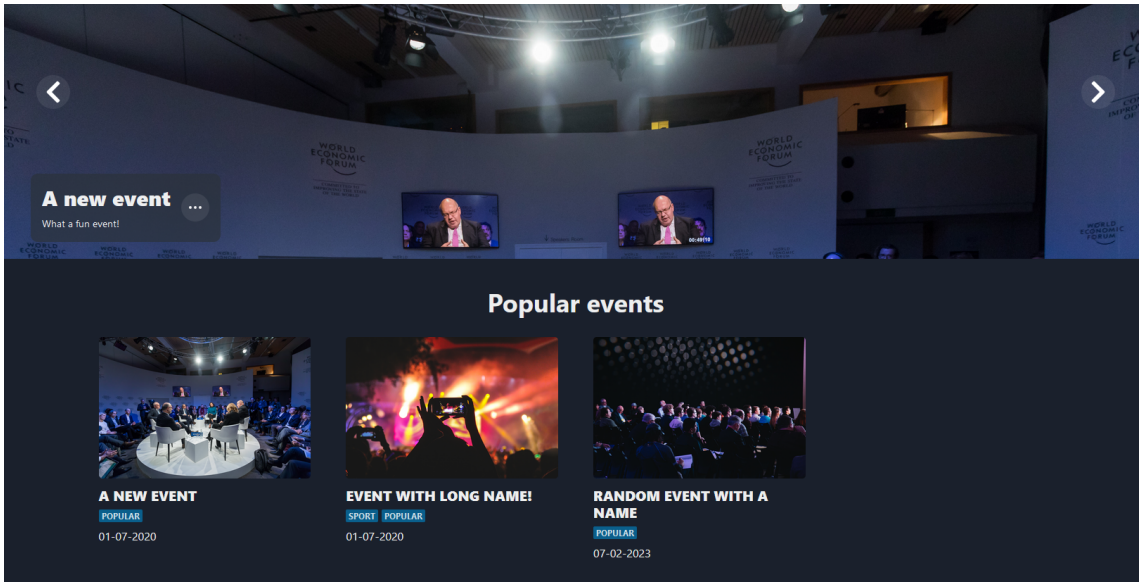

Figure 4.6 *"Popular Event" page, with its title.*

- The system has a navigation bar visible from every page that can help users to go to different places and explore various content, as shown in Figure 4.5 and

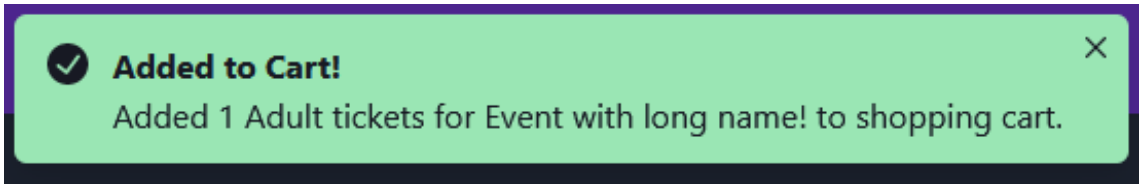

Figure 4.7 *A notification pop-up when a user adds a ticket to the shopping cart.*

4.9. The navigation bar provides users freedom of use, allowing them to fast and accurately traverse the service. This functionality follows the **flexibility and efficiency of use** heuristic.

- The buttons, labels, and icons of every element in the product are structured in a cohesive and ordered manner. This structure can also be seen in Figure 4.6, where vital information is always more significant, making it easier for the users to notice and understand them. Concise structuring is an implementation of the **match between the system and the real world** and the **recognition rather than recall** heuristic.

- When using the service, the users always have an available "exit" option when doing different tasks. This option will usually be a designated "Cancel" button, as demonstrated in Figure 4.8, that will return them to the previous screen. This functionality enforces the **user control and freedom** heuristic.

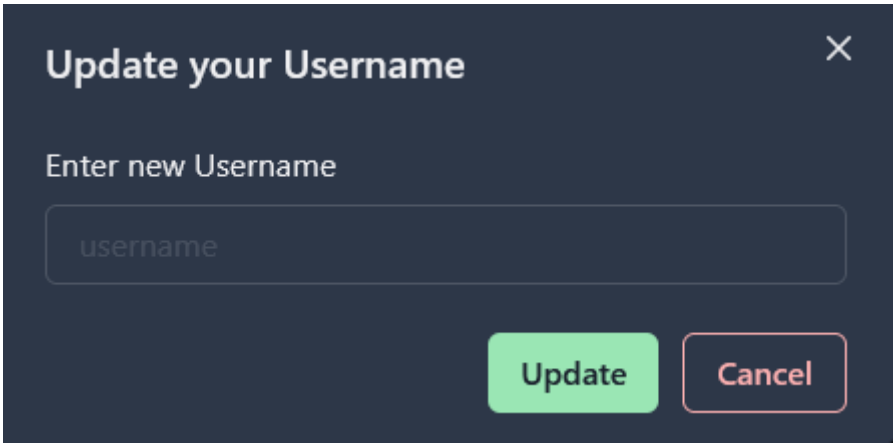

Figure 4.8 *Input pop-up box with "Update" and "Cancel" buttons.*

- The overall structure of this product is designed to be similar to other popular social media and e-commerce sites, such as Amazon, Facebook, or Twitter. Thus, the users can apply their knowledge from those popular sites to use this system without learning many new concepts. As shown in Figure 4.9, the homepage and navigation bar of the service drew inspiration from the aforementioned social media platforms. These design decisions implement the **consistency and standard** heuristic.

- This system prioritized using icons, where applicable, over plain-text labels for buttons or cards. The icons are drawn from popular and familiar icon

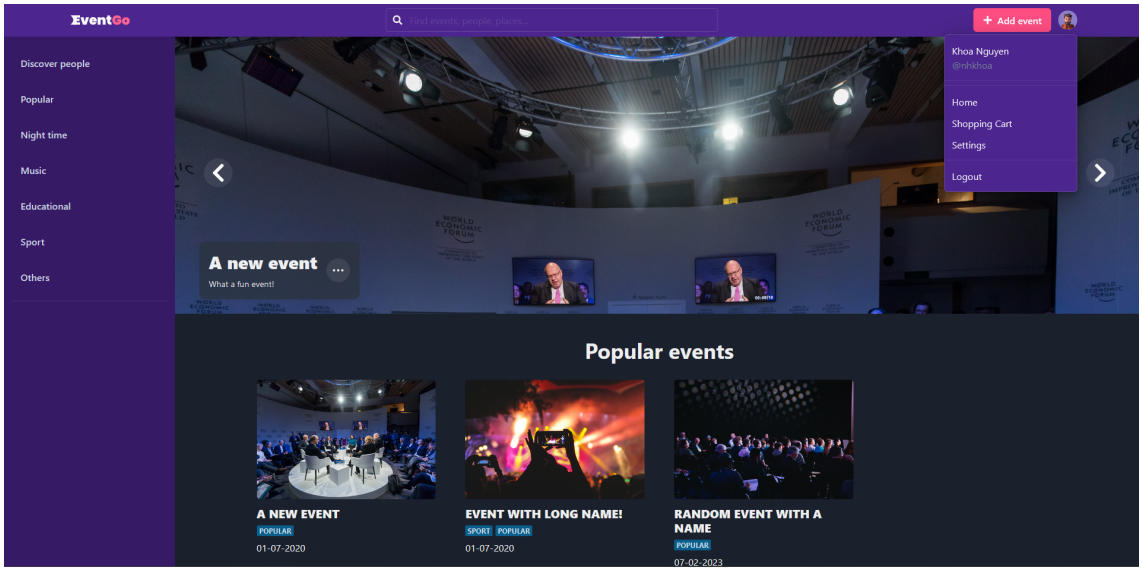

Figure 4.9 *Homepage and navigation bar of the platform.*

libraries so that users can apply their mental model to use the same icons. For example, they can know that an "X" icon means "Exit." These icons follow the **consistency and standard** heuristic.

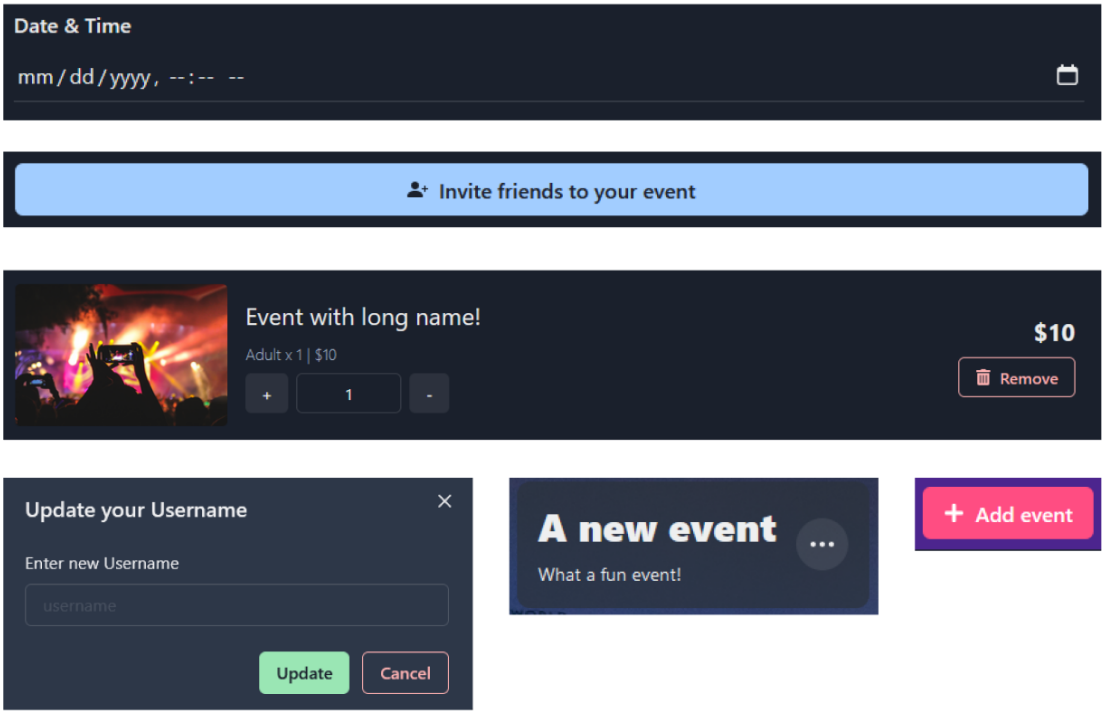

Figure 4.10 *Different UI elements with icons and text labels.*

- This system prevents user errors in multiple ways. Firstly, every error-prone input is sanitized so that a red error message will pop up if something is wrong, e.g., the user entering a letter instead of numbers or leaving a field empty. Secondly, the system will show an easy-to-understand error message that the user can check for input errors and try to submit again. These error messages and preventions comply with the **error prevention** heuristic, while

also **help users recognize, diagnose, and recover from errors**.

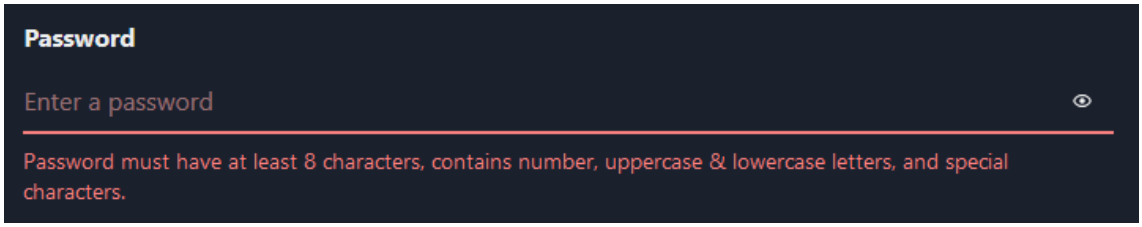

Figure 4.11 *Error message when the user leaves the field empty.*

- The design of the system aims to be minimal, avoid any information clusters and put content in a cohesive, easy-to-understand manner. Users can go through the information on each page straightforwardly without having to stop in confusion, trying to figure out what a specific part may mean. This system design obeys the **aesthetic and minimalist design**, and **recognition rather than recall** heuristic.

4.5 Communication and data persistence between back and front-end

As discussed above, the architecture of this application is composed of a database, a back-end server, and a front-end interface. Through the back-end server, the front-end fetches and displays different data and information from the system. The users can also interact with the information displayed and modify the data when needed, e.g., when users update their personal information or change their calendars. Thus, it is crucial that the communication between the back-end and front-end is consistent and that the data flow between them is fluent and fast.

4.5.1 Application API endpoints and usage

The data communication between the back-end and front-end is established through several API endpoints. Using these API endpoints, the front-end can make different AJAX requests to the back-end server to access information from the database, add new data, or modify data in it. Four API methods exist: *GET*, *POST*, *PUT*, and *DELETE*. Generally, the *GET* method fetches the database data and sends those packages to the front-end. The *POST* method will add a new entry to the database. The *PUT* method updates any detail of an entry from the database. Lastly, the *DELETE* method will delete an entry from the database. The detailed uses of these APIs will be discussed in Subsection 4.5.2.

The API endpoints defined in the back-end of Event Go are categorized into different classes; each will handle a distinct information group by utilizing multiple API methods.

- Firstly, there is the Users API class, which will handle information related to the users of this service. Users are the end-customers of this system since they mainly interact with the platform, using and making most changes to the database. Event Go uses API endpoints in the User class to fetch information about users (get a user or all users in the system), create a new user (handling sign-up), update user information, or delete a user. What is more, this API class also allows the system to interact with the friendships of a user - fetch friendships and add or remove friendships. Lastly, this API class allows Event Go to work with the personal events of users - adding a new event to their calendar and updating or deleting events from it.
- The next important class of API endpoints is the Events class. Events are a fundamental component of the system because most of the functionalities of the platform revolve around them: events recommendation, attending events and tracking them in personal calendars, buying event tickets, or making new events. For that reason, similar to the Users API class, the Events API group allows the system to fetch event information, create new events, modify events information, or delete events.
- On this platform, each event will have its corresponding organizer - an organization from the school or the user who created that event. Users can also check who organizes each event created on the platform. Thus, the back-end of Event Go defined an Organizer API class that fetches organizers information, enabling different organizers to create a new event, update an existing event, or delete an event.

Combining these API endpoints allows Event Go to create a fluent data flow between the front-end and back-end, allowing for persistent information. This data flow will be discussed in more detail in the following subsection, and the state management method implemented in the front-end will also be introduced.

4.5.2 State management and data flow on the front-end

Similar to any other interactive web service, this application needs a method to control the state of the system - so that the UI can react to changes and update the content of components accordingly. In this implementation, state management uses the *useContext* hook provided by React, which lets the application access, modify, and pass multiple global contexts from different components [44]. The traditional method of passing data between components in React is through ”props”, which only allow data to pass from a parent component to its children - or ”props drilling”. This practice can become verbose and inconvenient when some props must be passed deeply through the component tree. Some data must persist globally in

this application, as Figure 4.12 illustrates. This situation can be solved by using a globally available ”data store”, and every component can have access to read and write to that store. Solving this problem is where context comes in. A context in React lets a parent component provide data to all of its children, no matter how deep in the component tree that child is. The children can then freely access and modify the data in these contexts without passing props around.

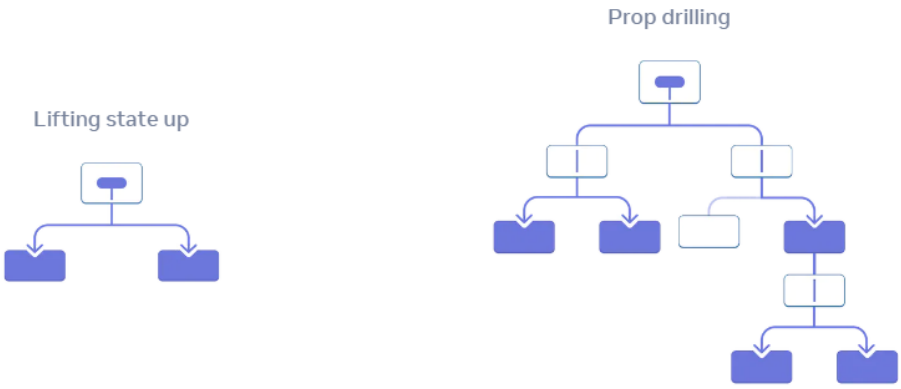

Figure 4.12 *React state lifting and prop drilling [44].*

In a React application, the *createContext* function can define a new context. First, define a new context with *createContext*. Then, using the *Provider* of React Context, developers can create different data states and define multiple functions inside the created context. These functions can be defined similarly to any other JS functions operating on top of the context state. They can be used in other components to modify and update the states inside the context. The context can then be exported and used in other places. Program .3 shows an example of defining such context.

The context can be used in a parent component, by wrapping the context provider around the child components. The child components wrapped inside the context provider can access all the values given by the context. This method of wrapping the context provider around the child components is shown in Program .4.

Then, the child component can use the data and functions provided by the context using the *useContext* hook in React. An example of using the useContext hook in a child component to access a context is shown in Program .5.

This application has four different contexts, and each will handle a group of data from the back-end. There are *shoppingCartContext.js*, *EventContext.js*, *Organizer-Context.js*, and *UserContext.js*. Each of these will handle data dedicated to their names.

- *EventContext.js*: handles data related to every event. This context includes fetching event data from the back-end, getting the different event tags, and

providing event groups to display on the interface as event categories.

- *OrganizerContext.js*: handles data related to event organizers. This context includes fetching organizers' data such as email, name, and tags. This information can then be used as organizer information in each event.
- *ShoppingCartContext.js*: handles items that the users have added to their shopping cart. With this context, the application can update and perform different actions on the items in the shopping cart, for instance, add or remove an item, get the total price, payment, or clear the cart.
- *UserContext.js*: This is the most complicated context of this service, which handles every user-related data - from user login, logout, and register, to getting all users' personal information and their friends. This context also allows users to add new friends, remove their friendship, and add or remove events to their calendars.

The defined context can handle data fetching and modification by utilizing different Axios methods combined with the API endpoints provided by the back-end server. As a simple promise-based HTTP client for the browser, Axios offers an easy-to-use library with a highly extendable interface [45]. In this implementation, Axios performs multiple AJAX requests to the back-end server to fetch and modify data. Shown in Program .6 is the process of using Axios *GET* to fetch user data, given the user's email.

Similarly, the system can use Axios *POST* to add new entries to the database. For example, as shown in Program .7, the application uses the POST method from Axios to add a new friend to a user.

Lastly, the system can delete an existing entry from the database using Axios *DELETE*. Illustrated in Program .8 is how the service uses the *DELETE* method from Axios to delete a friend from the friends network of a user, effectively "un-friending" them.

In the above context, these methods are used in multiple functions to get, create, and modify data from the database. Using React context, the data and these functions are available globally - any component can use them. With this approach, it is easy to set up a component system that works effortlessly with each other and saves development time. By defining data flow in one place - the different contexts - the code is easy to maintain and debug and scalable.

4.6 Styling system and building interface components

The next step of the implementation process is to start building the styling system for the platform and making different reusable interface components. The application can scale and reuse code using a globally defined styling system and component-based architecture. This way, the development team can maintain and expand the platform better in the future.

A styled system creates a set of defined decorations of different components - so that this different variant can be applied when that component is used. Chakra UI has a default styling system for all of its components, onto which the development team can implement their custom elements or extend them. In Chakra UI, the comprehensive theming system of the application can be defined using a *theme.js* file, as shown in Program .9. Then, this developer-defined theme is available everywhere in the code base. Developers can call a variant of a component and use a custom color everywhere in the implementation process.

With this styled system, the whole project can have multiple global properties defined in one place, making it easier to control and make changes later. Moreover, it is also easier to define unique colors, that are not default in Chakra UI, without using the hex code of that color. Chakra UI automatically uses this comprehensive theme system after definition, so there is no need to import it anywhere - it is available globally.

After the global styling system is completed, the design team can start implementing the ”building block” of the service - different interface components - which can be combined to create the completed UI of the application. The components to build are those that are being used across different pages or repetitive - being children of a more significant component. These subsections will provide an overview of the design and building process of the most fundamental components of this system.

4.6.1 Implementing the navigation bar

The navigation bar on almost every application page helps users access different service pages. It situates at the top of the web page and contains multiple smaller buttons, which users can use to navigate the system, log in and log out, search, and create new events.

The navigation bar uses React Router to handle page routing, allowing users to switch between different application pages. React Router defines *Link* to different application pages and automatically navigates the user to that *Link* when clicked. The result is a responsive navigation bar, as shown in Figure 4.5 and 4.9.

4.6.2 Implementing different card components

The card components are one of the fundamental components of any web application. They can be found anywhere and are used to display information pleasantly and tidily. These cards usually have an image or other forms of visual media, a headline or title, some description, and sometimes action buttons or links, as shown in Figure 4.13.

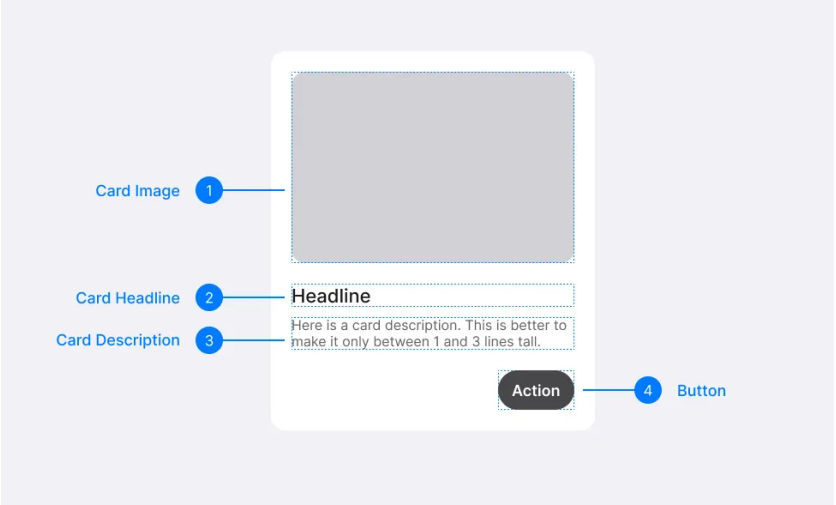

Figure 4.13 *Anatomy of a card component [46].*

The most important aspect of these cards is that they take standardized data input to create a uniform visual output and can be easily reused in multiple places. Figure 4.14 shows different cards from other web services. In the examples, it is evident that these cards follow the same anatomy discussed above.

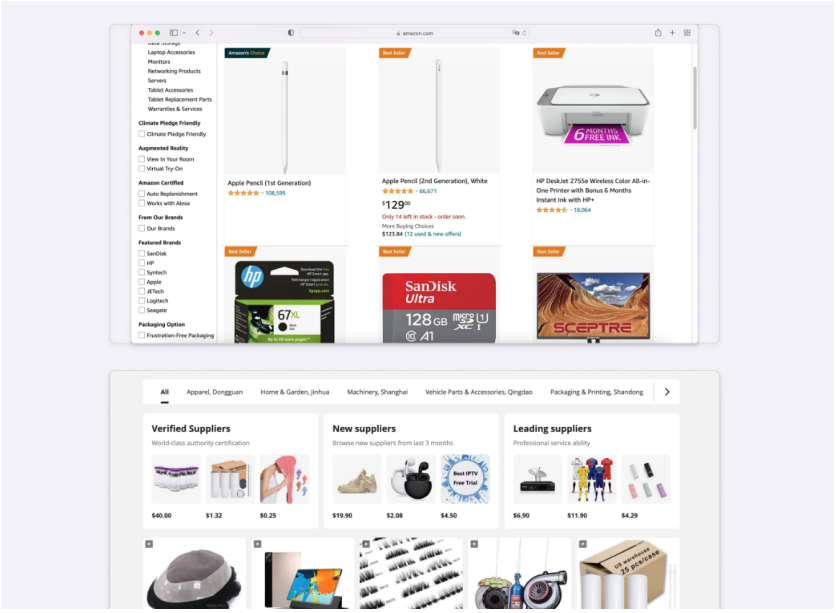

Figure 4.14 *Examples of card components from Amazon (top) and Alibaba (bottom) [46].*

These cards are usually implemented using the *Box* element of Chakra UI to define the overall shape of the card. Then, the *Image* element is used to display the

visual media in combination with the *Heading* and *Text* elements for the title and description of the card. Lastly, the card can have *Button* or *Lin*k elements as active components. Program .10 shows an example of defining a card element.

The system has multiple cards that display information on a small panel. These cards display information ranging from events, people and friends, to items in the shopping cart. Most of the information in the service is displayed in this manner, and these cards are then grouped up into different containers, which build the general layout of the web page.

4.6.3 Implementing containers

The application has multiple containers that display different children components, including the Card mentioned earlier, multiple buttons, or any other UI components. The most important aspect of these containers is their responsiveness, or how they can scale and modify the layout of their content to fit the screen resolution.

The *SimpleGrid* component of Chakra UI was used to make the container responsive and change the layout based on different screen resolutions, in combination with the object-based media query of Chakra UI. The child components - cards, buttons, or any other UI elements - are laid out within this grid. An example of this implementation can be seen in Program .11.

The containers can then be used in multiple places in the application. Since its clear input and output, it is easy to adopt and can be integrated anywhere.

4.6.4 Implementing input fields

Another common component to be implemented is different input fields in the system. Because the platform is highly interactive, users will have multiple fields to fill in the information. These components are implemented with the *FormControl* component from Chakra UI combined with the *useState* hook of React. This combination ensures that the form input from users is reactive, and it will store the entered data in a React state, which can be accessed when the form is submitted. An example of this approach can be seen from Program .12.

With these core components defined, the development team can build the system. They can combine these components and other UI elements, such as bottoms and links, to implement a working front-end for the service. There can be multiple variants of the cards, containers, and input fields, but their anatomy and structure should remain the same.

5 Result analysis and Future development

The scope of this thesis has shown the development process of this social media platform, from its initial design, planning, and studying to implementing the architecture and building different components for the front-end. The thesis also explains the different technology choices and discusses the software architecture. In the end, this chapter will discuss what this project has achieved and what is there for future development.

5.1 Current prototype analysis

The UI design process of Event Go follows the 10 Usability Heuristics to create user-friendly interfaces. The designs were made in Figma and were tested by the development team to find possible UX issues. Using the high-fidelity prototypes from Figma, Event Go developers can implement the front-end using React and Chakra UI while ensuring the usability of the UI. With these tools, the resulting prototype is interactive, responsive, and available for different user groups on multiple platforms. However, usability testing and evaluation with end-users have not been carried out on the final prototype because of the time restriction. For that reason, the development team needs to plan and test the UX of this platform in the future with real, potential users to identify and fix potential drawbacks. Only then the usability of this system can be polished, and the application can be officially published.

The social media project discussed in the scope of this thesis is available through Netlify [47], with the front-end source code open at a GitHub repository [48]. In the current implementation of this application, the development team managed to design and build a working skeleton for the platform, with its most basic functionalities in place. Most of the desired functions of the system were implemented.

- Users can create a new account for the service and log in to their account.
- Users can browse and get recommendations for different events on the platform.
- Users can purchase, view their shopping cart, and make payments to buy event tickets.
- Users can discover new people, add new friends, and view their information. Users can also unfriend and remove people from their friend list.

- Users can view their personal information and references.
- Organizers can create new events for users to attend.

Within the scope of this thesis, this implementation has finished the most basic functionalities of the platform. Future improvements can be designed and built efficiently with the existing component-based architecture. The project also has the design and styling system defined, which can ensure proper UX in its interface.

5.2 Prototype limitations and Future development

Despite the architecture and basic structure finished, this platform implementation still needs some distinct features that make this social media service stand out. Those potential functionalities include:

- Personal event creation for users. As of now, only organizers can create new events. The current implementation does not allow users to create their events and invite their friends to attend them - albeit this was listed as one of the fundamental features of this system. However, the front-end implementation of this function is relatively easy. At the current state, the navigation bar of the front-end already has a button that takes the user to a page where they can fill in forms to create a new event. This page can later be linked to a back-end API that handles event creation and adds a new event to the database.
- Modification of users personal information. Users can view their personal information in the settings but cannot modify it for now. The API endpoints to modify the data exist but are not integrated into the front-end. Therefore, this feature can be implemented efficiently using the readily made *PUT* methods in the Users API group that handles updating user information. The development team only needs to connect the existing forms in the setting page to submit an AJAX request to the API, as mentioned above, to update the user's personal information.
- Event calendar synchronization. Even though this function is listed as one of the goals of the project, it is not implemented. Users can only access their calendar in this application but cannot sync another calendar (Google Calendar, Outlook) into the calendar of this system. This functionality can be done in the back-end server and hooked to the front-end. This functionality allows users to seamlessly transfer their events from one calendar to another without manually creating and adding events into different calendars.

- No searching function. The content of this system is relatively small at the moment, so no dedicated search feature is implemented. There are multiple methods of implementing a search function for the system. The first would be to make a search algorithm from scratch in the back-end, and access that through an API. Making a search function from scratch would be costly and hard to design, but the developer would have total control of the search. Another method is integrating a readily defined search library for React.js into the front-end. This approach is easier to implement and costs less time and effort, but it would require some setup in the database, and developers would not like to have complete control over the search outcomes.

- Integration with other social media platforms. Cross-platform sign-in has become a standard for modern web applications, and almost every social media platform can connect. Nowadays, most prominent social media platforms have their dedicated API, which Event Go can use to perform the integration. The integration can be done in the back-end server of the system, and the result can be called in the front-end as an API, and users can use a button to perform the connection. For example, in the future, users can connect their Event Go account to their Google account to sync their Google Calendar into the calendar of Event Go and vice versa. Alternatively, a user can connect to their Facebook or Twitter account and find friends using Event Go to connect. The possibilities are endless.

- Proper software tests. The functionalities of this system are tested by the development team only, and no dedicated test system has been implemented. Many testing libraries are available for React to define unit tests; similarly, a testing framework can also be utilized to implement integration tests for the UI. These tests ensure that the UI can run smoothly without problem, providing users with a smooth experience.

With these limitations introduced, the expansion potential of Event Go is excellent. In the future, there will be multiple different features and functionalities that the development team can take on and implement to scale and expand the platform further.

6 Conclusion

This thesis explores the benefits of component-based web architecture for creating scalable and maintainable web applications with a user-friendly interface. Component-based web architecture has been introduced and used widely, gaining immense popularity thanks to its innovative building approach to making interfaces. The development team can build scalable, maintainable, and efficient systems with a component-based approach. Although its numerous advantages in many development aspects, component-based architecture is not a panacea for creating web applications. Developers must consider the benefits and drawbacks of the component-based approach compared to other structures before using it to build the interface.

In combination with component-based architecture, designing an excellent UX for the interface was also discussed in the scope of this thesis. By following defined rules and heuristics, designers and developers can build responsive and intuitive UI, significantly improving the UX of the service. This thesis also provides some examples of how to implement good UX in the front-end and explains the workflow of the development team when designing and testing the UI of this system.

The development process of an events management web application for university students is discussed, using the component-based architecture to build the interface while also focusing on UX. Overall, the thesis presents the development process in detail, from the initial planning stage to the complete working prototype of the application. This thesis can serve as a detailed case study for inspiring front-end developers, which explains different technology and guidelines that a developer can follow to deliver a scalable, easy-to-maintain, intuitive, and user-friendly front-end.

In conclusion, this thesis has covered many theoretical backgrounds on building an excellent, intuitive interface for web applications. It also shows the process of planning, designing, and implementing a web service's front-end from start to finish. Even though the final prototype is missing some of its key features, the future of the Event Go project is optimistic. When finished, it can help university students manage their schedules tremendously.

References

[1] B. Gallina, M. A. Javed, F. U. Muram, and S. Punnekkat, "A model-driven dependability analysis method for component-based architectures," in *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, 2012, pp. 233–240. DOI: 10.1109/SEAA.2012.35.

[2] S. Chardigny, A. Seriai, M. Oussalah, and D. Tamzalit, "Extraction of component-based architecture from object-oriented systems," in *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, 2008, pp. 285–288. DOI: 10.1109/WICSA.2008.44.

[3] *Building reliable component-based software systems* (Artech House computing library), eng. Boston: Artech House, 2002, ISBN: 1-58053-558-5.

[4] C. Y. Baldwin and K. B. Clark, "Modularity in the design of complex engineering systems," in *Complex Engineered Systems: Science Meets Technology*, D. Braha, A. A. Minai, and Y. Bar-Yam, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 175–205, ISBN: 978-3-540-32834-6. DOI: 10.1007/3-540-32834-3_9.

[5] N. Chondamrongkul, J. Sun, B. Wei, and I. Warren, "Parallel verification of software architecture design," in *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, 2019, pp. 50–57. DOI: 10.1109/HASE.2019.00018.

[6] N. Dragoni, S. Giallorenzo, A. L. Lafuente, *et al.*, "Microservices: Yesterday, today, and tomorrow," eng, *arXiv.org*, 2017, ISSN: 2331-8422.

[7] M. Nouman and M. Azam, "A systematic review of non-functional requirements mapping into architectural styles," *Bulletin of Electrical Engineering and Informatics*, pp. 1226–1236, Apr. 2023.

[8] D. Raggett, A. Le Hors, I. Jacobs, *et al.*, "Html 4.01 specification," *W3C recommendation*, vol. 24, 1999.

[9] E. A. Meyer, *CSS: The Definitive Guide: The Definitive Guide.* " O'Reilly Media, Inc.", 2012.

[10] D. Crockford, *JavaScript: The Good Parts: The Good Parts.* " O'Reilly Media, Inc.", 2008.

[11] M. Persson, *Javascript dom manipulation performance: Comparing vanilla javascript and leading javascript front-end frameworks*, 2020.

[12] T. Powell, F. Schneider, and N. Maragioglio, *JavaScript: The complete reference*. McGraw-Hill, Inc., 2004.

[13] X. Zhang, Y. Zhang, and J. Wu, "Research and analysis of ajax technology effect on information system operating efficiency," in *Research and Practical Issues of Enterprise Information Systems II*, L. D. Xu, A. M. Tjoa, and S. S. Chaudhry, Eds., Boston, MA: Springer US, 2008, pp. 641–649, ISBN: 978-0-387-75902-9.

[14] A. Pano, D. Graziotin, and P. Abrahamsson, "Factors and actors leading to the adoption of a javascript framework," in *Empirical software engineering : an international journal*, vol. 23, 2018, pp. 3503–3534. DOI: 10.1007/s10664-018-9613-x.

[15] StackOverflow, *Stackoverflow 2022 developer survey*, 2022. [Online]. Available: https://survey.stackoverflow.co/2022, (Accessed: 8/04/2023).

[16] *React*. [Online]. Available: https://reactjs.org/, (Accessed: 23/04/2023).

[17] *Vue.js*. [Online]. Available: https://vuejs.org/, (Accessed: 23/04/2023).

[18] *Angular*. [Online]. Available: https://angular.io/, (Accessed: 23/04/2023).

[19] H. Narayn, *Just React: Learn React the React Way*. Berkeley, CA: Apress L. P, 2022.

[20] S. Aggarwal *et al.*, "Modern web-development using reactjs," *International Journal of Recent Research Aspects*, vol. 5, no. 1, pp. 133–137, 2018.

[21] M. K. Caspers, "React and redux," *Rich Internet Applications w/HTML and Javascript*, vol. 11, 2017.

[22] *Netflix likes react*, Netflix Technology Blog, Jan. 28, 2015. [Online]. Available: https://netflixtechblog.com/netflix-likes-react-509675426db, (Accessed: 09/04/2023).

[23] *The evolution of scalable css*, Frontend Master, Nov. 12, 2022. [Online]. Available: https://frontendmastery.com/posts/the-evolution-of-scalable-css/, (Accessed: 09/04/2023).

[24] J. Karlsson, *Responsive web design with css frameworks*, 2014.

[25] M. Laaziri, K. Benmoussa, S. Khoulji, K. M. Larbi, and A. El Yamami, "Analyzing bootsrap and foundation font-end frameworks: A comparative study," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 713–722, 2019.

[26] *Bootstrap*. [Online]. Available: https://getbootstrap.com/, (Accessed: 10/04/2023).

[27] S. Bose, *Top 5 css frameworks for developers and designers*, BrowserStack, Mar. 25, 2022. [Online]. Available: https://www.browserstack.com/guide/top-css-frameworks, (Accessed: 28/03/2023).

[28] *Foundation*. [Online]. Available: https://get.foundation/, (Accessed: 15/03/2023).

[29] *Bulma*. [Online]. Available: https://bulma.io/, (Accessed: 15/03/2023).

[30] *Tailwind css*. [Online]. Available: https://tailwindcss.com/, (Accessed: 15/03/2023).

[31] *Chakra ui*. [Online]. Available: https://chakra-ui.com/, (Accessed: 26/03/2023).

[32] *The trillion dollar ux problem: A comprehensive guide to the roi of ux*, The UX School, Aug. 3, 2017. [Online]. Available: https://s3.amazonaws.com/coach-courses-us/public/theuxschool/uploads/The_Trillion_Dollar_UX_Problem.pdf, (Accessed: 10/04/2023).

[33] D. Benyon, *Designing user experience*. Pearson UK, 2019.

[34] J. McRee, T. E. Team, R. Wilson, and J. Anderson, *Effective UI: building great user experience-driven sites and software*, eng. O'Reilly, 2010, ISBN: 9780596154783.

[35] P. W. Jordan, *An introduction to usability*. Crc Press, 1998.

[36] M. Kulkarni, "Digital accessibility: Challenges and opportunities," *IIMB Management Review*, vol. 31, no. 1, pp. 91–98, 2019.

[37] J. Nielsen, "Enhancing the explanatory power of usability heuristics," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1994, pp. 152–158.

[38] P. Skalski, R. Tamborini, A. Shelton, M. Buncher, and P. Lindmark, "Mapping the road to fun: Natural video game controllers, presence, and game enjoyment," *New Media & Society*, vol. 13, no. 2, pp. 224–242, 2011.

[39] J. Yablonski, *Laws of UX: Using psychology to design better products & services*. O'Reilly Media, 2020.

[40] *Human-technology interaction 2: Design, lectures*, Tampere University. [Online]. Available: https://www.tuni.fi/en/study-with-us/human-technology-interaction-2-design-lectures, (Accessed: 20/03/2023).

[41] *Figma*. [Online]. Available: https://www.figma.com/, (Accessed: 12/04/2023).

[42] K. Nguyen, A. Nguyen, and D. A. Pham, *Event go - web service*. [Online]. Available: https://www.behance.net/gallery/134056463/EventGo-Web-Service, (Accessed: 20/04/2023).

[43] P. Morville, *User experience design*, Jun. 21, 2004. [Online]. Available: http://semanticstudios.com/user_experience_design/, (Accessed: 18/03/2023).

[44] *Passing data deeply with context*, React. [Online]. Available: https://react.dev/learn/passing-data-deeply-with-context, (Accessed: 07/04/2023).

[45] *Axios*. [Online]. Available: https://axios-http.com/docs/intro, (Accessed: 23/04/2023).

[46] A. Prokhorova, *8 best pratices for ui card design*, Sep. 14, 2022. [Online]. Available: https://uxdesign.cc/8-best-practices-for-ui-card-design-898f45bb60cc, (Accessed: 07/04/2023).

[47] K. Nguyen, *Event go website*, 2023. [Online]. Available: `https://tuni-event-go.netlify.app/`, (Accessed: 20/04/2023).

[48] K. Nguyen, *Event go repository*, 2023. [Online]. Available: `https://github.com/hkhoa-ng/event-go`, (Accessed: 20/04/2023).

Appendix

This appendix contains the code snippet examples for this thesis.

```
<html>
  <head>
    <title>My first HTML document</title>
  </head>
  <body>
    <h1>My HTML example</h1>
    <p>Hello world!</p>
  </body>
</html>
```

Program .1 *Example of an HTML document.*

```css
h1 {
  color: maroon;
  font: bold 2rem Times, serif;
  text-decoration: underline;
  border: 5px solid black;
  margin: 0;
  padding: 10px;
}
```

Program .2 *Example of changing the styling of h1 element using CSS.*

```javascript
import { createContext, useState } from 'react';

const NewContext = createContext();

export function NewContextProvider({children}) {
    const [state, setState] = useState('new state');

    // An example of a function to update the state of this context
    function updateState(newState) {
        setState(newState);
    }

    // Define more functions in the body of this context provider

    return (
        // Return the context provider, and put the defined data states with functions in the
        // value of the provider
        <NewContext.Provider
            value={{
                state,
                updateState,
            }}
        >
            // Here, we wrap the provider outside the children components so that they can
            // access data in the context
            {children}
        </NewContext.Provider>
    )
}
```

Program .3 *Example of defining a new React context.*

```javascript
import { NewContextProvider } from './NewContext';
import ChildComponent from './ChildComponent';

export default function ParentComponent() {
    return (
        <NewContextProvider>
            <ChildComponent />
        </NewContextProvider>
    )
}
```

Program .4 *Context wrapping for children components of ParentComponent.js.*

```
import React, { useContext } from 'react';
import NewContext from './NewContext'

export default function ChildComponent() {
    // Import the data and functions provided by the context
    const { state, updateState } = useContext(NewContext);

    // Those functions and state can then be used in this component
    console.log('The state from NewContext is: ' + state);
    updateState('update a new state');
}
```

Program .5 *Accessing provided context data in ChildComponent.js.*

```
import axios from 'axios';

// URL of the backend server
const backendURL = 'https://back.end.url'

// Define Axios header configuration
const config = {
    headers: {
        'x-api-key': xApiKey,
        'Content-Type': 'application/json',
    },
};

function getUserByEmail(userEmail) {
    axios
        .get(`${backendURL}/users/${userEmail}`, config)
        .then(response => {
            console.log(response.data);
        })
        .catch(error => {
            console.error(`Error while getting user data: ${error}`);
        });
};
```

Program .6 *Fetching personal user data using Axios GET.*

```
import axios from 'axios';

// URL of the backend server
const backendURL = 'https://back.end.url'

// Define Axios header configuration
const config = {
    headers: {
        'x-api-key': xApiKey,
        'Content-Type': 'application/json',
    },
};

async function addFriendToUser(userEmail, friendEmail) {
    // Define the data package to post
    const data = {
        userEmail: userEmail,
        friendEmail: friendEmail,
    };
    axios
        .post(`${backendURL}/user/friend`, data, config)
        .then(res => {
            console.log(
                `Successfully added ${friendEmail} as a friend to ${
                    userEmail}`
            );
            console.log(res.data);
        })
        .catch(err => {
            console.error(
                `Error while trying to add ${friendEmail} as friend of
                    ${userEmail}: ${err}`
            );
        });
  }
```

Program .7 *Adding a new friend to a user using Axios POST.*

```
import axios from 'axios';

// URL of the backend server
const backendURL = 'https://back.end.url'

// Define Axios header configuration
const config = {
    headers: {
        'x-api-key': xApiKey,
        'Content-Type': 'application/json',
    },
};

async function deleteFriendFromUser(userEmail, friendEmail) {
    axios
        .delete(
            `${backendURL}/user/friend?user_email=${userEmail}&
               friend_email=${friendEmail}`,
            config
        )
        .then(res => {
            console.log(
                `Successfully deleted ${friendEmail} from ${userEmail}
                   friend list`
            );
            console.log(res.data);
        })
        .catch(err => {
            console.error(
                `Error while trying to delete ${friendEmail} from ${
                   userEmail} friend list: ${err}`
            );
        });
}
```

Program .8 *Removing friendship from a user using Axios DELETE.*

```
import { extendTheme } from '@chakra-ui/react';
import { mode } from '@chakra-ui/theme-tools';

const styles = {
  global: props => ({
    body: {
      bg: mode('white', 'gray.800')(props),
    },
  }),
};

const components = {
  Heading: {
    variants: {
      'product-name': {
        // Styling for this variant is defined here
      },
    },
  },
};

const fonts = {};

const colors = {
  customerColor: '#f4f4f8',
  // More colors can be defined here in a similar manner
};

const config = {
  initialColorMode: 'dark',
  useSystemColorMode: false,
};

const theme = extendTheme({
  config,
  styles,
  components,
  fonts,
  colors,
});

export default theme;
```

Program .9 *Example of extending the default styling theme of Chakra UI.*

```jsx
import React from 'react';
import { Box, Text, Heading, VStack, HStack, Link } from '@chakra-ui/react';
import { Link as ReactLink } from 'react-router-dom';

// Passing event information as prop for the card
function SmallEventCard({ event }) {
  const eventTime = new Date(event.event_time);
  return (
    // Using a Link as the parent component to allow users to navigate to other page when
    // clicking on this card
    <Link as={ReactLink} to={`/${event.event_id}`} w="100%">
      // HStack (Horizontal stack) to make a horizontal layout
      <HStack gap="20px" p={5}>
        <Box
          bg="gray.700"
          w="5px"
          h="80px"
          _hover={{ background: 'gray.800', textDecoration: 'none' }}
        />
        // VStack (Vertical stack) to make a vertical layout
        <VStack alignItems={'left'} w="100%">
          <Heading fontSize={{ base: '1.2rem', lg: '1.5rem' }}>
            {event.event_name}
          </Heading>
          <Text>{`Time: ${eventTime.toLocaleTimeString()}, ${eventTime.toDateString()}`}</Text>
          <Text>{`Location: ${event.location}`}</Text>
        </VStack>
      </HStack>
    </Link>
  );
}

export default SmallEventCard;
```

Program .10 *Example of defining a new Card component that shows event information in React with Chakra UI.*

```jsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { Heading, SimpleGrid, Divider, Button, VStack, HStack } from '@chakra-ui/react';
import { nanoid } from 'nanoid';
import EventCard from '../cards/EventCard';

// Import events data as prop for this element
export default function EventContainer({ events, type }) {
  const [elements, setElements] = useState();
  const navigate = useNavigate();

  // Create EventCard children elements to put inside the container
  setElements(
    events.map(event => <EventCard event={event} key={nanoid()} />)
  );

  return (
    <VStack m={10} mb={20}>
      <Heading textAlign="center">
        {type.replace(/^\w/, c => c.toUpperCase())} events
      </Heading>

      // A simple grid with the number of columns changed based on the screen size
      <SimpleGrid
        columns={{ base: 1, md: 2, lg: 3, xl: 4 }}
        spacing="10px"
        maxW={{ base: '100%', md: '90%' }}
      >
        {elements}
      </SimpleGrid>
      {props.showMore && (
        <HStack w="90%">
          <Divider />
          <Button
            w={{ base: '60%', sm: '45%', md: '40%', lg: '35%', xl: '20%' }}
            onClick={() => {
              navigate(`/${props.type.replace(/\s+/g, '-')}-events`);
            }}
          >
            Show more
          </Button>
          <Divider />
        </HStack>
      )}
    </VStack>
  );
}
```

Program .11 *Example of defining a responsive container in React with Chakra UI.*

```
import React, { useState } from 'react';
import { Button, Text, Input, FormControl, FormLabel, VStack } from '
    @chakra-ui/react';

export default function NewForm({ submitForm }) {
  const [input, setInput] = useState('');

   <VStack>
      <Text alignSelf={'left'} w="100%">
        Sign up using your credentials.
      </Text>
      <FormControl>
        <FormLabel style={labelStyle}>Email address</FormLabel>
        <Input
          mt="-1rem"
          type="text"
          placeholder="Enter your input here!"
          value={input}
          onChange={e => {
            setInput(e.target.value);
          }}
        />
      </FormControl>
      <Button
        onClick={() => {
          // Handle form submitting here!
          submitForm(input);
        }}
        w="100%"
        colorScheme={'green'}
        variant="solid"
      >
        Submit
      </Button>
    </VStack>
}
```

Program .12 *Example of defining an input form in React with Chakra UI.*