POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Design and implementation of web components

Supervisors

Prof. GIOVANNI MALNATI

Candidate

Mostafa ASADOLLAHY

July 2024

Summary

In order to improve user experience and application functionality, this thesis explores the design and implementation of complex web components for a caregiving platform, making use of cutting-edge technology. Keycloak was used to integrate secure authentication; TypeScript and Redux RTK Query were used to facilitate efficient data management; React was used to create user-friendly interfaces with features like infinite scrolling and debounced search; and Kepler.gl was utilized to enable advanced data visualization for geospatial mapping of patient locations.

The study makes an important contribution to the field of web development by showing the successful integration of several technologies, especially in the context of healthcare applications. These include Kepler.gl for perceptive data visualization, React and Redux for dynamic and responsive user interfaces, Keycloak for strong user authentication, and TypeScript for improved code dependability and maintainability. The complicated requirements of healthcare systems are addressed by this well-thought-out integration, which focuses on enhancing user interfaces, data handling, and security.

The project's outcomes highlight the potential of modern web technologies to significantly improve the efficiency, security, and user engagement of healthcare applications. The integration of these technologies has led to a more maintainable and error-resistant codebase, secure and efficient user authentication processes, optimized state management, and enhanced data visualization capabilities that support caregiver decision-making.

The produced prototype provides a flexible and clean practical framework that forms the basis for future applications. It emphasizes how crucial complete technology integration is to tackle the many issues that healthcare services must deal with. The thesis ends with research directions that should be pursued, such as investigating real-time data updates, improving state management strategies, and developing interactive geospatial data visualization tools.

This summary encapsulates the essence of the thesis, emphasizing the innovative integration of technologies and their practical applications in enhancing web-based healthcare services, and provides a clear pathway for future research and development in this vital area.

Acknowledgements

I am deeply grateful to my supervisor, Professor Giovanni Malnati, for your invaluable guidance and support throughout this journey. Your expertise was essential to my success.

Special thanks to David Lomuscio for your insightful feedback and suggestions that improved my work.

I must also thank my friends for your moral support and encouragement, and my family, especially my parents, for your endless love and support.

To everyone who supported me, thank you for making this thesis possible.

To all these great people, I owe the realization of this thesis.

Table of Contents

List of Tables

List of Figures

Acronyms

SSO
: single sign-on

MFA
: multi-factor authentication

RBAC
: role-based access controls

ABAC
: attribute-based access control

JWT
: JSON Web Token

JSX
: JavaScript XML

DOM
: Document Object Model

HTML
: HyperText Markup Language

CSS
: Cascading Style Sheets

SPA
: single-page application

RSA	Rivest-Shamir-Adleman
JSON	JavaScript Object Notation
HMAC	Hash-Based Message Authentication Code
ECDSA	Elliptic Curve Digital Signature Algorithm
CSRF	Cross-Site Request Forgery
OWASP	Open Web Application Security Project
XSS	Cross-Site Scripting

Chapter 1

Introduction

1.1 Background and Motivation

In the fast-changing landscape of digital technology, web apps have become critical in a variety of industries, including healthcare, education, finance, and others. These applications not only improve operational efficiency but also increase user engagement through dynamic and interactive platforms. However, the complexity of today's online applications brings serious challenges, particularly in terms of security, data management, and user interface responsiveness. The need for resilient, scalable, and secure applications is greater than ever, particularly in sensitive sectors such as healthcare, where data integrity and security are important.

This thesis is motivated by the problems and requirements of a caregiving platform, in which effective data management, security, and user engagement can have a significant impact on care quality and operational efficiency. Caregivers and patients rely largely on the platforms' seamless functionality for scheduling, communication, and access to critical information. The application mentioned in this thesis seeks to meet these requirements by integrating sophisticated web technologies.

The integration of several technologies into a single integrated application shows a modern approach to addressing the multifaceted issues of web development. This thesis investigates how these tools can be efficiently used to produce a web application that is useful, secure, scalable, and maintained. The primary objective is to improve the user experience and operational efficiency of caregiving platforms, which will directly contribute to better healthcare outcomes.

1.2 Problem Statement

In such a dynamically changing environment of web development as it is nowadays, the ability to be able to create efficient, safe, and user-friendly web applications is still a very serious challenge. The developers of web applications have to develop modular, reusable, and interactive components of web interfaces that will respond to the permanently changing requirements in terms of security, possibilities for the passage of huge volumes of data, and speed of operation of interfaces.

Secure user authentication and authorization have become one of the indispensable features of the modern web application. With increasing concerns over data privacy and cybersecurity, a stronger authentication system is demanded to let user data and the integrity of applications be well guarded. However, there are several such solutions available in the market, but none of them perfectly merge with the React-based ecosystem to provide flexible and secure identity management. Particularly, web developers need a solution that:

Provides comprehensive security features, including single sign-on (SSO), multi-factor authentication (MFA), and role-based access controls (RBAC). Integrates easily with modern frontend frameworks like React. Is easy to implement, customize, and manage. In this setting, the thesis will discuss the application development making use of Keycloak—an open-source identity and access management system—into a React application in order to provide secure authentication.

Besides the security concerns, efficient and user-friendly web components are equally important for the effective user engagement of the web applications and for the proper management of data. Therefore, the thesis focuses on the introduction of reusable web components with more details, such as what they are and how to develop them using React, TypeScript, and Redux.

User Management Features: Listings of users, detailed profiles for each user, and with the help of a simulated backend using json-server, it allows for editing or deletion of a user. Bookmark Feature: Allows a user to highlight favorite profiles and save a list for personal use. Integration with Kepler.gl: Enables advanced geospatial user data visualization, giving an engaging and informative experience in mapping. The core problems addressed in this thesis are:

Secure Authentication: How to make the authentication system strong enough in a web application that user data stays secure with Keycloak while ensuring a seamless user experience with single sign-on (SSO)? User Management Components: How can reusable web components be designed that enable effective management of user data and, at the same time, ensure safe interaction, whether in editing or deletion? Bookmark Feature: How can this feature be implemented so that users can add the content they like to their own list of favorites? Geospatial Visualization: How can Kepler.gl be effectively used with React to visualize user data on a map? Hence, this thesis is a pragmatic attempt to address these challenges by designing and implementing secure, reusable, and efficient web components within a React ecosystem. In doing so, the study contributes to the broader field of web development and sets a reference for developers looking to implement similar functionalities.

1.3 Objectives of the Study

The primary objective of this thesis is to design and implement secure, reusable, and efficient web components within a React ecosystem, utilizing modern technologies like TypeScript, Redux, and Keycloak. Specifically, the study aims to address the following objectives:

1. **Implement a Secure Authentication System**
    - Integrate Keycloak into the web application for robust authentication and authorization.
    - Ensure that the authentication system provides comprehensive security features such as single sign-on (SSO), multi-factor authentication (MFA), and role-based access control (RBAC).
    - Create reusable authentication components that seamlessly fit into the React ecosystem.
2. **Develop Reusable Web Components for User Management**
    - Design and implement a user management system that allows for listing, viewing, editing, and deleting user profiles.
    - Simulate a backend using the json-server library to facilitate dynamic data management during development and testing.
    - Ensure that these components are reusable, modular, and maintainable across different parts of the web application.
3. **Integrate Geospatial Visualization with Kepler.gl**
    - Utilize Kepler.gl to visualize user data on a map, providing an engaging and informative user experience.
    - Ensure smooth integration of Kepler.gl with the React application to present geographical data dynamically and efficiently.
4. **Evaluate the Performance, Security, and Usability of the Implemented Components**

- Conduct severe testing and evaluation to assess the performance and security of the authentication system.
- Measure the usability and effectiveness of the user management components and the bookmark feature.
- Analyze the geospatial visualization's impact on user engagement and application performance.

5. **Contribute to the Field of Web Development**

   - Provide practical solutions and best practices for implementing secure and reusable web components in modern web applications.
   - Serve as a reference model for developers and researchers interested in using React, TypeScript, Redux, and Keycloak.

1.4 Scope and Limitations

The scope of this study encompasses the design and implementation of secure, reusable, and efficient web components within a React ecosystem. The research focuses specifically on integrating Keycloak for secure authentication, developing user management components, creating a bookmark feature, and visualizing data with Kepler.gl. The following aspects define the boundaries of the study:

1. **Scope**

   - **Authentication System**: The study covers the integration of Keycloak for user authentication and authorization, ensuring comprehensive security features such as single sign-on (SSO), multi-factor authentication (MFA), and role-based access control (RBAC).
   - **User Management Components**: The thesis includes designing and implementing web components for user management, including listing, viewing, editing, and deleting user profiles.
   - **Geospatial Visualization with Kepler.gl**: The integration of Kepler.gl for visualizing user data on a map, providing an engaging and informative user experience.

2. **Limitations**

   - **Authentication Alternatives**: While Keycloak is used for secure authentication, the study does not explore alternative authentication systems in detail.

- **Backend Simulation**: The use of the json-server library provides a simulated backend for development and testing purposes, which may not fully represent real-world application backends.
- **Scalability Considerations**: The implemented web components are designed to be reusable and efficient, but the study does not deeply explore scalability challenges in high-traffic production environments.
- **User Management Depth**: The user management components are implemented with limited functionalities (e.g., listing, viewing, editing, and deleting profiles). More advanced features like role-based access control and batch processing are not included.
- **Geospatial Visualization Scope**: Kepler.gl is integrated for geospatial data visualization, but the visualization is limited to mapping user profiles without deeper geospatial analysis.

1.5 Significance of the Study

The significance of this study lies in its contribution to the field of web development by demonstrating practical methods for designing and implementing secure, reusable, and efficient web components using modern technologies like React, TypeScript, Redux, and Keycloak. The specific contributions of this study are as follows:

1. **Secure Authentication System**
   - The study demonstrates the effective integration of Keycloak for implementing a secure authentication system with comprehensive features such as single sign-on (SSO), multi-factor authentication (MFA), and role-based access control (RBAC).
   - It provides a reusable set of authentication components that can be seamlessly integrated into React-based applications.
2. **Reusable Web Components**
   - The study develops a suite of reusable web components for user management, including listing, viewing, editing, and deleting user profiles.
   - It contributes practical guidelines and best practices for building modular, maintainable, and scalable web components using React, TypeScript, and Redux.
3. **Geospatial Data Visualization**
   - The integration of Kepler.gl for geospatial data visualization introduces a novel approach to mapping user data in web applications.

- The study demonstrates how Kepler.gl can be used to enrich user interfaces with interactive and informative data visualizations.

4. **Evaluation of Performance, Security, and Usability**
   - The rigorous testing and evaluation conducted in this study provide valuable insights into the performance, security, and usability of modern web components.
   - The findings offer practical benchmarks and recommendations for developers building similar systems.

5. **Reference Model for Developers and Researchers**
   - The study serves as a reference model for developers and researchers interested in implementing secure and reusable web components using React, TypeScript, and Redux.
   - It contributes to the body of knowledge in web development by providing practical solutions and best practices for modern web applications.

1.6 Thesis Structure

This thesis is organized into the following chapters:

1. **Introduction** This chapter provides background information on web component design and implementation, the motivation for the study, the problem statement, objectives, scope, limitations, significance, and the structure of the thesis.

2. **Literature Review** This chapter reviews relevant literature on web technologies, web components, and related works. It discusses the evolution of web development, the role of React, TypeScript, Redux, and the importance of authentication systems like Keycloak in modern web applications.

3. **Methodology** This chapter presents the research design and methodology. It details the tools and technologies used, including React, TypeScript, Redux, Keycloak, json-server, and Kepler.gl, and explains the development process and testing methods.

4. **Design and Implementation of Web Components** This chapter describes the architecture, design, and implementation of the web components. It focuses on the secure authentication system with Keycloak, user management components, the bookmark feature, and geospatial visualization with Kepler.gl.

5. **Conclusion and Future Work** This chapter summarizes the key findings and contributions of the thesis, acknowledges the limitations of the study, and provides recommendations and suggestions for future research.

Chapter 2

Literature Review

2.1 Evolution of Web Technologies

The evolution of web technologies over the past few decades has transformed the landscape of software development. From static HTML pages to dynamic, data-driven applications, the journey has been marked by several key milestones:

- **Early Web (1990s):** The early web consisted of static HTML pages with minimal interactivity. Technologies like CGI and server-side scripting languages (Perl, PHP) enabled basic form processing.
- **Dynamic Web (2000s):** The rise of JavaScript, AJAX, and server-side frameworks (ASP.NET, JSP) brought about dynamic, interactive web applications.
- **Web 2.0 (the Late 2000s):** Web 2.0 emphasized user-generated content, social networking, and cloud computing. AJAX and RESTful services became standard.
- **Modern Web (2010s):** The adoption of HTML5, CSS3, and JavaScript frameworks like AngularJS, React, and Vue.js revolutionized front-end development.
- **Current Trends (2020s):** The current era is characterized by Progressive Web Apps (PWAs), serverless architectures, microservices, and advanced state management solutions like Redux.

2.2 Web Components: An Overview

Web components are reusable, encapsulated blocks of code that are used to build web applications. They consist of several key technologies:

- **Custom Elements:** Defines new HTML elements using the `HTMLElement` API.
- **Shadow DOM:** Provides encapsulation for styles and markup, preventing style leakage.
- **HTML Templates:** Allows for defining reusable HTML structures that can be cloned and inserted into the DOM.
- **ES Modules:** Enables modular and reusable JavaScript code.

These technologies provide the foundation for building modular, maintainable, and reusable web components. Frameworks like React, Angular, and Vue.js build upon these concepts to offer high-level abstractions and enhanced development capabilities.

2.3 React in Modern Web Development

React, developed by Facebook in 2013, is a JavaScript library for building user interfaces. Key features of React include:

- **Component-Based Architecture:** Promotes the use of reusable and composable components.
- **Virtual DOM:** Optimizes rendering by maintaining a lightweight representation of the actual DOM.
- **One-Way Data Binding:** Simplifies data flow and state management.
- **JSX:** A syntax extension that enables writing HTML-like code directly within JavaScript.

React has become one of the most popular frontend libraries due to its flexibility, performance, and extensive ecosystem. It is widely used for building dynamic single-page applications (SPAs) and user interfaces.

2.4 The Role of TypeScript and Redux

2.4.1 TypeScript

TypeScript, developed by Microsoft, is a superset of JavaScript that adds static typing and other features:

- **Static Typing:** Detects type errors at compile-time, reducing runtime bugs.

- **Interfaces and Generics:** Provides better code organization and reusability.
- **Enhanced Tooling:** Improves IDE features such as autocompletion, refactoring, and error checking.

Integrating TypeScript with React helps in building more maintainable and robust applications.

2.4.2 Redux

Redux is a predictable state container for JavaScript applications. It follows the Flux architecture pattern and provides:

- **Single Source of Truth:** The entire application state is stored in a single object.
- **State Immutability:** State is never mutated directly but replaced with a new object.
- **Predictable State Management:** Changes are made via pure functions called reducers.
- **Middleware Support:** Enables handling asynchronous actions and side effects.

Redux, when combined with React, provides a powerful mechanism for managing application state consistently and predictably.

2.5 Authentication in Web Applications

Authentication is crucial in modern web applications to ensure secure access to data and functionality. Key considerations include:

- **Authentication Mechanisms:**
  - **Session-Based Authentication:** Server stores session data; cookies identify clients.
  - **Token-Based Authentication:** ?? (JSON Web Tokens) are used for stateless authentication.
  - **OAuth 2.0:** Open standard for access delegation.
- **Authorization:** Role-based access control (RBAC) and attribute-based access control (ABAC) define user permissions.

- **Identity Management Systems:** Systems like Keycloak provide comprehensive identity and access management, including single sign-on (SSO), multi-factor authentication (MFA), and social login.

The integration of secure authentication systems like Keycloak into web applications is crucial for protecting user data and ensuring compliance with security standards.

Chapter 3

Methodology

3.1 Research Design

The methodology of this thesis is rooted in a practical, experimental approach to software development, focusing on the iterative design and implementation of web components within a controlled setting. This approach allows for the systematic investigation of how different technologies and designs affect the usability, functionality, and performance of web applications.

3.2 Frontend development

The term frontend describes the area of a software program that interacts directly with users, or the user-facing side. It includes the interface, organization, and features that users interact with.

Frontend development involves using various programming languages and technologies to create the visual and interactive elements of a software application.

As a result, the frontend development takes into account a number of factors, including the creation of the user interface, server connectivity, internal state and cache management, and client-side business logic.

The following technologies are frequently used in frontend development: HTML (HyperText Markup Language): This is the common markup language used to design web page structure. Cascading Style Sheets, or CSS, are used to style how web pages look. JavaScript: A computer language that allows for dynamic content updates, form validation, and animations on websites.

The React framework was utilized in the frontend development for this project, which was entirely web-based.

Website development relied on the construction of HTML code, one or more CSS style sheets, one or more Javascript scripts, often put straight into HTML to add dynamism to the page, and other components before the emergence of libraries and frameworks like React, Vue.js, and Angular [1].

These days, nearly all web apps are built using Javascript frameworks, which provide a full development environment and effectively and automatically integrate the layout declaration, the business logic, and the design.

3.2.1 Selection of Tools and Technologies

A critical aspect of this research was selecting appropriate tools and technologies that provide robust support for modern web development practices. The choices were made based on the popularity, community support, documentation quality, and suitability for enterprise-level applications [2].

3.2.2 React

For this project, the React library was chosen for front-end development.

Facebook launched the open-source React project, which has received a lot of support from the front-end developer community. Unlike other competitors, React is just a library and just intended for use in the creation of user interfaces, and not a framework [3].

To be sincere, several technologies have become vital in recent years due to their excellent effectiveness in setting standards for the creation and implementation of React applications.

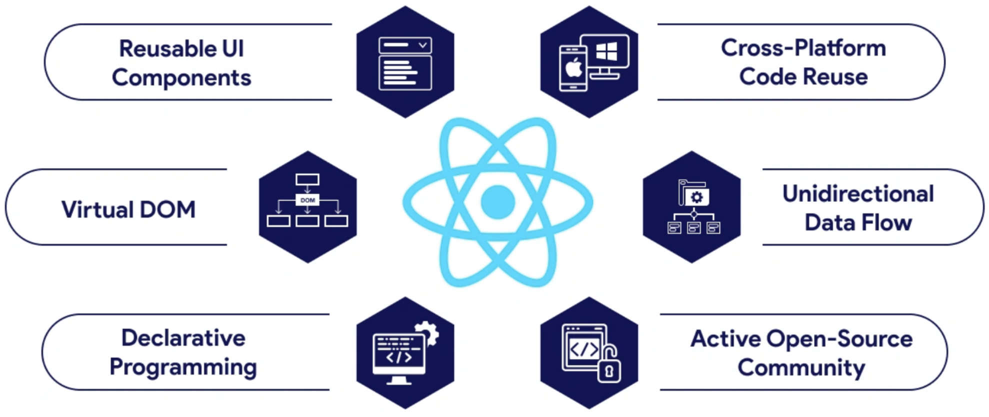


Figure 3.1: Major Benefits of React JS [4]

Among these tools, Redux stands out as the primary state management tool, and Webpack is notable for efficiently bundling static resources into two or three files, which will be further discussed later.

React adopts a component-based approach where each component, a fundamental unit of the user interface, integrates its visual elements, business logic, and internal state.

These components take in parameters, known as props, and output user interface elements. User interface declarations are made directly in JavaScript, specifically using JSX, which will be examined in greater detail subsequently. This approach eliminates the need for developers to write HTML or manually manage the DOM. Instead, they utilize “virtual” React components that exist in memory within the “virtual DOM,” allowing React to manage DOM updates autonomously and efficiently, typically triggered by changes in the component’s state [5].

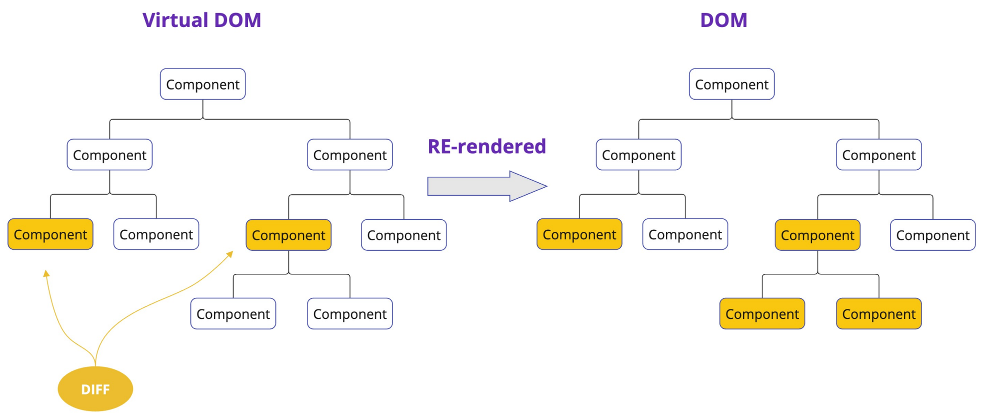


Figure 3.2: React Virtual Dom

3.2.3 JSX

JSX, or JavaScript XML, is a syntax extension for JavaScript used predominantly in React to describe the appearance of the user interface. It allows developers to write HTML-like code alongside JavaScript in the same file, blending the presentation logic directly with JavaScript functions. This syntax, which resembles HTML, makes it intuitive for those familiar with HTML to adopt and use in React applications [6].

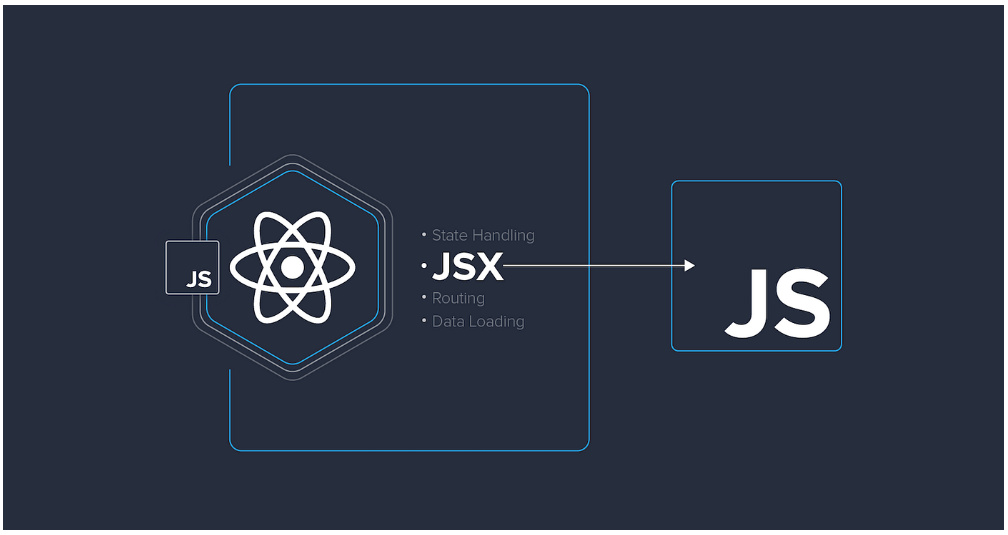


Figure 3.3: JSX role inside React

3.2.4 Webpack

Modern online development relies heavily on Webpack, a powerful and adaptable module bundler for JavaScript applications, particularly in frameworks such as React.

Webpack takes modules with dependencies and generates static assets representing those modules. It starts from one or more entry points and then transforms and bundles the required modules and assets into fewer, usually only one or two, files. This process is customizable with the help of loaders and plugins:

- Loaders in Webpack transform the files into modules as they are added to your application's dependency graph. They can convert SASS to CSS, TypeScript to JavaScript, and include Babel transpilation to convert ES6 code down to ES5 for broader browser compatibility.

- Plugins are the backbone of Webpack's functionality, offering a rich feature set that is used to perform a wider range of tasks like bundle optimization, asset management, and environment variable injection. [7]

Webpack's Role in a React Application

In the React application, Webpack is primarily used for bundling and serving assets efficiently.

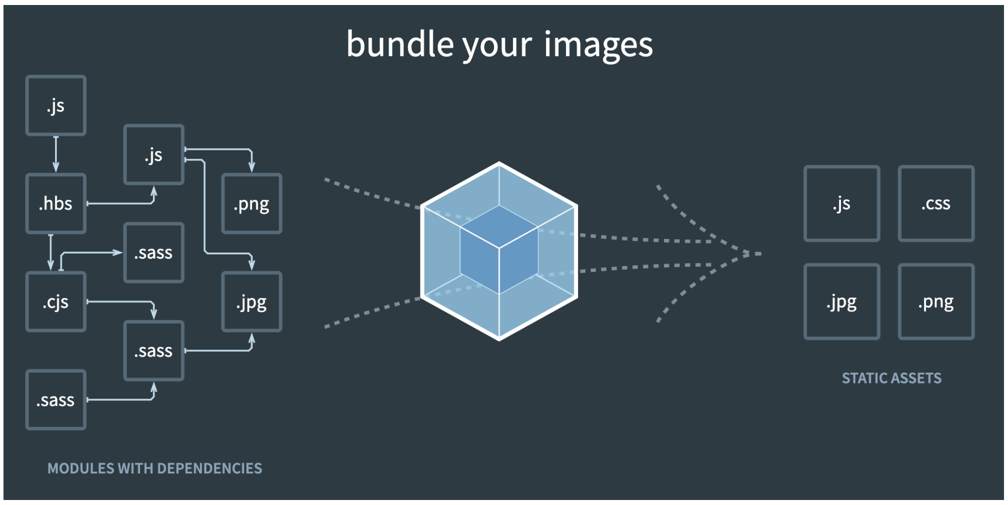


Figure 3.4: Webpack flowchart

Single Page Applications:

One of Webpack's key contributions to React development is its ability to bundle complex multi-file applications into a single-page application (SPA). This bundling not only simplifies deployment but also improves performance by reducing the number of server requests during navigation.

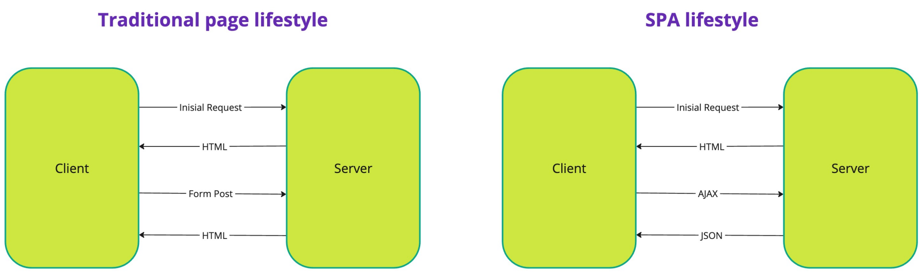


Figure 3.5: SPA and traditional page lifecycles comparison [8]

Optimization:

Webpack optimizes the loading of your application by reducing the size of your JavaScript bundle. It does this through techniques like code splitting, tree shaking (removing unused code), and minifying code. This results in faster load times and a better user experience. React code is often written using JSX and modern JavaScript (ES6+), which aren't supported in all browsers. Webpack uses Babel (through loaders) to transpile JSX into JavaScript that browsers can understand and ES6+ into ES5.

Development Efficiency:

Webpack enhances developer productivity by providing features like hot module replacement (HMR). HMR updates the application on the fly without needing a full reload when code changes, which is particularly useful in React development as it preserves the state of the application.

Asset Management:

Besides JavaScript, React applications often involve other assets such as images, fonts, and CSS. Webpack can manage all these assets, allowing them to be bundled as part of the dependency graph of the application. It can also inline images as data URLs, dynamically load images as needed, and bundle CSS either in JavaScript or into separate CSS files.

Customization:

Webpack's extensive configuration options allow React developers to tailor the build process to their specific needs, optimizing performance, and enhancing functionality.

To put it simply, Webpack is essential to current React programming. It simplifies multi-page architectures into single-page apps while also optimizing load times and consolidating resources. In intricate web applications, this feature greatly improves resource management and user experience.

3.2.5 Entry Point of the Application: App.js

The React project is defined as a SPA. SPA stands for Single Page Application. Actually, the server returns an index.html file to the base path "/" that contains the whole program.

While some static resources are included in the index.html page, the logic and CSS style are loaded dynamically from the bundle files. There are scenes and components within the project.

Repetitively, a scene includes other scenes or parts. A scene is an element used just once inside the application that is typically related to the global state, whereas a component is a reusable user interface element that is typically not tied to the global state of the application. The app.js file serves as the application's entry point and is used to initialize the router and instantiate the state manager redux.

3.2.6 React Router

Routing in single-page applications (SPAs) is crucial for managing navigation and rendering components without full page reloads. React Router v6, the latest iteration of the popular library, is specifically tailored for React applications, providing dynamic routing capabilities that are both powerful and easy to manage. This section discusses the adoption of React Router v6 in the application, highlighting its necessity and the enhancements it offers over previous versions.

React Router v6 was chosen for its significant improvements in performance, flexibility, and developer experience. It introduces several new features and optimizations that streamline routing in React applications:

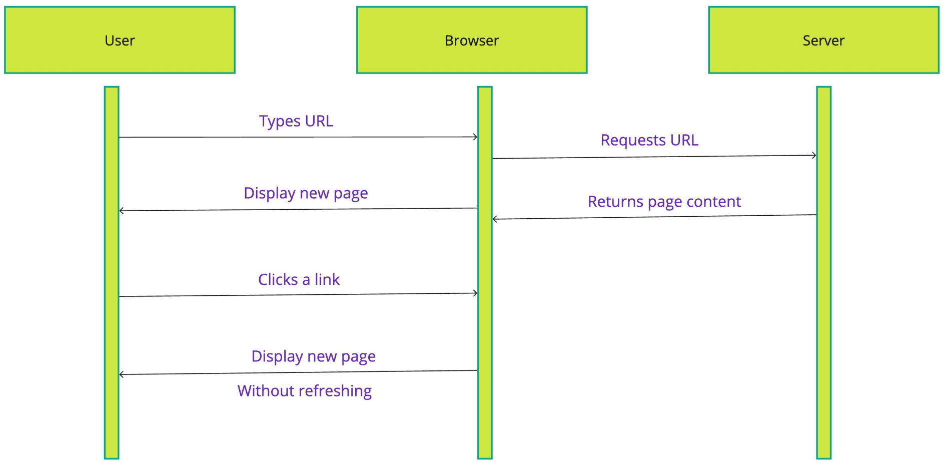


Figure 3.6: How React router works

Simplified Configuration:

Simplified Configuration: React Router v6 simplifies route configuration, making it more intuitive and declarative. Routes are now naturally nested, matching the UI structure, which aligns closely with React's component-based architecture.

Built-in Lazy Loading:

The library supports React's native lazy loading mechanism, allowing developers to split their application code into chunks that are only loaded when needed. This reduces the initial load time, enhancing the performance and user experience.

Hooks API:

React Router v6 expands its hooks API, offering more granular hooks like useNavigate, useParams, and useLocation. These hooks provide a more React-centric way of building navigation and interaction, adhering to the modern React programming style which favors hooks over higher-order components and render props.

Automatic Route Ranking:

The library automatically ranks routes to decide which route is the best match based on specificity. This means less manual error handling and override setup for developers, streamlining route management.

Enhanced Code Simplicity and Performance:

With smaller bundle sizes and a more efficient approach to routing, React Router v6 not only speeds up application performance but also simplifies the codebase, making it easier to maintain and scale.

Security is a critical aspect of any modern application, particularly when it comes to managing access to certain routes or sections that contain sensitive information or functionalities. These sections are often referred to as "private routes"[9] and are designed to be accessible only to authenticated and authorized users. The importance of securing these routes cannot be overstated, as they often handle personal data or critical business functions that, if compromised, could lead to significant privacy breaches or operational disruptions.

In a well-secured application, any attempt by an unauthorized user to access a private route should lead to immediate redirection to a login page. This safeguard is essential for enforcing access controls and maintaining the integrity and confidentiality of the application's sensitive areas. Implementing such redirection not only blocks unauthorized access but also cues the user to authenticate, potentially streamlining the user experience by guiding them towards obtaining the appropriate access permissions.

This redirection mechanism is crucial not just for security purposes but also for user guidance. It prevents exposure of sensitive interfaces and data to unauthorized users and helps legitimate users access their desired resources by prompting them to log in. Proper implementation of such security measures requires a robust authentication system and carefully designed logic to handle route protection, ensuring that all private routes are shielded from unauthorized access effectively.

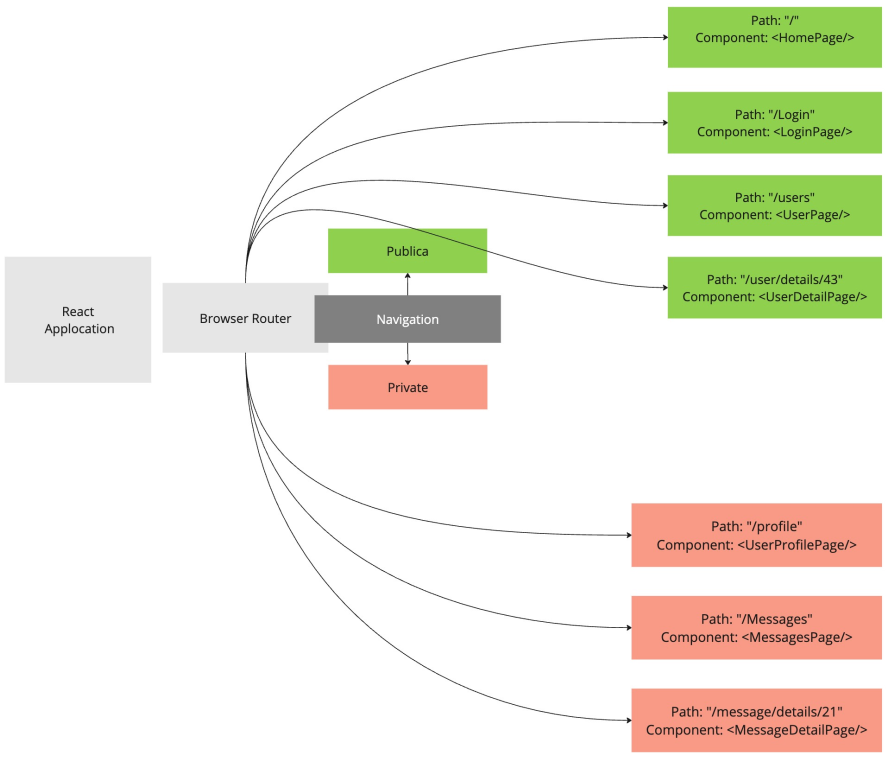


Figure 3.7: The schema of the routing structure [10]

3.2.7 Material UI

Material-UI was selected as the foundational library for the graphical components of this project due to its comprehensive incorporation of best practices in performance and user experience, which align closely with the principles of Google's Material Design guidelines.

Opting for Material-UI allows for the utilization of pre-designed, high-quality components rather than building from scratch with basic HTML elements. This choice is particularly advantageous for projects where graphic customization is not the primary focus. By leveraging the extensive work done by a community of skilled frontend developers, the project benefits from years of expertise and ongoing improvements without the overhead of initial development.

Material-UI offers a wide range of components, from simple utilities like the Paper, which acts as a basic container, to more complex elements like Chips, used for listing tags or attributes and customizable with options for editing or removal. It also includes Tab bars, which facilitate navigation across different views within the same interface, and Buttons that can function both as clickable buttons or links.

A key feature of Material-UI is the Typography component, which organizes text elements from headers (h1, h2, h3, etc.) to captions and body text, ensuring text across the application maintains consistent styling and readability.

From a user experience perspective, many of these components include default animations for actions such as clicks or focus, enhancing user interaction with tactile feedback. Graphically, Material-UI supports a powerful theming system, allowing developers to define high-level variables like primary and secondary colors. Components within the application then automatically adjust their styles to match these theme settings.

Customization extends through the use of component properties (props). For instance, assigning the 'primary' color prop to a button applies the theme's primary color, and similarly, animations and other interactive elements adapt to align with the defined theme settings.

The creation of a custom theme object, rather than using the default provided by Material-UI, was a deliberate choice to align the application's visual elements with a specific design intent. The primary and secondary colors were chosen using coolors.co, a tool that generates color palettes optimized for user interface and experience.

Material-UI's theme object offers extensive control over the appearance and sizing of components, from interactive elements to textual presentations. Additionally, the palette type property within each theme specifies whether to adopt a light or dark mode, influencing the library to set appropriate contrast text colors that ensure legibility against varying background conditions.

the adoption of Material-UI not only streamlines development with its rich set of ready-to-use components but also ensures that the application maintains a high standard of design consistency and user accessibility. This integration reflects a strategic choice to harness proven technologies that enhance the application's functionality and aesthetic appeal while supporting efficient development practices.

3.2.8 Redux - the state manager

Redux is a model and library for using "actions" or events to manage and update the state of an application. With rules ensuring that the state can only be modified in a manner that is predictable, it provides a single repository for state that must be used throughout the entire program.

Redux facilitates the management of "global" state, or state required by various components of your application.

It is simpler for you to understand when, where, why, and how your application's state is being modified, as well as how your application logic will respond when those changes take place, thanks to Redux's patterns and tools. Redux lets you write predictable and testable code so you can be more confident that your application will function as intended. Redux is a useful tool for managing shared states, but it has drawbacks like every other technology. Both the concepts to understand and the code to write are increasing. It also requires you to abide by certain limitations and adds some indirection to your code. There is a trade-off between productivity over the long run and short term.

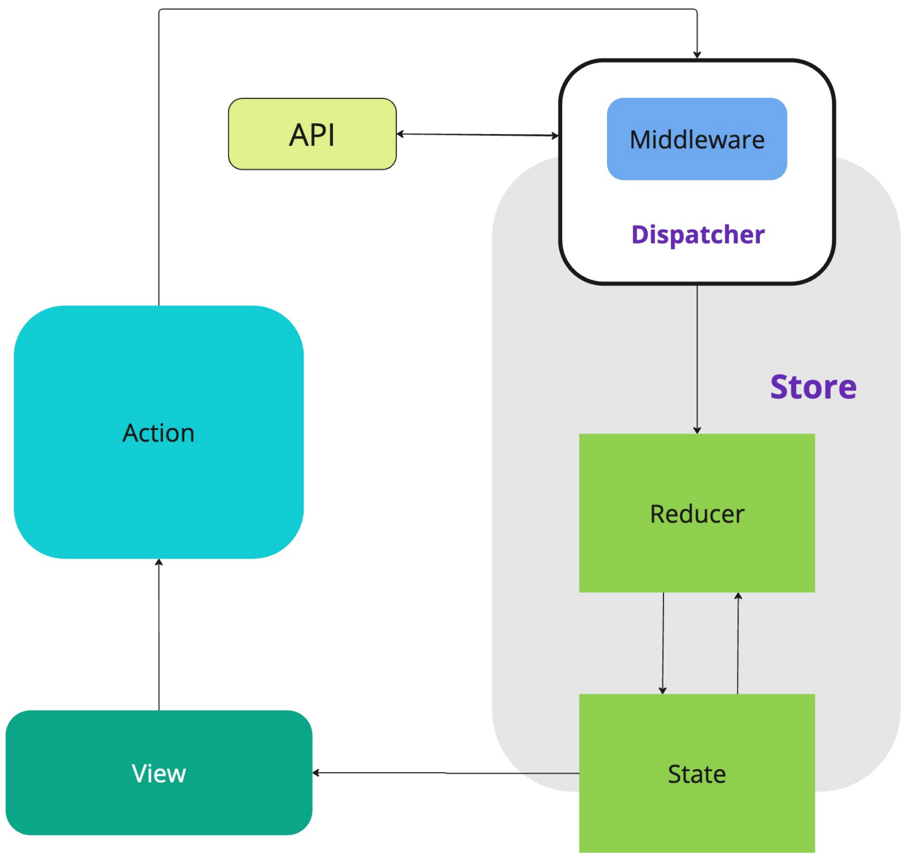


Figure 3.8: Redux data flow

Redux works best when You have a lot of application states that are required throughout the app. Over time, the app state is updated often. It could have complicated logic to update that state. The application may have a huge or medium-sized codebase with numerous developers working on it [11].

3.2.9 React-redux

React-Redux provides the critical linkage between the Redux state management library and React components. At the heart of this integration is the React Context API, which React-Redux utilizes to enable publish/subscribe interactions between the Redux store and React components. This setup allows for efficient data flow and state management across the application.

The Provider component, a fundamental part of React-Redux, acts as a wrapper at the root of your application. It accepts the Redux store as a prop, which is created using Redux's createStore function. The primary role of the Provider is to place the Redux store into the React Context, making it accessible to any nested components that might need to subscribe to the store updates.

```
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import rootReducer from './reducers';
import App from './App';

const store = createStore(rootReducer);

const RootComponent = () => (
  <Provider store={store}>
    <App />
  </Provider>
);
```

3.2.10 Redux Toolkit

Redux Toolkit (RTK) represents the official, opinionated, batteries-included toolset for efficient Redux development. It is designed to simplify the process of setting up and working with Redux in a React application. RTK addresses common issues such as verbosity, complexity, and maintainability of Redux with a more straightforward and powerful approach to state management.

Redux Toolkit simplifies Redux application development and encourages best practices. It reduces boilerplate code significantly, which not only makes the code more manageable but also helps prevent common mistakes.

RTK includes performance optimizations out of the box. It uses Immer internally to handle immutable state updates more efficiently, which helps avoid common bugs associated with mutating state directly.

With utilities like configureStore, createSlice, and createAsyncThunk, Redux Toolkit provides more straightforward APIs that consolidate various setup steps into single, coherent operations. This makes the setup and operation of the Redux store easier and more intuitive.

This utility simplifies the store setup process with sensible defaults. It automatically sets up the store with recommended middleware for a better development experience, such as Redux Thunk for asynchronous actions and Redux DevTools integration.

A function that accepts an initial state, an object of reducer functions, and a "slice name", and automatically generates action creators and action types that correspond to the reducers and state. This reduces the redundancy of declaring action types and action creators separately.

```
import { createSlice } from '@reduxjs/toolkit';

const todoSlice = createSlice({
  name: 'todos',
  initialState: [],
  reducers: {
    addTodo: (state, action) => {
      state.push({ id: action.payload.id, text: action.payload.text,
    completed: false });
    },
    toggleTodo: (state, action) => {
      const todo = state.find(todo => todo.id === action.payload);
      if (todo) {
        todo.completed = !todo.completed;
      }
    }
  }
});

export const { addTodo, toggleTodo } = todoSlice.actions;
export default todoSlice.reducer;
```

3.2.11 RTK Query: Data Fetching and Caching

RTK Query, a powerful data fetching and caching tool included in Redux Toolkit, abstracts the handling of data fetching, caching, synchronization, and error handling, providing a seamless integration for server-state management in React applications.

Some features of RTK Query:

Auto-generated Hooks:

RTK Query generates custom React hooks for each endpoint query or mutation, simplifying the integration with the UI.

aching and Invalidation:

It automatically provides caching logic, which minimizes the number of requests needed. It also supports tag-based invalidation to refresh data as required.

Polling and Prefetching:

Supports features like automatic polling and prefetching, making it easier to keep the application data up-to-date without manual intervention.

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/
    react';

const api = createApi({
  reducerPath: 'api',
  baseQuery: fetchBaseQuery({ baseUrl: '/api' }),
  endpoints: (builder) => ({
    getTodos: builder.query({
      query: () => 'todos',
      providesTags: ['Todo'],
    }),
    addTodo: builder.mutation({
      query: (todo) => ({
        url: 'todos',
        method: 'POST',
        body: todo,
      }),
      invalidatesTags: ['Todo'],
    }),
  }),
});

export const { useGetTodosQuery, useAddTodoMutation } = api;
```

Redux Toolkit and RTK Query significantly enhance the development experience and capabilities within React applications. By reducing boilerplate, streamlining state management practices, and offering powerful data fetching capabilities, these tools allow developers to focus more on building features and less on configuring and maintaining state management and network logic. They represent essential advancements in the modern React development ecosystem, promoting efficiency, maintainability, and scalability.

3.2.12 TypeScript

TypeScript has become increasingly popular in the development of modern web applications, especially in complex projects where scalability, maintainability, and developer productivity are key. TypeScript is a free and open-source high-level programming language developed by Microsoft that adds static typing with optional type annotations to JavaScript. By integrating TypeScript with React, developers can significantly enhance code quality and readability, while also leveraging TypeScript's powerful type-checking capabilities to catch errors at compile time rather than at runtime.

TypeScript introduces static typing to JavaScript, allowing developers to specify types for variables, function parameters, and returned values. This type enforcement helps prevent many common bugs that can occur in dynamically typed languages like JavaScript, particularly those related to unexpected data types.

TypeScript provides better tooling at development time with features like autocompletion, type inference, and more descriptive error messages. This leads to more efficient coding and debugging processes.

For large-scale projects, TypeScript's modular architecture and type system make the codebase easier to manage and scale. Developers can more easily understand and work with the code, reducing the risk of introducing bugs when making changes or adding new features.

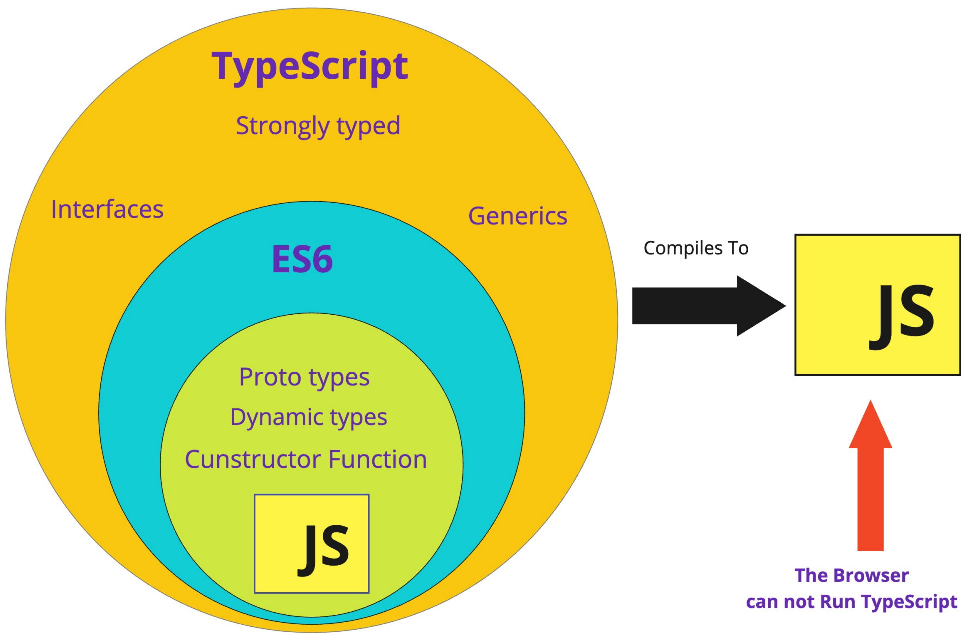


Figure 3.9: How TypeScript works

3.2.13 What is JWT Token

JWT or JSON Web Tokens are encrypted tokens that can be used to securely transmit information between client and server as a JSON object. This is the backbone of token-based authentication and is developed to transfer data securely and prevent any data theft or badgering. JWT is segregated into three distinct parts, with each separated from one other via a full stop. These three parts of the JWT are called as Header, Payload, and Signature [12].

Header

The header is the initial segment of a JSON Web Token (JWT). It primarily specifies the cryptographic algorithms used to secure the token. Typically, the header includes two essential pieces of information: the type of token, which is JWT, and the algorithm used for signing, such as HMAC SHA256 or RSA. This information is crucial as it dictates how the JWT should be verified and handled by the server. The header is JSON formatted and then encoded using Base64URL to ensure safe transmission over network protocols. An example of a header using the HS256 algorithm is:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

This JSON structure is Base64URL encoded, forming the first part of the JWT, and outlines the security process to be employed in verifying the token.

Payload

The payload of a JWT contains the actual data meant to be conveyed, which typically includes user-specific details and additional metadata necessary for the application's functionality. Common fields in the payload might include the user ID, email address, issuer's name, and the token's issuance time. This part of the JWT is also JSON formatted and then encoded using Base64URL, ensuring the data remains intact and tamper-proof during transmission. An example of a typical JWT payload might look like this:

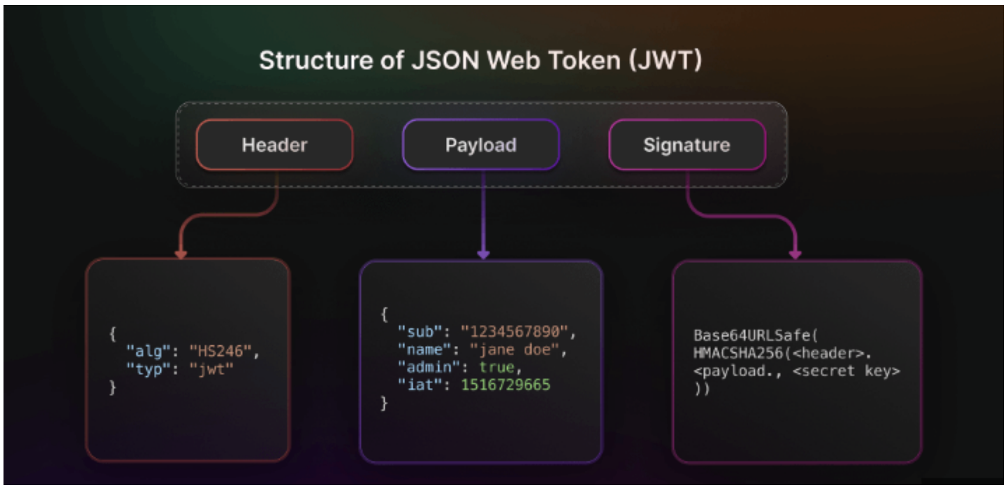
```
{
  "id": "1",
  "email": "xyz@abc.com",
  "issuer": "company name",
  "iat": 2321442213
}
```

It's important to note that sensitive information should not be stored in the payload unless it is encrypted, as Base64URL is an encoding technique, not an encryption method, and can be easily decoded.

Signature

The signature is the final component of a JWT, vital for ensuring the token's integrity and authenticity. It is generated by concatenating the encoded header, the encoded payload, and a secret or private key using the algorithm specified in the header. The signature serves to verify that the JWT has not been altered after it was issued and, in scenarios involving a private/public key pair, to authenticate the identity of the token's sender. The process to create the signature using the HMAC SHA256 algorithm can be illustrated as follows:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```



```
{
  "alg": "HS246",
  "typ": "jwt"
}
```

```
{
  "sub": "1234567890",
  "name": "jane doe",
  "admin": true,
  "iat": 1516729665
}
```

```
Base64URLSafe(
HMACSHA256(<header>.
<payload., <secret key>
))
```

Figure 3.10: Structure of JSON web Token [13]

This cryptographic process ensures that any alterations made to the header or payload after the token's generation will invalidate the signature, thereby alerting the receiving party to potential tampering.

JSON Web Tokens (JWTs) are structured in a compact format, comprising three distinct parts—header, payload, and signature—that together facilitate secure means of transmitting data and assertions between parties. Each segment of the JWT serves a specific purpose, from declaring the type and method of security used, to carrying user or system-specific information, and finally ensuring the integrity and authenticity of the token through its signature. Proper understanding and implementation of JWTs are critical in securing modern web applications, maintaining trust, and ensuring that communications between client and server remain secure and reliable.

Cryptography techniques used by JWT

JSON Web Tokens (JWTs) are not encrypted in their entirety but are instead encoded using the Base64 format. This encoding serves to compact the data into a URL-safe string, which simplifies the transmission of JWTs across different systems. It's important to note that Base64 encoding is not a security measure—it merely ensures that the token's structure remains intact during transport.

The contents within a JWT, including its header, payload, and signature, are thus easily readable by anyone who intercepts or is given the token. However, despite this transparency, the integrity and authenticity of the JWT remain protected.

Alterations to the token can only be validated through a cryptographic key known only to the issuer and, in some cases, the recipient. This key can be a shared secret using HMAC algorithms or a public/private key pair using RSA or ECDSA for added security.

The main purpose of JWT encoding is to efficiently handle user-specific information that the application frequently accesses, such as user permissions or session identifiers. However, to prevent unauthorized access to the sensitive data encoded within the JWT, additional precautions are necessary. While encoding simplifies data handling, it does not secure it against unauthorized viewing.

Regarding encryption, the signature part of the JWT, which is crucial for verifying the token's integrity, can be generated using various cryptographic algorithms. For instance, a commonly used RSA algorithm encrypts data with a public key, ensuring that only the holder of the corresponding private key can decrypt and thus verify the contents. This asymmetric encryption method is particularly useful in scenarios where enhanced security measures are required.

JWT algorithms offer great flexibility, allowing for the use of different hashing techniques. For example, a JWT might employ the HS256 algorithm, which utilizes HMAC to hash both the header and payload. The signature is then generated by hashing these two hashed parts together, potentially using a different algorithm, to further secure the token against tampering. This dual-layer hashing mechanism ensures the persistence of token contents and protects them from unauthorized modifications.

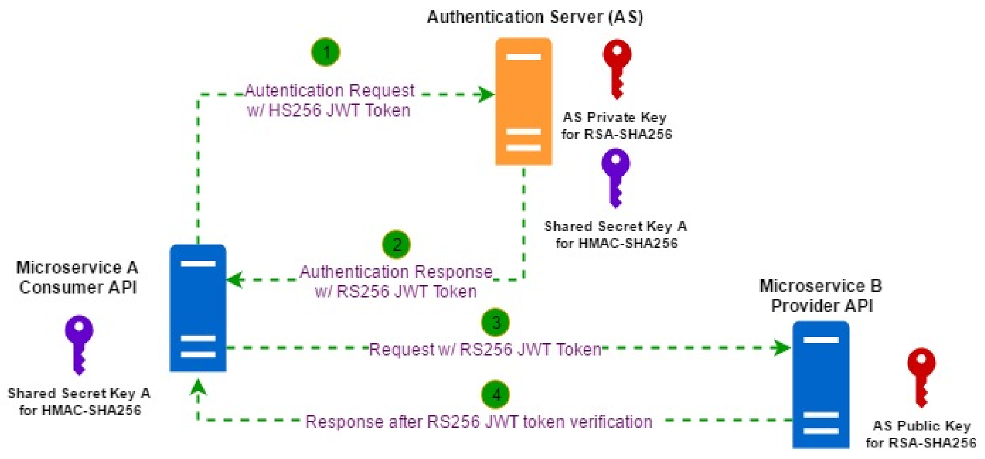

Authentication Server (AS)
1
Autentication Request w/ HS256 JWT Token
AS Private Key for RSA-SHA256
Shared Secret Key A for HMAC-SHA256
2
Authentication Response w/ RS256 JWT Token
Microservice A Consumer API
Shared Secret Key A for HMAC-SHA256
Microservice B Provider API
3
Request w/ RS256 JWT Token
4
Response after RS256 JWT token verification
AS Public Key for RSA-SHA256

Figure 3.11: Cryptography techniques used by JWT[14]

Session Based Authentication

In session-based authentication, when a user logs in, the server initiates a session and generates a session ID, which is subsequently stored as a cookie in the user's browser. As long as the user remains logged in, this cookie is automatically included with each subsequent request made to the server. The server uses this session ID to retrieve the session data from its memory or a session store, thereby confirming the user's identity. It then processes the request and delivers a response that is consistent with the user's state and permissions. This mechanism ensures that the user's state is preserved across multiple interactions with the application during the session, providing a seamless and secure user experience.

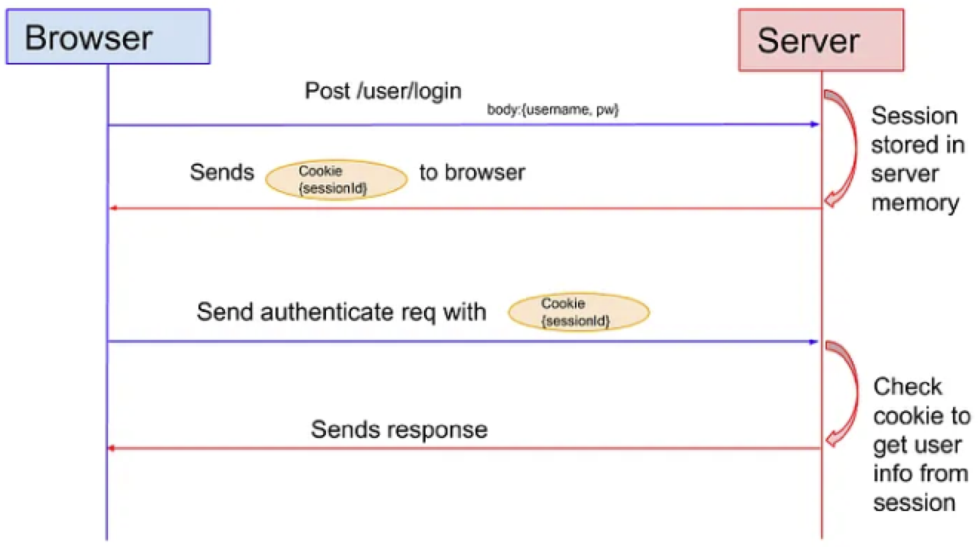


Figure 3.12: Session Based Authentication Flow [15]

Token Based Authentication

Many web applications have adopted JSON Web Tokens (JWT) as an alternative to traditional session-based authentication. In a token-based architecture, the server generates a JWT using a cryptographic secret and transmits it to the client. Typically, the client stores this JWT in local storage and includes it in the Authorization header of every subsequent request. Upon receiving a request, the server validates the JWT to ensure its integrity and authenticity before processing the request. If the token is valid, the server proceeds to fulfill the request and respond accordingly. This method not only streamlines the authentication process but also enhances the application's security by leveraging the self-contained nature of JWTs, which encapsulate the user's identity and claims.

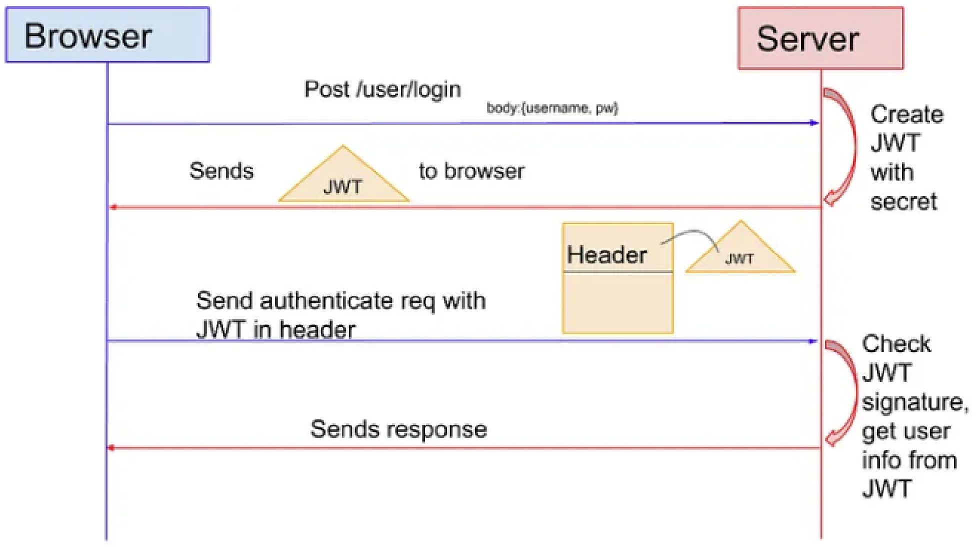


Figure 3.13: Token Based Authentication Flow [15]

Invalidating issued JSON Web Tokens (JWTs) upon logout presents a significant challenge, as JWTs are inherently designed to be stateless, meaning their validity isn't directly managed by the server. Stored either on the client-side or within cookies, JWTs cannot be easily revoked at the administrator's discretion since their validity isn't tracked server-side. This constraint complicates the process of securely managing session termination and can potentially create security risks. To address this issue, the following techniques are commonly employed:

Short-lived Tokens: One effective strategy is to issue short-lived tokens, which have a brief expiration time. This approach limits the duration for which a token is valid, thereby reducing the window of opportunity for unauthorized use should the token be compromised. After expiration, the user is required to re-authenticate, obtaining a new token to continue their session. This method enhances security but can detract from user experience by requiring frequent logins. Additionally, it poses a risk that a user's session might expire in the midst of critical activities, such as financial transactions, potentially disrupting their completion.

Blacklisting: Another method involves blacklisting or maintaining a list of invalidated tokens. When a token is no longer valid—due to logout or other reasons—it is added to a blacklist. This list is stored server-side, often within a database, and is consulted with each transaction to ensure no blacklisted token is accepted. While this approach effectively allows for the invalidation of specific tokens.

Use of short lived token

Short-lived tokens, despite presenting challenges due to their frequent expiration and subsequent user re-authentication requirements, are utilized for their enhanced security benefits. To mitigate the inconvenience of frequent logins, these tokens are often paired with a longer-term credential known as a refresh token. When a short-lived token expires, the validity of the long-lived refresh token is verified, and if valid, a new short-lived token is issued and securely attached to the cookies. This process eliminates the need for users to manually re-login each time their access token expires. The primary rationale behind this strategy is to enhance security; if a token were to be compromised, the limited validity period of short-lived tokens restricts the time frame in which an attacker could exploit it, thus significantly reducing potential damage and enhancing the overall security posture against attacks like Cross-Site Request Forgery (CSRF) or other types of unauthorized access attempts.

Refresh Token

If somehow the JWT token lands in the wrong hands it would result in an identity problem as the attacker would be able to access any private data of the original user. To curb this issue JWT tokens are generated with short expiry. This short expiration time ensures that even if the token lands in the wrong hands, then the attacker only has a small amount of time to use that token which might not be too much to fetch any useful data from the server. But on the contrary, this would result in a tedious and cumbersome process for the original user to login in again and again on the server and generate a fresh token. This problem is then solved using a pair of JWT tokens one of which is short-lived while the other is a long-lived token. Whenever the short-lived token expires this long-lived token also known as the refresh token is verified and a new JWT token is generated post successful verification. The process of creation of this refresh token is similar to that of the original token with the prime difference that it has a longer expiration time. This token must be stored in HTTP only cookies as it needs more security as a person possessing this token can generate an endless amount of JWT tokens throughout the lifetime of this token. This token should only regenerate a new JWT token if the request possesses an expired JWT token. If no JWT token is provided in the request then this throughout the lifetime of this token. This token should only regenerate a new JWT token if the request possesses an expired JWT token.If no JWT token is provided in the request then this part is skipped and an unauthenticated response is returned back to the user.

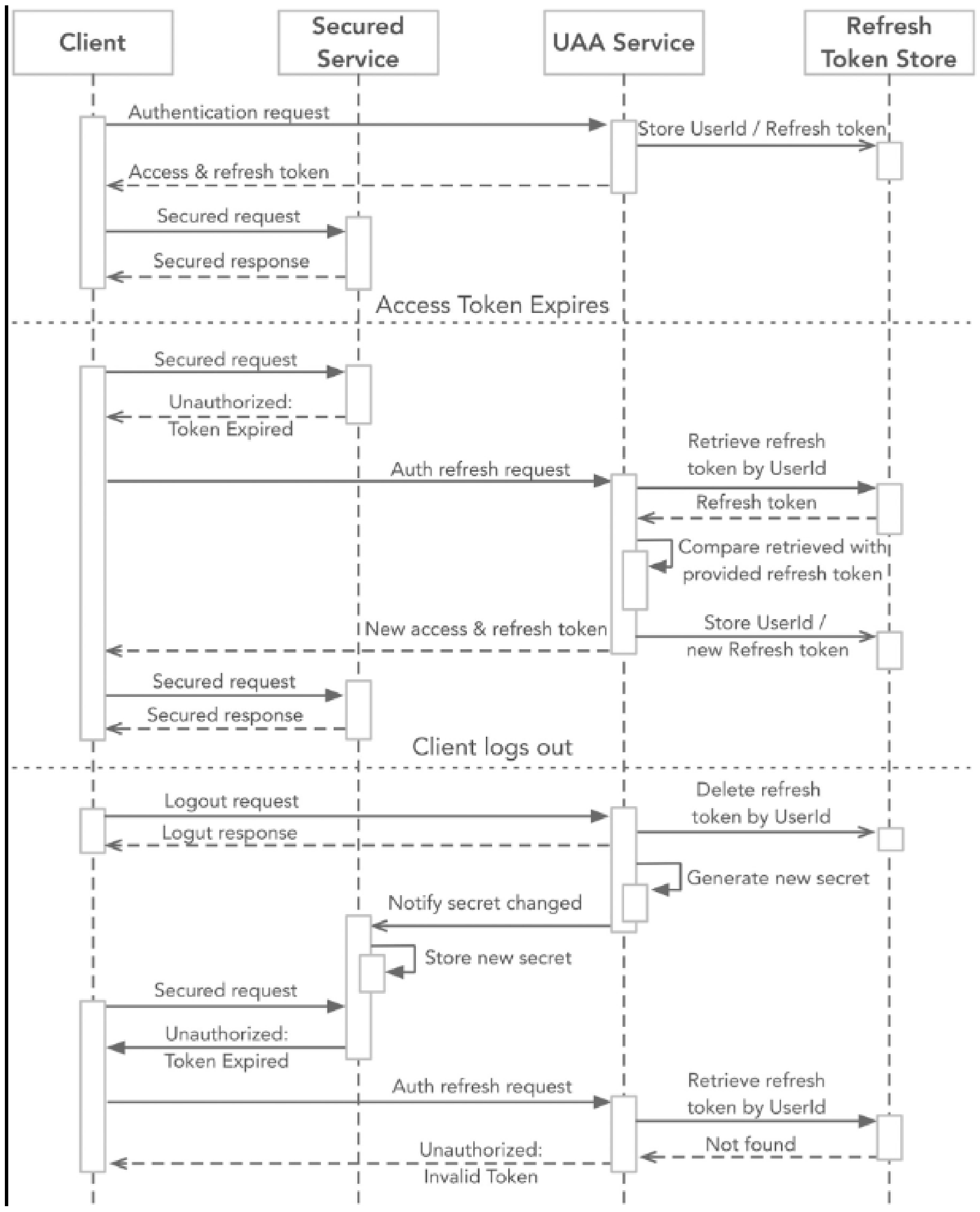


Figure 3.14: JWT Authentication and Token Lifecycle Management Flow[16]

CSRF Token

Cross-Site Request Forgery (CSRF), also known as a one-click attack or session riding, is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It exploits the trust that a site has in a user's browser, causing the user's browser to perform an unwanted action on a trusted site when the user is authenticated.

CSRF attacks can have varying impacts, depending on the specific vulnerabilities exploited and the level of access of the victim. The Open Web Application Security Project (OWASP) has introduced CSRF Guard as a countermeasure to protect against these attacks. Developed under the leadership of Eric Sheridan, CSRF Guard is a server-side tool that implements a version of the synchronizer token pattern, which helps mitigate the risk associated with CSRF by associating a secret, unpredictable token with each user session.

CSRF Guard has gained recognition as a robust defense mechanism against CSRF attacks. Studies and experiences reported by various researchers confirm its efficacy in securing web applications from such threats. However, it is crucial for developers to implement CSRF Guard without compromising the overall information security objectives of the application.

1. CSRF Guard Mechanism: CSRF Guard uses a synchronizer token pattern to validate user requests and protect against CSRF attacks. This mechanism involves injecting a unique security token into each session and verifying it with each request, aiming to ensure that requests are legitimate and originate from the authenticated user.

2. Vulnerabilities and Limitations: CSRF Guard primarily defends against CSRF attacks but can be bypassed through other vulnerabilities such as cross-site scripting (XSS) and session hijacking. The protection CSRF Guard offers is limited to environments where session identifiers are secure and not compromised. The tool is most effective on specific server environments (like Tomcat) and might not provide adequate security on others (e.g., IIS).

3. Security Scenarios: Various web application types, from content management systems to web servers like Apache, have different levels of susceptibility to CSRF attacks. The paper discusses several scenarios where CSRF Guard is beneficial, as well as somewhere its use might be inadequate or overkill, depending on the application's architecture and security requirements.

4. Recommendations: Web developers should carefully select which parts of their applications require CSRF protection based on sensitivity and functionality. Developers should ensure that CSRF Guard configurations do not compromise application usability or navigation[17].

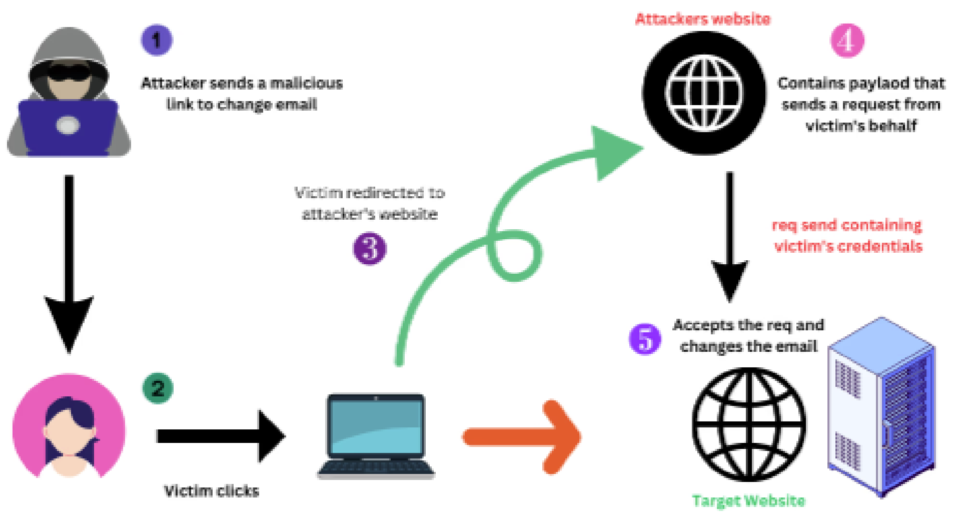

1
Attacker sends a malicious link to change email
2
Victim clicks
Victim redirected to attacker's website
3
Attackers website
4
Contains paylaod that sends a request from victim's behalf
req send containing victim's credentials
5
Accepts the req and changes the email
Target Website

Figure 3.15: CSRF in Action[18]

3.2.14 Keycloak

Keycloak is an open-source identity and access management tool with a focus on modern applications such as single-page applications, mobile applications, and REST APIs.

In 2014, the project got underway. Since then, it has developed into an excellent open-source project with a strong community. It is employed by both small and large businesses.

In addition to many other features, Keycloak offers fully configurable login screens, password recovery, and acceptance of terms. Without any coding knowledge at all, you can effortlessly incorporate all of Keycloak's functionality into your application. By giving Keycloak control over user authentication, you can stop worrying about password storage security and other related issues. Without altering the program, you can enable two-factor authentication. Because your application can only know the tokens that Keycloak has provided and cannot access user credentials in this scenario, it also improves the security of your application.

Keycloak offers robust session management features together with single sign-on. It involves allowing consumers to access several apps with just a single authentication.

Refresh Tokens

Keycloak implements the OAuth 2.0 specification, which includes the use of refresh tokens to maintain user sessions without requiring the user to repeatedly enter their credentials.

1. Token Granting: When a user successfully authenticates, Keycloak issues an access token and a refresh token. The access token is short-lived and used to access the protected resources. The refresh token lasts longer and is used to obtain new access tokens when the original expires. 2. Using Refresh Tokens: When the access token expires, the application can request a new access token by submitting the refresh token to Keycloak's token endpoint. If the refresh token is valid and has not been revoked, Keycloak issues a new access token (and possibly a new refresh token). 3. Security Considerations: Refresh tokens are particularly sensitive because they can remain valid for a long period. Keycloak provides mechanisms to revoke refresh tokens when they are suspected of being compromised. Moreover, applications should use secure transport (HTTPS) to prevent token interception, and tokens should be stored securely to prevent unauthorized access.

Handling CSRF Attacks

Keycloak mitigates CSRF attacks primarily through the use of anti-CSRF tokens, also known as state tokens or one-time use tokens. Here's how it generally works:

1. State Tokens: When a client (such as a web application) makes an authentication request to Keycloak, Keycloak can include a state parameter in the request. This state is a random token that the client must store and send back with the authentication response. 2. Validation: When the user is redirected back to the application after authenticating with Keycloak, the application sends the state token back to Keycloak. Keycloak then verifies that this token matches the one it originally issued. If they match, the response is considered valid; if not, it's possible that the request may be part of a CSRF attack, and the request is rejected.

This CSRF token mechanism is essential for OAuth 2.0 and OpenID Connect flows, particularly in scenarios where the redirection back to the application could potentially be manipulated.

Keycloak's robust handling of CSRF attacks through state tokens and its comprehensive token management system utilizing refresh tokens are critical for maintaining the security and integrity of applications using its services.

Building upon industry standards, Keycloak supports OpenID Connect (OAuth 2.0 + Authentication Layer), SAML 2.0, and OAuth 2.0.

Keycloak employs a user database of its own. Additionally, you can integrate with already-existing user directories like LDAP servers and Active Directory.

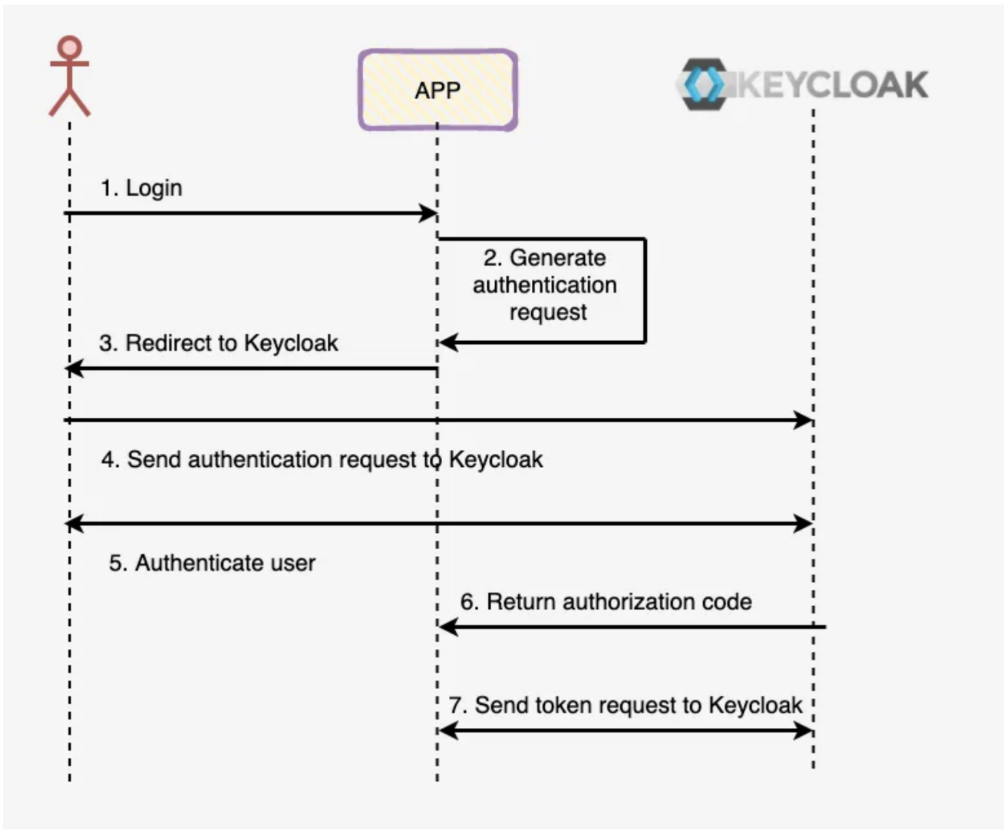

APP
KEYCLOAK
1. Login
2. Generate authentication request
3. Redirect to Keycloak
4. Send authentication request to Keycloak
5. Authenticate user
6. Return authorization code
7. Send token request to Keycloak

Figure 3.16: Keycloak integration [19]

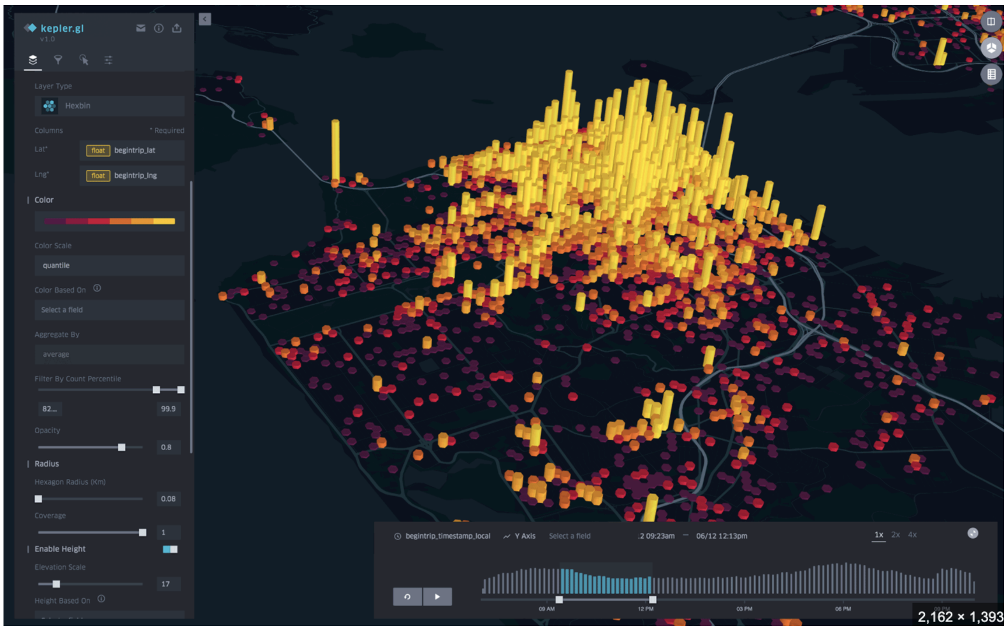
3.2.15 Kepler.gl

Kepler.gl is a data-agnostic, high-performance web-based application for visual exploration of large-scale geolocation data sets. Built on top of MapLibre GL and deck.gl, kepler.gl can render millions of points representing thousands of trips and perform spatial aggregations on the fly.

Kepler.gl provides a rich set of tools for creating complex geospatial visualizations directly in the browser. It supports a wide array of visualization types, from simple point maps to sophisticated heatmaps and choropleths.

One of Kepler.gl's strengths is its user-friendly interface and the ability to handle large datasets without significant performance trade-offs. It is designed to be used by both developers and non-developers, making it accessible for a wide range of users.

Kepler.gl can be easily customized and embedded in React applications. It allows extensive customization of map styles, colors, and interactions, making it suitable for tailored visualization needs.



Layer Type: Hexbin
Columns * Required
Lat*: float begintrip_lat
Lng*: float begintrip_lng
Color
Color Scale: quantile
Color Based On: Select a field
Aggregate By: average
Filter By Count Percentile: 82... 99.9
Opacity: 0.8
Radius
Hexagon Radius (Km): 0.08
Coverage: 1
Enable Height
Elevation Scale: 17
Height Based On

begintrip_timestamp_local Y Axis Select a field .2 09:23am — 06/12 12:13pm

Figure 3.17: Kepler.gl [20]

Kepler.gl enables powerful geospatial analysis, which can be crucial for applications dealing with logistics, urban planning, or environmental monitoring.

The tool provides an interactive user experience that allows end-users to explore and manipulate geospatial data intuitively.

Kepler.gl efficiently handles large datasets, which is essential for applications requiring the visualization of complex or extensive geospatial data.

The integration of Kepler.gl into React applications brings sophisticated geospatial visualization capabilities that enhance the analytical power and user engagement of modern web platforms. By leveraging its extensive features for creating dynamic and customizable maps, developers can provide users with powerful tools for data exploration and interpretation. Kepler.gl not only elevates the application's functionality but also its aesthetic appeal, making complex data accessible and understandable through intuitive visualizations.

3.2.16 json-server

In the development of modern web applications, especially during the early stages of development and testing, having a quick and reliable way to simulate a full-featured REST API can significantly speed up the process. json-server offers a straightforward solution by providing a full fake REST API with zero coding in less than 30 seconds, making it an invaluable tool for developers working on frontend applications who need to mock a backend system efficiently.

json-server is incredibly simple to set up. With just a minimal configuration in a JSON file, it can mimic a real API, allowing developers to focus on frontend development without waiting for backend services to be built.

It provides full flexibility in designing the API as per development needs. Developers can create custom routes, simulate various endpoints, and even integrate middleware to handle complex behaviors.

By providing instant back-end to front-end interaction possibilities, json-server accelerates development cycles. Developers can test features and handle data without any backend constraints, leading to faster iteration and debugging.

Developers can prototype a new application rapidly by providing an interactive API that can be used to mock data interactions.

json-server is ideal for testing front-end applications, providing reliable responses and the ability to quickly adjust the output data without touching any actual back-end code.

It is also an excellent tool for training purposes and demonstrations, where a full-fledged backend is not required.

json-server provides a powerful yet simple solution for simulating REST APIs, making it an excellent tool for frontend developers needing to work independently of backend development progress. Its ability to be up and running in seconds, combined with the ease of creating a mock database, ensures that it remains a favorite tool among developers for rapid development and testing. By integrating json-server into React application development workflows, teams can ensure that they are able to test features thoroughly and accelerate the development cycle without waiting for backend services to be fully implemented.

3.3 Development Process

The development process adopted in this thesis follows the Agile methodology, which supports iterative development and frequent reassessment of project requirements and goals. The key phases include:

1. **Planning:** Setting up goals, choosing the stack, and planning the sprints.
2. **Development:** Iterative cycles of coding, where components are designed, built, and integrated.
3. **Review:** Regular assessment of the progress and functionality through meetings and code reviews.
4. **Deployment:** Staging and production deployments to assess performance in real-world conditions.

5. **Feedback:** Gathering user feedback to inform future development cycles.

Chapter 4

Design and Implementation of Application

This chapter will demonstrate the application environment, the numerous components used, and the implementation of the program.

4.1 Overview of the Application

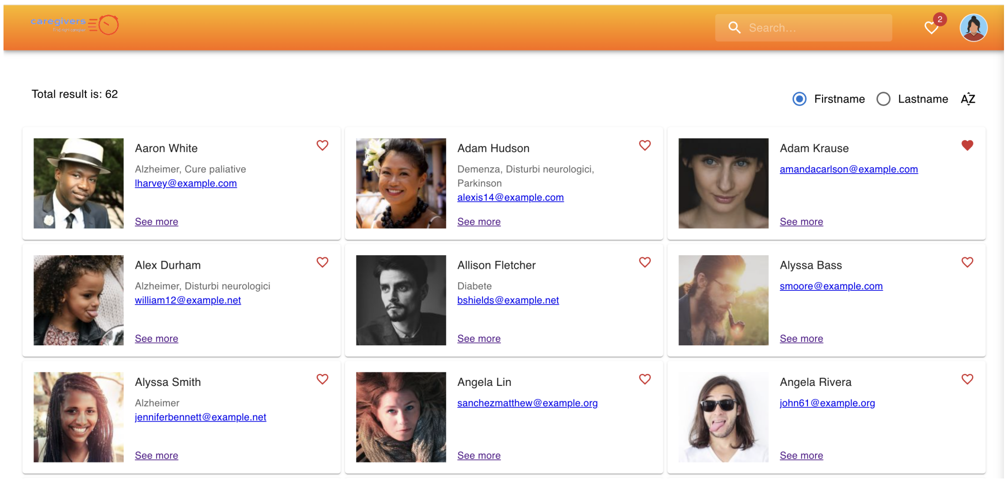
The goal of this web application is to show off advanced features for managing user interactions, secure data processing, and dynamic content display. The program provides a platform for demonstrating how modern web technologies may be effectively used to improve functionality and security in any web-based system, making it perfect for educational and professional demonstrations.

4.2 User Management Component

4.2.1 User Listing

The project is a React-based front-end application that uses Material UI for UI components and styling. On the application's home page, you can see a list of users. for user data display using infinite scroll to efficiently handle large datasets. As the user scrolls down, more data is fetched. The user can bookmark items, selected items that have been saved in the session, and click the see more button for further information. This home page is visible to all users, even those who are not authorized.

an anonymous user has the ability to bookmark items from a list. Upon selecting an item, the bookmark badge on the navigation menu is instantly updated to reflect the count of bookmarked items. When the user clicks on this badge, a display


Total result is: 62

Firstname Lastname

Aaron White
Alzheimer, Cure paliative
lharvey@example.com
See more

Adam Hudson
Demenza, Disturbi neurologici, Parkinson
alexis14@example.com
See more

Adam Krause
amandacarlson@example.com
See more

Alex Durham
Alzheimer, Disturbi neurologici
william12@example.net
See more

Allison Fletcher
Diabete
bshields@example.net
See more

Alyssa Bass
smoore@example.com
See more

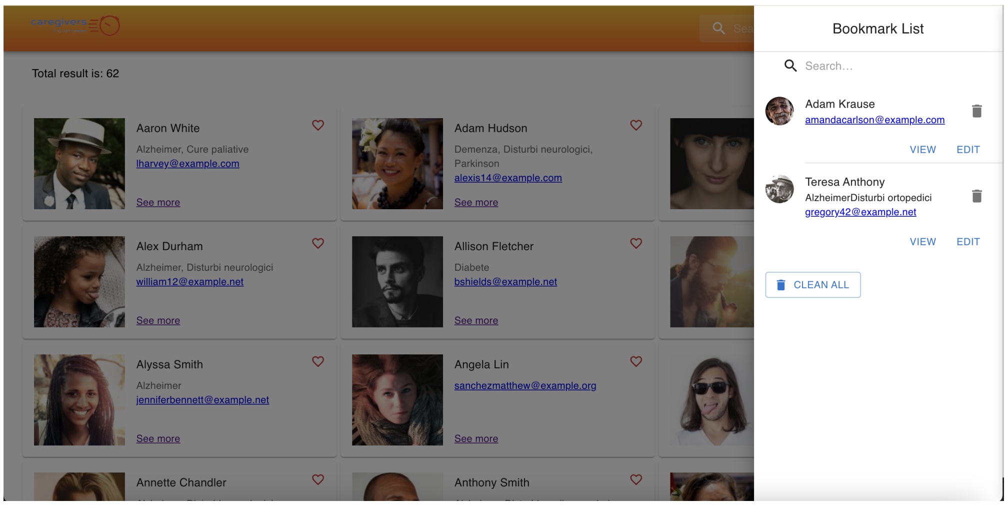
Alyssa Smith
Alzheimer
jenniferbennett@example.net
See more

Angela Lin
sanchezmatthew@example.org
See more

Angela Rivera
john61@example.org
See more

Figure 4.1: Home page of application

of all bookmarked items appears, allowing the user to review their selections. Furthermore, users have the option to delete any of the bookmarked items directly from this view, providing a seamless and intuitive interface for managing their preferences.


Bookmark List

Adam Krause
amandacarlson@example.com
VIEW EDIT

Teresa Anthony
AlzheimerDisturbi ortopedici
gregory42@example.net
VIEW EDIT

CLEAN ALL

Total result is: 62

Aaron White
Alzheimer, Cure paliative
lharvey@example.com
See more

Adam Hudson
Demenza, Disturbi neurologici, Parkinson
alexis14@example.com
See more

Alex Durham
Alzheimer, Disturbi neurologici
william12@example.net
See more

Allison Fletcher
Diabete
bshields@example.net
See more

Alyssa Smith
Alzheimer
jenniferbennett@example.net
See more

Angela Lin
sanchezmatthew@example.org
See more

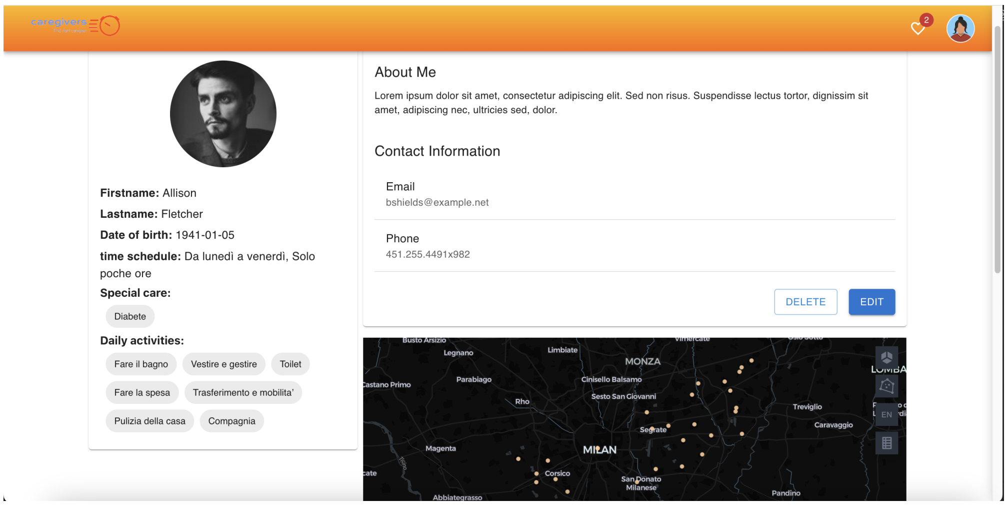
Annette Chandler

Anthony Smith

Figure 4.2: Manage the bookmark items

4.2.2 User Details Page with Enhanced Location Visualization

The User Details page is an essential part of the application, providing a full and interactive profile for each user. This page's major feature is the integration of the Kepler.gl map, which not only visualizes geographical data but also allows users to add and examine individual location points. This functionality improves comprehension of user actions and preferences based on location data.



Firstname: Allison
Lastname: Fletcher
Date of birth: 1941-01-05
time schedule: Da lunedì a venerdì, Solo poche ore
Special care:
Diabete
Daily activities:
Fare il bagno · Vestire e gestire · Toilet · Fare la spesa · Trasferimento e mobilita' · Pulizia della casa · Compagnia

About Me

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor.

Contact Information

Email
bshields@example.net

Phone
451.255.4491x982

Figure 4.3: User Detail page of application

In the data center, each user is assigned specific locations, which are represented on the map with colored points. This visual differentiation allows users to easily identify and interact with their selected locations, enhancing the usability and navigational efficiency of the map interface.

Authenticated users have the capability to edit and delete pages directly. Visible controls, such as buttons for these actions, are prominently displayed, enabling users to easily manage content and make necessary adjustments with just a few clicks.

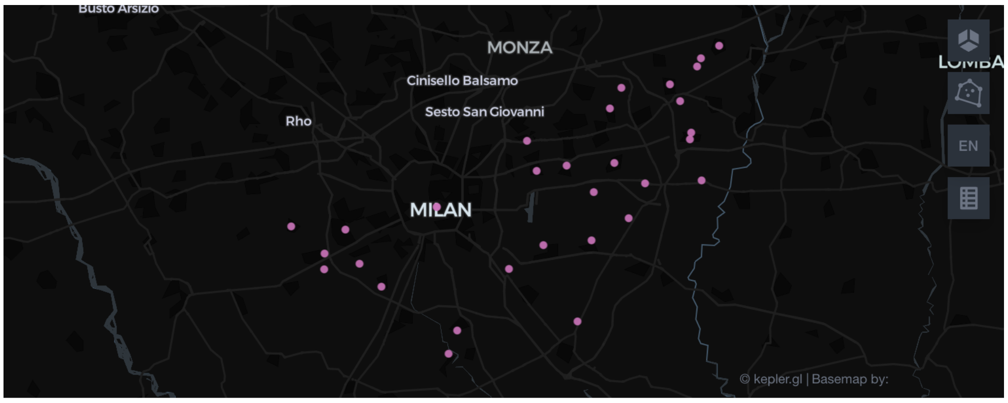
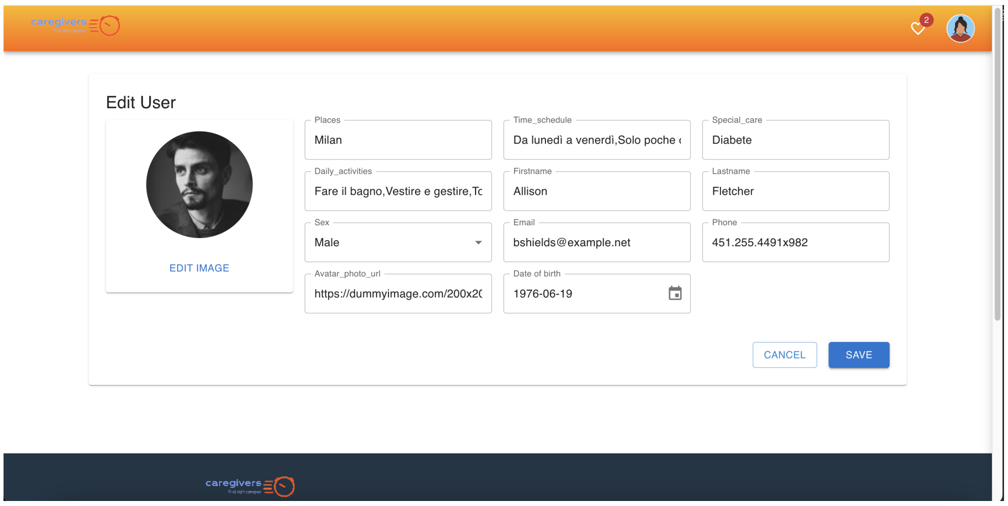


Figure 4.4: location points in the map

4.2.3 Edit page

The edit page of the application is designed to provide a user-friendly interface, allowing authenticated users to easily modify content. Upon accessing this page, users are presented with an intuitive layout where all editable fields are clearly delineated. Each field is pre-populated with existing content, enabling users to quickly see what changes may be needed. Functional buttons for saving changes or reverting edits are strategically placed to facilitate easy navigation.



Edit User

Places: Milan	Time_schedule: Da lunedì a venerdì,Solo poche	Special_care: Diabete
Daily_activities: Fare il bagno,Vestire e gestire,Tc	Firstname: Allison	Lastname: Fletcher
Sex: Male	Email: bshields@example.net	Phone: 451.255.4491x982
Avatar_photo_url: https://dummyimage.com/200x2(	Date of birth: 1976-06-19	

Figure 4.5: location points in the map

4.2.4 Login page

The application also have private route if user try to use the private route or click on the login button, the login status has been checked, and if they are not authenticated, they are redirected to the login page, which has been implemented by Keycloak.

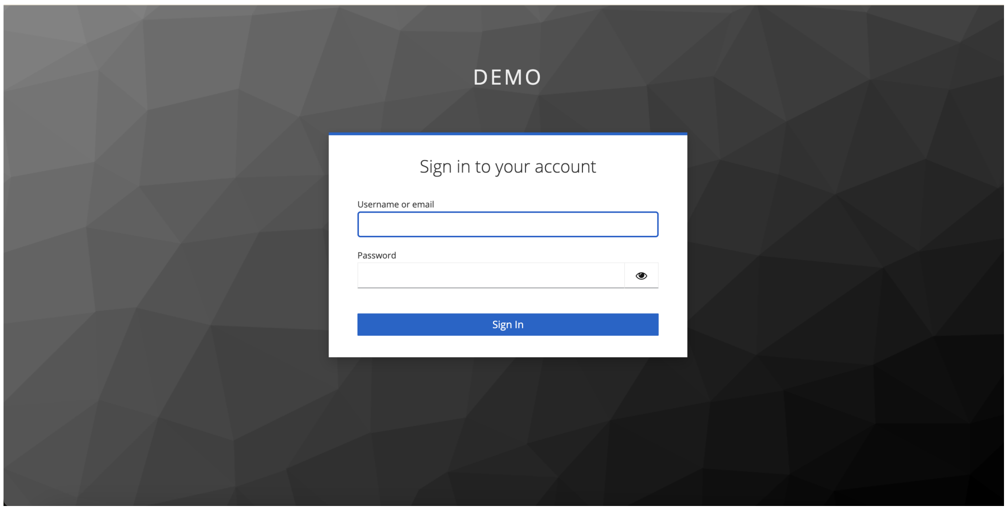


Figure 4.6: Login page of application

After logging in, users are seamlessly returned to the last page they visited before logging in. This is accomplished by saving the user's last path in local storage; upon successful authentication, the system utilizes this saved path to implement a return policy, ensuring users can continue their session without interruption.

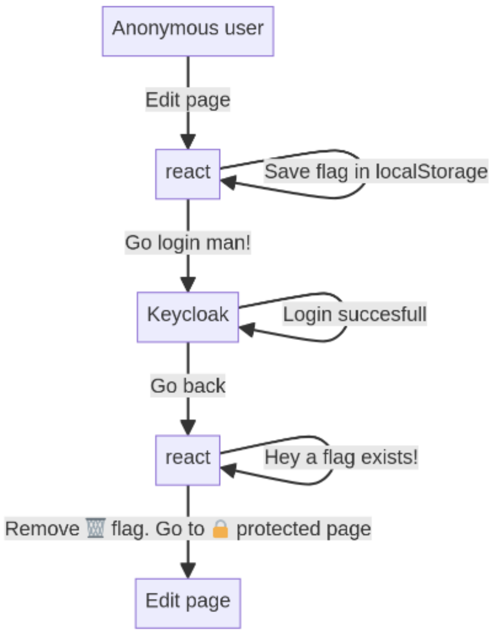
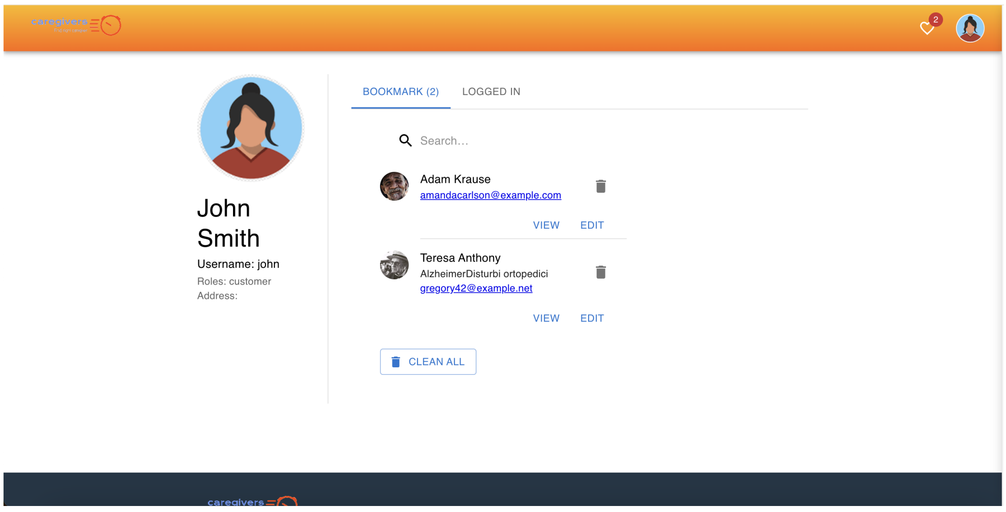


Figure 4.7: Return url pattern

4.2.5 User profile page

in this component, the user can see the list of bookmarked items and also manage them and in the other tab, it could be check the logged time into the system

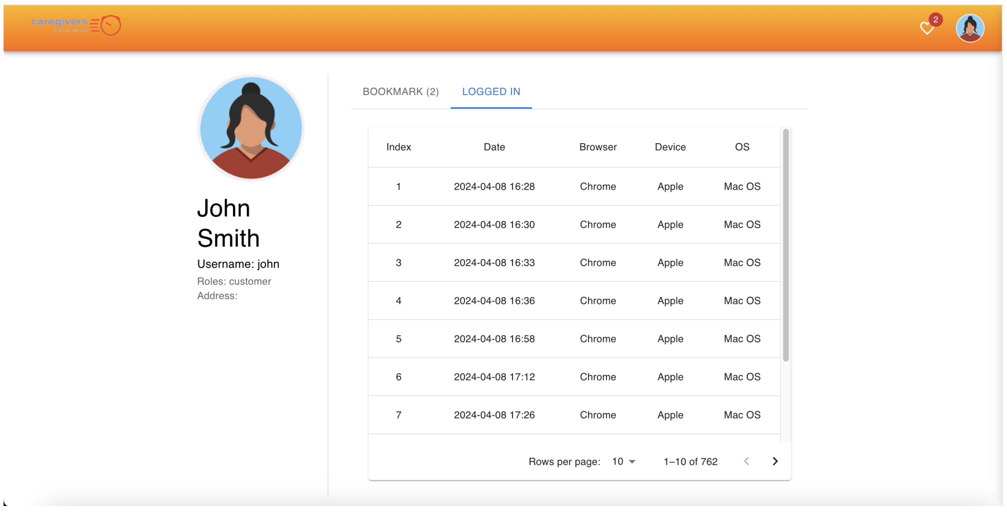


John Smith
Username: john
Roles: customer
Address:

Adam Krause
amandacarlson@example.com

Teresa Anthony
AlzheimerDisturbi ortopedici
gregory42@example.net

Figure 4.8: Profile page bookmark management



John Smith
Username: john
Roles: customer
Address:

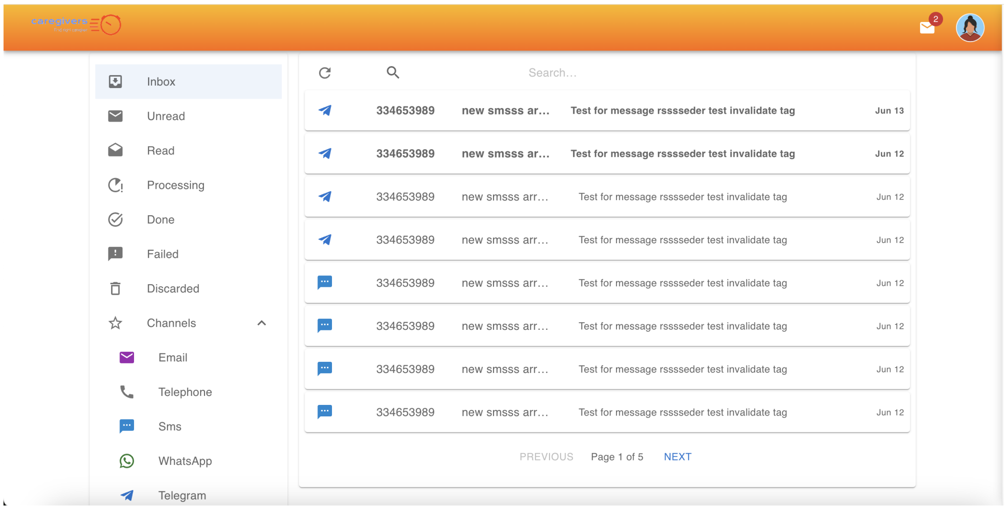
Index	Date	Browser	Device	OS
1	2024-04-08 16:28	Chrome	Apple	Mac OS
2	2024-04-08 16:30	Chrome	Apple	Mac OS
3	2024-04-08 16:33	Chrome	Apple	Mac OS
4	2024-04-08 16:36	Chrome	Apple	Mac OS
5	2024-04-08 16:58	Chrome	Apple	Mac OS
6	2024-04-08 17:12	Chrome	Apple	Mac OS
7	2024-04-08 17:26	Chrome	Apple	Mac OS

Rows per page: 10 1–10 of 762

Figure 4.9: Profile page user logged

4.2.6 Message page of the application

The Message Listing component is an important part of the application, since it collects and displays messages from multiple communication channels such as WhatsApp, Telegram, SMS, and email. This centralization improves user interaction by offering a consistent picture of communications, resulting in increased efficiency and reaction time. This component addressed a significant issue in handling messages coming from Gmail, which involved transforming them from base64 encoding to readable text.



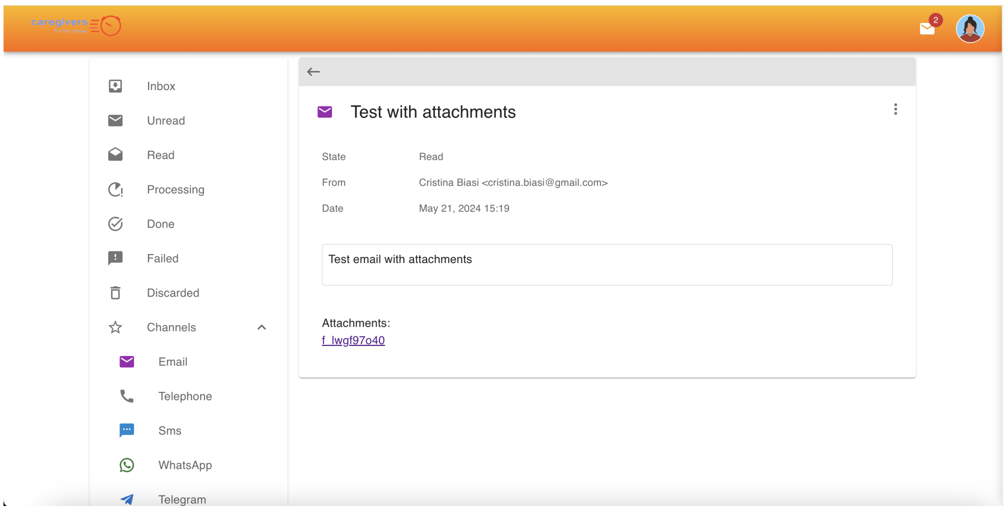
334653989	new smsss ar...	Test for message rsssseder test invalidate tag	Jun 13
334653989	new smsss ar...	Test for message rsssseder test invalidate tag	Jun 12
334653989	new smsss arr...	Test for message rsssseder test invalidate tag	Jun 12
334653989	new smsss arr...	Test for message rsssseder test invalidate tag	Jun 12
334653989	new smsss arr...	Test for message rsssseder test invalidate tag	Jun 12
334653989	new smsss arr...	Test for message rsssseder test invalidate tag	Jun 12
334653989	new smsss arr...	Test for message rsssseder test invalidate tag	Jun 12
334653989	new smsss arr...	Test for message rsssseder test invalidate tag	Jun 12

Figure 4.10: Message list page

The design of the Message Listing component focused on creating a unified interface where users could easily manage and access messages from various sources. The objectives included providing real-time updates, facilitating easy search and filtering capabilities, and ensuring a user-friendly experience across devices.

4.2.7 Message detail page

The Message Detail component provides a comprehensive view of individual messages, facilitating not only the display of detailed information but also the management of message status. This feature is particularly useful for handling messages from various channels. Users can update the status of each message to reflect its current processing stage, such as Processing, Done, Failed, or Discarded, enabling efficient tracking and management of communications.



Test with attachments

State Read

From Cristina Biasi <cristina.biasi@gmail.com>

Date May 21, 2024 15:19

Test email with attachments

Attachments:
f_lwgf97o40

Figure 4.11: Message detail page

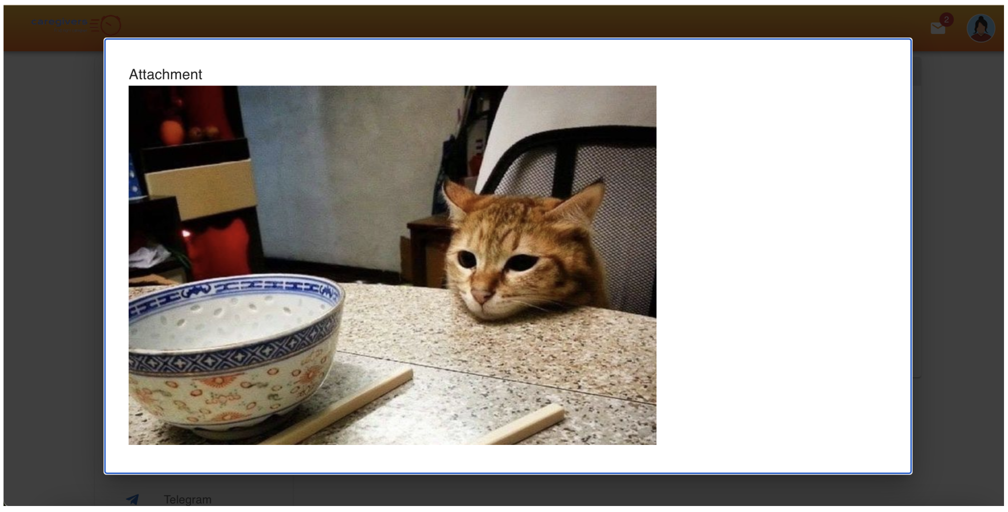


Figure 4.12: Display Attachment in message detail page

Attachments Display

For messages that include attachments, such as the email example provided, attachments are listed and can be accessed or downloaded directly from the page. This integration is crucial for emails where attachments often accompany communications.

4.3 Challenges and Solutions

One major challenge was designing a secure login mechanism that could protect user data against emerging cyber threats while maintaining a swift authentication process. Additionally, managing large datasets without degrading the application's performance posed a significant technical hurdle.

Chapter 5

Conclusions and potential future development

this thesis has successfully developed a web application that addresses key aspects of security and functionality for user interaction.

The primary objective was to create secure login mechanisms that protect user data while maintaining ease of use. By employing the latest encryption and authentication technologies, the application ensures that user credentials are securely managed.

Additionally, we concentrated on optimizing the application's performance, particularly in managing large data sets. By utilizing the highest and most strategic methods available, we significantly improved the efficiency of displaying extensive lists and conducting advanced searches. This not only ensures a seamless user experience but also enhances the application's capability to handle complex data interactions swiftly and effectively.

Furthermore, the application integrates a dynamic mapping feature that efficiently displays various points of interest on a map. This functionality enhances user engagement by providing real-time geographical insights, which are crucial for applications requiring spatial awareness.

Lastly, the system is designed to aggregate messages from multiple platforms, presenting them in a unified list. This not only improves the user experience by centralizing communication but also employs advanced search techniques to manage and navigate through large datasets effectively.

Overall, the thesis has not only demonstrated the practical application of cutting-edge security and data management techniques but also has laid a foundation for future improvements and iterations in web application development. The methodologies implemented herein are scalable and adaptable, ensuring that the application remains robust and relevant in the face of evolving technological challenges.

Bibliography

[1] Gursheen Kaur and Raj Gaurang Tiwari. «Comparison and Analysis of Popular Frontend Frameworks and Libraries: An Evaluation of Parameters for Frontend Web Development». In: *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)*. 2023, pp. 1067–1073. DOI: 10.1109/ICESC57686.2023.10192987 (cit. on p. 13).

[2] Pankaj Keshari, Priya Maurya, Pankaj Kumar, and Alok Katiyar. «Web Development Using ReactJS». In: *2023 5th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*. 2023, pp. 1571–1575. DOI: 10.1109/ICAC3N60023.2023.10541743 (cit. on p. 13).

[3] Mark Thomas. 2018 (cit. on p. 13).

[4] *Major Benefits of React JS*. https://www.icoderzsolutions.com/blog/react-js-benefits/. Accessed: 2024-06-15 (cit. on p. 13).

[5] Giuseppe Psaila. «Virtual DOM: An Efficient Virtual Memory Representation for Large XML Documents». In: *2008 19th International Workshop on Database and Expert Systems Applications*. 2008, pp. 233–237. DOI: 10.1109/DEXA.2008.117 (cit. on p. 14).

[6] *What is JSX and How JSX works*. https://medium.com/@danyal_imran/everything-react-all-about-jsx-4a5123ac8606. Accessed: 2024-05-23 (cit. on p. 14).

[7] *How webPack is works*. https://webpack.js.org/. Accessed: 2024-06-10 (cit. on p. 15).

[8] *Single-page application vs. multiple-page application*. https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. Accessed: 2024-06-02 (cit. on p. 16).

[9] Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, Frank Piessens, and Wouter Joosen. «Protected Web Components: Hiding Sensitive Information in the Shadows». In: *IT Professional* 17.1 (2015), pp. 36–43. DOI: 10.1109/MITP.2015.12 (cit. on p. 19).

[10] *Understanding Session Management Using React Router V6.* https://hackernoon.com/understanding-session-management-using-react-router-v6. Accessed: 2024-06-11 (cit. on p. 20).

[11] *what Is Redux.* https://redux.js.org/. Accessed: 2024-05-13 (cit. on p. 23).

[12] Akanksha and Akshay Chaturvedi. «Comparison of Different Authentication Techniques and Steps to Implement Robust JWT Authentication». In: *2022 7th International Conference on Communication and Electronics Systems (ICCES).* 2022, pp. 772–779. DOI: 10.1109/ICCES54183.2022.9835796 (cit. on p. 27).

[13] *Structure of JSON web token.* https://www.linkedin.com/pulse/understanding-jwt-syed-shamim-hosan/. Accessed: 2024-06-25 (cit. on p. 29).

[14] *JWT – Token Based Authentication.* https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4. Accessed: 2024-06-21 (cit. on p. 30).

[15] *Session vs Token Based Authentication.* hhttps://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4. Accessed: 2024-06-19 (cit. on pp. 31, 32).

[16] *JWT Authentication and Token Lifecycle Management Flow.* https://journals.sagepub.com/doi/10.1177/1550147718801535. Accessed: 2024-06-24 (cit. on p. 34).

[17] Boyan Chen, Pavol Zavarsky, Ron Ruhl, and Dale Lindskog. «A Study of the Effectiveness of CSRF Guard». In: *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing.* 2011, pp. 1269–1272. DOI: 10.1109/PASSAT/SocialCom.2011.58 (cit. on p. 35).

[18] *Exploring Cross-Site Request Forgery (CSRF) vulnerabilities: Still a threat!* https://www.akto.io/blog/csrf-comprehensive-guide. Accessed: 2024-06-29 (cit. on p. 36).

[19] *Introduction to Keycloak.* https://abdulsamet-ileri.medium.com/introduction-to-keycloak-227c3902754a. Accessed: 2024-06-20 (cit. on p. 38).

[20] *kelper gl.* https://kepler.gl/. Accessed: 2024-06-19 (cit. on p. 39).