CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Physics

Bachelor Thesis:

Enhancing User Experience in React-Based Question-Answering Bot Applications: A Comprehensive Study and Implementation

Project of: Gökdeniz Kaymak

Supervisor: Ing. Jan Šedivý, CSc.

Study program: Electrical Engineering and Computer Science

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kaymak Gokdeniz**	Personal ID number:	**506123**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Electrical Power Engineering**		
Study program:	**Electrical Engineering and Computer Science**		

II. Bachelor's thesis details

Bachelor's thesis title in English:

Multi-Modal Chat Web Application in React

Bachelor's thesis title in Czech:

Multi-Modální Konverzační Webová Aplikace v Reactu

Guidelines:

Design a Multi-Modal client web application with emphasis on question answering with the following features:
- Utilize REACT as the primary framework for component-based development along with StoryBook application.
- Design and document REACT components in Storybook to display text (Markdown), images, video, and voice control.
- Implement a responsive design strategy to seamlessly adapt to diverse devices, including desktops, tablets, and mobile phones, ensuring an optimal user experience.
- Enable bidirectional streaming support for both text and speech services from Google and Microsoft, facilitating a seamless and dynamic flow of data in both directions.
- Allow setting up the application using a configuration file (part of HTTPS).
- Design a protocol that enables the server to dynamically upload components, configure applications, such as components' colors, shapes, and other features.
- Provide well-documented and organized source code for the entire client application.
- Include a testing plan covering unit tests, integration, UI/UX, security, authentication and testing.
- The assignment does not include the server, it will be provided including the API description.

Bibliography / sources:

[1] Mastering React.js: Best Practices and Design Patterns for High-Quality Applications
https://medium.com/@techsuneel99/mastering-react-js-best-practices-and-design-patterns-for-high-quality-applications-0e0ef381df8
[2] React File Structure
https://medium.com/front-end-field-guide/react-file-structure-55e94d15c3be
[3] Building a React Components Living Documentation using React-Storybook.
https://medium.com/@mlthuret/building-a-react-components-living-documentation-using-react-storybook-5f11f0e7d23e
[4] Jan Šedivý spouští Flowstorm, nástroj pro vývoj aplikací ovládaných hlasem
https://www.lupa.cz/aktuality/jan-sedivy-spousti-flowstorm-nastroj-pro-vyvoj-aplikaci-ovladanych-hlasem/
[5] Flowstorm: Open-Source Platform with Hybrid Dialogue Architecture
https://arxiv.org/abs/2212.09377

Name and workplace of bachelor's thesis supervisor:

Ing. Jan Šedivý, CSc. Big Data and Cloud Computing CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **26.01.2024** Deadline for bachelor thesis submission: _____

Assignment valid until: **21.09.2025**

Ing. Jan Šedivý, CSc. Supervisor's signature	doc. Ing. Zdeněk Müller, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Declaration:

“I hereby declare that this bachelor’s thesis is the product of my own independent work and that I have clearly stated all information sources used in the thesis according to Methodological Instruction No. 1/2009 – “On maintaining ethical principles when working on a university final project, CTU in Prague“

Date Author’s signature

....

Acknowledgment:

I extend my heartfelt gratitude to Ing. Jan Šedivý, CSc., for his invaluable guidance, support, and encouragement throughout the development of this thesis. His expertise and mentorship have been instrumental in shaping the content and direction of this research.

Abstract

This thesis presents a comprehensive analysis of React one-page application development focusing on designing and implementing an intuitive interface for question-answering bots. The research encompasses the integration of various input file formats, including PDF, XLSX, and TXT, and explores best React practices, libraries, and tools to ensure a seamless user experience. A thorough testing plan was devised to evaluate functionality, performance, and reliability. The mechanisms for collecting user feedback were incorporated and analyzed to identify areas for improvement. This document provides detailed technical specifications, user manuals, and guidelines for future development and maintenance of the developed application.

Table of contents

List of Abbreviations

1. CI/CD - Continuous Integration/Continuous Deployment
2. YAML - YAML Ain't Markup Language
3. JSX - JavaScript XML
4. CDN - Content Delivery Network
5. SSL - Secure Sockets Layer
6. README.md - Readme Markdown
7. TypeDoc - TypeScript Documentation Generator
8. UI - User Interface
9. LLM - Large Language Models
10. PDF - Portable Document Format
11. DOM - Document Object Model
12. MVC - Model-View-Controller
13. SSR - Server-Side Rendering
14. NLU - Natural Language Understanding
15. API - Application Programming Interface
16. CSS - Cascading Style Sheets
17. Git - Version Control System
18. AWS - Amazon Web Services
19. Heroku - Cloud Platform as a Service (PaaS)
20. Azure - Microsoft Azure
21. Vue.js - JavaScript Framework
22. HTML - HyperText Markup Language
23. TTS - Text-to-Speech
24. ESLint - ECMAScript Lint
25. PC - Personal Computer

Figures

1. Introduction

The thesis delves into the realm of question-answering bots, emphasizing the importance of both user interface (UI) quality and the robustness of underlying code. In this context, attention is paid to ensuring that the codebase is documented, tested, and deployed. With a core objective of aiding customers in navigating user queries and optimizing internal resource management through PDF analysis and question resolution, the thesis underscores the significance of an efficient and user-friendly bot interface. To achieve these objectives, the study adopts React with TypeScript for front-end development, recognizing its suitability for creating dynamic and scalable user interfaces.

2. State-of-the-Art

2.1 Requirement Specification

2.1.1 Target Audience

The target audience consists of individuals and organizations seeking to integrate a web bot into their systems. This audience may include businesses looking to enhance customer support by providing automated assistance with inquiries, as well as organizations aiming to streamline internal communication by empowering employees with a tool to address their queries efficiently. Whether utilized for customer-facing interactions or internal knowledge management, the software caters to users seeking to leverage the capabilities of a web bot to improve productivity, streamline processes, and enhance user experience within their respective domains.

2.1.2 Requirements

The software needs an intuitive and user-friendly interface. It must possess the capability to display diverse media formats like PDF files, code snippets, and images, accommodating various types of content seamlessly. Furthermore, it should support multiple input methods including text input, voice input, and button clicks to ensure accessibility for users. A key requirement is the provision of extensive configuration options, enabling customers to tailor the software to their specific preferences and requirements. For stable development, testing procedures and version control methods are important. comprehensive documentation of the codebase is also essential to support future development efforts, aiding developers in understanding and extending the software's functionality effectively.

2.2 User Interface (UI)

A UI is the interface between users and digital systems, comprising elements like buttons and menus. We choose React, a front-end library, as our framework for development.

2.2.1 What is React

React is an open-source JavaScript library developed by Facebook, primarily used for building highly dynamic and interactive UI's. It operates on a component-based architecture, allowing developers to create reusable UI elements, thereby enhancing efficiency and simplifying interface management. Central to React is its virtual DOM feature, which optimizes rendering by updating only specific parts of a web page when data changes, leading to faster, more responsive, and scalable web applications. React's impact on modern web development practices is significant, providing a streamlined approach to constructing robust single-page applications and promoting a structured methodology in front-end development.

2.2.2 Brief History

In 2011, Facebook aimed to enhance user experience by creating a more dynamic and responsive interface. To address this challenge, Jordan Walke, a Facebook software engineer, developed React.js. Initially implemented in Facebook's newsfeed, React's unique approach to manipulating the Document Object Model (DOM) transformed the company's web development strategies. Subsequently, React gained rapid popularity within the JavaScript ecosystem after its release as an open-source technology. **[1]**

2.2.3 Advantages and Disadvantages

React streamlines web app development with minimal coding. It emphasizes UI speed improvement but comes with pros and cons.

Advantages:

1. Dynamic Apps: Simplifies complex HTML string creation.
2. Reusable Components: Simplifies app development and maintenance.
3. Performance Boost: Speeds up app performance with virtual DOM.

Disadvantages:

1. Rapid Development Pace: Challenges in keeping up with updates.
2. Limited Scope: Focuses mainly on UI layers, requiring additional tech.
3. JSX Complexity: JSX complexity can be daunting for new developers. **[2]**

2.2.4 Alternatives Considered

1. Angular

Angular is an open-source framework by Google, utilizes TypeScript and offers a full-fledged MVC (Model-View-Controller) framework. Angular is robust for enterprise-grade applications, providing clean code development and dependency injection. Its two-way data binding, comprehensive features, and ready-made solutions make it suitable for certain project requirements.

The preference for React over Angular is often due to React's flexibility, efficient performance with virtual DOM, ease of learning, and scalability. Its modular approach and extensive community support make it a versatile choice for building customizable applications, especially when developers have expertise in HTML, CSS, and JavaScript.

Therefore, the choice between React and Angular hinges on project-specific needs, developer expertise, and the complexity of the application being developed. For those seeking a more customizable and scalable approach in front-end development, React's attributes often make it the preferred choice over Angular. **[3]**

2. NextJS

Next.js is a JavaScript framework built upon React, shines in enabling server-side rendering (SSR) and static website development, offering features like SSR, static export, and enhanced performance. While React's extensive resources, adaptability, and support make it the go-to choice for diverse and dynamic application development needs, Next.js simplifies server-side rendering and static site generation, catering to specific requirements in a more specialized domain. Ultimately, React's flexibility and extensive toolset continue to position it as the preferred option for developers seeking versatility and robustness in creating sophisticated web applications. **[4]**

Here is a graph from Google Trends displaying the popularity of the 3 frameworks compared (Searched on 12/05/2024, with the worldwide parameter)

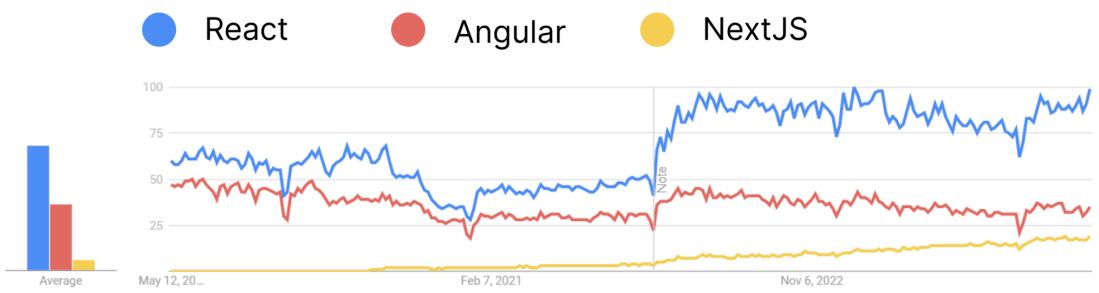


Figure 2.1: Google Trends comparison for React, NextJS and Angular.

2.3 Project Folder Structure

The project folder structure is essentially the organized arrangement of files and directories within a project's main folder, outlining where different types of files, such as source code, assets, and configuration files, are located.

The beginner folder structure is simplistic, suitable for smaller projects with limited folders (components, hooks, tests).

The advanced structure introduces an extensive system, grouping code by features, providing comprehensive organization. However, it might be overwhelming for smaller projects, leading to underutilized folders and complexity.

The intermediate structure strikes a balance, expanding from the beginner setup by introducing more folders for better organization without overwhelming users. It maintains clarity between global and page-specific code, making it suitable for varying project sizes. The comparison of folder structures—beginner, intermediate, and advanced—derives from **[5]**.

In addition, Each component within their respective folders now contains separate CSS files.

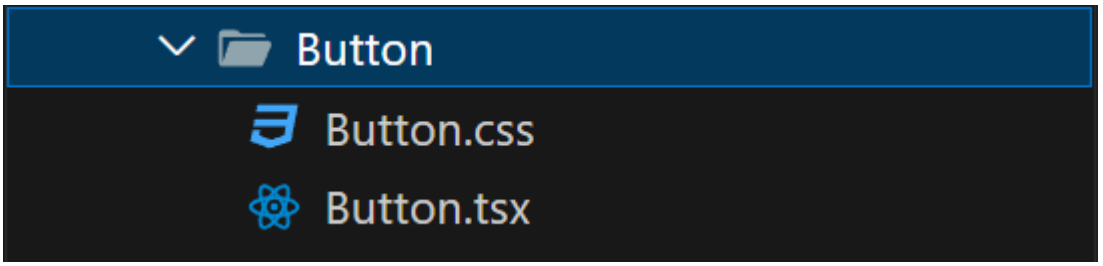


Figure 2.2: Displaying the Button component folder.

2.4 Creating Dialogues

Flowstorm is an open-source conversational AI platform designed for creating, executing, and analyzing conversational applications. It offers a novel dialogue architecture that combines tree structures with generative models, allowing for rapid dialogue execution and flexibility in handling various conversation scenarios. The platform utilizes tree structures for training Natural Language Understanding (NLU) models specific to dialogue scenarios, while generative models enhance dialogue functionality across applications. Flowstorm provides a user-friendly visual editor, enabling both novice users and experts to design applications easily and extend functionality with custom code. Notably, it facilitates asset reuse across applications and supports multiple languages. Flowstorm's architecture has been proven effective, with applications like Alquist, a winning socialbot in the Alexa Prize Socialbot Grand Challenge, demonstrating its suitability for complex conversational systems. **[6]**

With Flowstorm, crafting tailored applications for diverse clients is straightforward. Its intuitive interface and hybrid dialogue architecture empower developers to create interactive experiences that meet clients' unique needs. Leveraging its out-of-the-box components streamlines the process, enabling rapid prototyping and deployment across various domains.

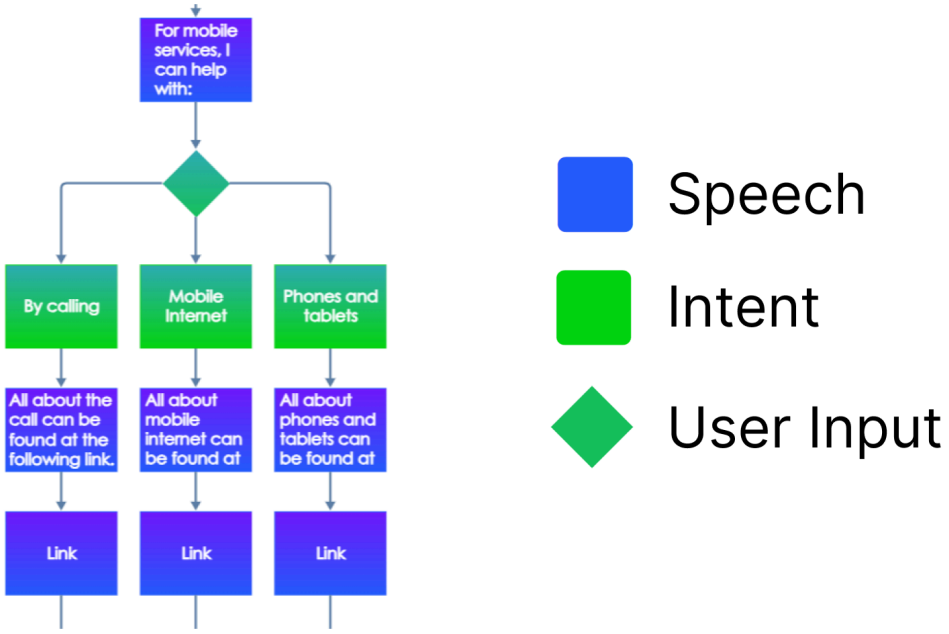


Figure 2.3: Flowstorm’s visual dialogue building interface

2.5 Client-Server Architecture

Client-server architecture is a computing model that separates client and server roles, with clients making requests and servers processing them, allowing for efficient and distributed computing. In our application, the client-server architecture functions as follows: When a user visits our client's website, the build files of our embedded bot (JavaScript and CSS) are loaded on their device, allowing them to interact with the user interface. This UI connects to Flowstorm, which acts as an intermediary between the user and the dialogues it manages. When the user asks a question, we call our API through Flowstorm, which is equipped with query-answering capabilities and can also provide PDF files as needed. This architecture ensures smooth and efficient interactions between the user, the application, and the server-side resources.

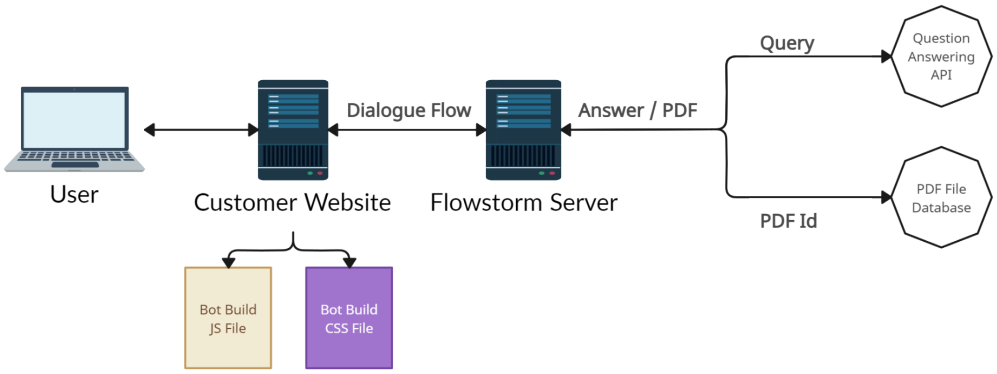


Figure 2.4: Displaying the client-server architecture

2.6 Version Control

Git is a version control system that tracks changes to files in a project, allowing multiple developers to collaborate efficiently by managing different versions of the codebase. Services like GitLab provide a platform for hosting Git repositories, offering additional features such as issue tracking, code review tools, and continuous integration pipelines to streamline the software development process further. **[7]**

2.7 Testing

Testing ensures the functionality and reliability of software by systematically evaluating its performance against predefined criteria. It involves executing the code under various conditions to uncover errors or bugs, ensuring that the software behaves as expected. Through rigorous testing, developers identify and rectify issues, thereby enhancing the overall quality of the product and providing users with a seamless experience.

2.7.1 Types of Testing

Software testing includes manual and automated methods. Manual testing involves human interaction, which can be error-prone and resource-intensive. Automated testing relies on pre-written scripts executed by machines for robustness and reliability. Types of testing range from unit and integration testing to functional, end-to-end, acceptance, performance, and smoke testing, each serving unique purposes in ensuring software quality and functionality. **[8]**

2.7.2 Testing Methods of Choice

For our project, we adopt a blend of integration testing and unit testing methodologies. Integration testing is crucial as it allows us to simulate interactions between various components of the application, particularly focusing on mocking the server to isolate UI testing from backend functionality, which falls outside our project scope. Meanwhile, unit testing plays a pivotal role in ensuring the fundamental functionality of our React components, providing granular assessments of their behavior and performance. This hybrid approach enables us to comprehensively evaluate both the integration of our application's components and the core functionality of its frontend elements.

2.7.3 Cypress

Cypress stands out as an ideal tool for integration testing thanks to its user-friendly syntax, powerful mocking capabilities, and seamless test execution. With features designed specifically for web applications, Cypress simplifies the process of testing interactions between frontend components and backend services.

2.7.4 Jest

Jest shines for unit testing TypeScript, offering simplicity and robustness. Tailored for JavaScript and TypeScript, Jest integrates seamlessly into projects, enabling straightforward testing with built-in TypeScript support. With features like snapshot testing and coverage reporting.

2.8 Deployment

Deployment in frontend development is the act of making a web application accessible to users by releasing it onto a server or hosting platform. This process involves transferring files and configuring settings to ensure the application functions properly in its production environment. Firebase was chosen for its ease of use and robust features, allowing for efficient and seamless deployment of frontend applications.

2.8.1 What is Firebase

Firebase, Google's versatile application development platform, offers a comprehensive suite of services tailored to streamline web application development and expansion. Its serverless architecture simplifies development by eliminating the need for developers to manage servers, allowing them to focus on building and optimizing their applications. With Firebase, developers can seamlessly integrate features like real-time database synchronization, user authentication, cloud storage, and cloud functions directly into their web projects. This platform's popularity stems from its smooth database management, scalability, and robust security features, making it a preferred choice for many organizations and seasoned developers in the web development community. Furthermore, Firebase provides extensive documentation and support for various web frameworks, including React, Angular, and Vue.js, empowering developers to create dynamic and interactive web apps tailored to their clients' needs. **[9]**

2.8.2 Alternatives Considered

In comparing backend platforms for web application deployment, alternatives like AWS, Heroku, and Azure were considered alongside Firebase. AWS boasts strong performance and scalability but requires more management and can lead to higher costs. Heroku simplifies deployment but lacks certain features like static IP addresses. Azure offers scalability but demands more management and expertise. Ultimately, Firebase's user-friendly interface, beginner-friendly setup, extensive documentation, and generous deployment credits made it the preferred choice. Its real-time database, authentication services, and seamless scalability further solidified its suitability for rapid application development. **[10]**

2.9 Documentation

Comprehensive documentation improves collaboration among team members and supports onboarding new developers efficiently. We decided to handle this by implementing a robust README.md file to outline the project's general structure and usage. Additionally, we incorporated code comments to explain complex logic and functions within the codebase. For visual component documentation, we utilized React Storybook, providing a centralized hub for showcasing and exploring our React components. These approaches enable us to maintain an accessible and well-organized codebase that is straightforward for new developers to understand and begin working with.

2.9.1 What Is Storybook

Storybook is a powerful tool widely used in React development that facilitates the creation, testing, and showcasing of UI components in isolation. It acts as a sandbox environment where developers can build and view individual components outside the context of the larger application. Storybook allows for efficient component-driven development by enabling developers to craft reusable and interactive components independently, write test cases for these components, and display them in various states or scenarios, making it easier to spot design inconsistencies or potential issues early in the development process. This tool significantly streamlines the workflow for React developers, fostering collaboration, enhancing component quality, and ultimately contributing to the overall robustness of the application's user interface.

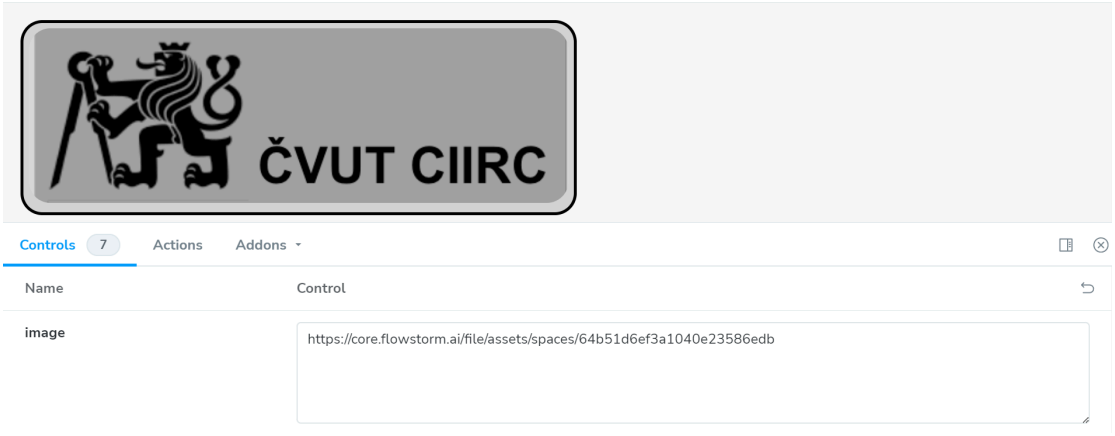


Figure 2.5: Displaying the UI of Storybook and the control panel of the component

2.9.2 Advantages, Disadvantages of Storybook

"Tim Davidson delves into the benefits and challenges of using Storybook as a development environment for UI components. Storybook offers advantages like isolated component development, faster UI creation, and simplified debugging, making it a go-to tool for managing complex projects. It enables developers to create, test, and maintain UI components independently of the main application, aiding in component organization and reuse. Storybook's compatibility with various frameworks like React, Vue, and Angular amplifies its utility for projects with numerous components. However, its setup complexity, management of a large number of stories, and limited support for certain frameworks pose challenges. Despite these frustrations, Storybook facilitates faster development, eases testing, streamlines collaboration between designers and developers, and enables customization through its extensive ecosystem of add-ons. The tool's best practices include maintaining organized stories, leveraging add-ons, integrating with testing frameworks, and regular updates to maximize its potential in software development." **[11]**

Reasons for using Storybook:
Storybook serves as a crucial tool in UI development, offering benefits like:

1. ***Isolation of components***: Allows for independent development and testing of UI components without affecting the entire application.
2. ***Faster and efficient development***: Facilitates faster development and efficient testing of different component variations.
3. ***Easy access and management***: Provides a centralized platform to view, manage, and modify components.
4. ***Collaboration and customization***: Enhances collaboration between designers and developers and offers extensive customization through add-ons.
5. ***Simplified testing and debugging***: Provides a dedicated environment for testing and debugging components, ensuring early issue detection.

Ultimately, Storybook streamlines UI component management and testing, despite its initial complexities, making it an invaluable asset for software developers.

3. Development

3.1 Project Setup

The initial setup of the project began with the creation of the UI using "npx create-react-app." This command established the foundational structure necessary for React development, configuring essential dependencies and settings. Following this setup, the project incorporated server-side communication with Flowstorm, a system providing application flow. The integration involved cloning the code from a pre-existing codebase **[12]**, enabling interaction with Flowstorm's server-side resources. By leveraging this existing infrastructure, the project rapidly integrated essential functionalities for managing application flows, streamlining development efforts.

3.2 Component Creation

The React documentation provides guidance on building modular UI components for web applications. It highlights the use of JSX for integrating markup with JavaScript, facilitating dynamic UI development. Components communicate via props, enabling data exchange and flexible rendering. Techniques like conditional rendering and rendering lists enhance component versatility. Emphasizing pure functions for component logic promotes scalability and reduces bugs. Overall, the documentation equips developers with best practices for creating robust and maintainable UIs. **[13]**

3.3 Component Hierarchy and State Management

In React, hierarchy refers to the organization of components within a tree-like structure, where each component is nested within its parent component. This hierarchical arrangement determines the flow of data and props between components, facilitating the building of complex user interfaces. State management in React involves handling and updating the internal state of components, allowing them to maintain and display dynamic data. By managing state effectively, React ensures that changes in data trigger re-rendering only where necessary, optimizing performance and maintaining a consistent user experience.

Figure 3.1: Different components marked on the UI

3.3.1 Hierarchy

Understanding the component hierarchy is essential for developers to effectively manage and maintain their codebase, as it provides a clear visual representation of how different parts of the application interact and depend on each other.

Here are two figures explaining the hierarchy of our React components

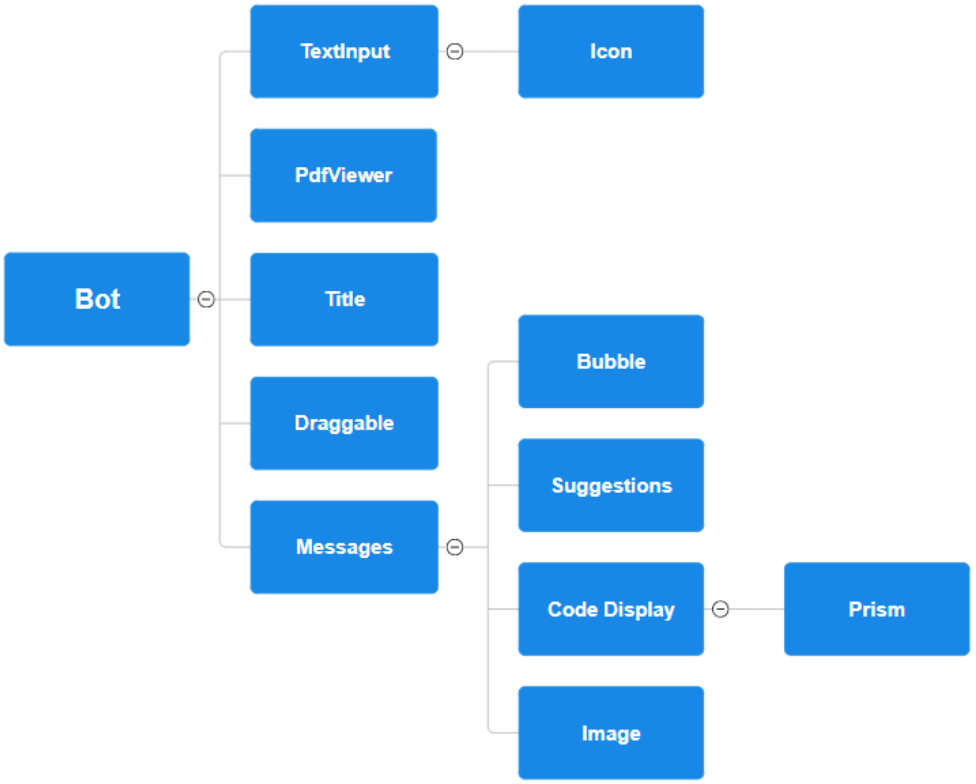


Figure 3.2: The hierarchy of React components in a tree graph

3.3.2 State Management

React state management relies on the useState, useEffect, and useRef hooks. We utilize useState to effectively manage component-level state, ensuring smooth data tracking and updates within each component. With useEffect, we synchronize state changes creating a consistent application. Additionally, useRef allows us to manipulate DOM elements directly, increasing our control over the UI. To facilitate seamless communication between parent and child components, we pass useState variables down the component tree, enabling multiple different components to adjust to any changes on the variable.**[14]**

The useState variable named messages, coupled with its setter function setMessages, plays a pivotal role in managing the messages displayed on the screen. This state variable serves as the central repository for all messages, allowing for seamless manipulation and rendering throughout the application. As messages are added or removed, setMessages updates the state accordingly, triggering re-renders as necessary to reflect the changes in real-time. To facilitate user interaction, both messages and setMessages are passed down to various components, such as the TextInput component for adding new messages and the Messages component responsible for displaying the messages. By centralizing message management through the useState hook and effectively distributing it across relevant components, we foster a robust and responsive messaging system that enhances the overall functionality and usability of our software project.

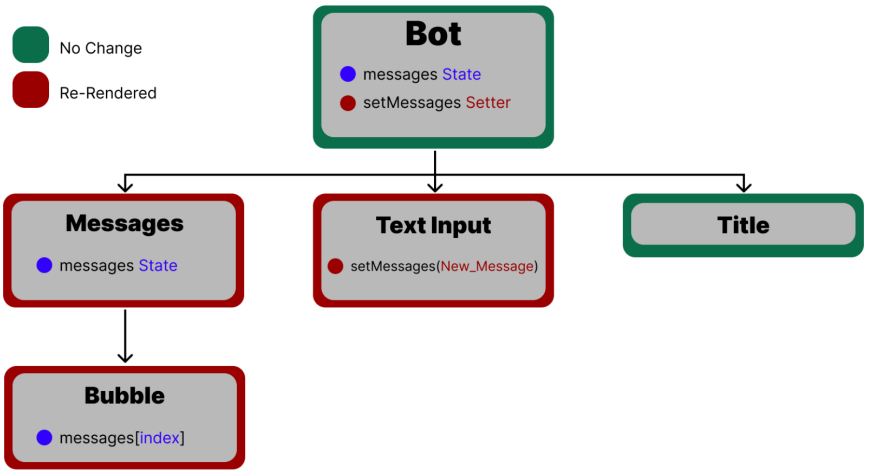


Figure 3.3: messages and setMessages state changes.

3.4 UI Development

UI development involves creating the visual and interactive elements of a digital application, such as buttons, menus, and forms. We have developed a variety of user input methods, such as button clicks, text input, and voice input. In addition, we've created various UI elements to visualize information effectively, such as code displays, PDF views, and image displays. To maintain high code quality, We used the following tools: ESLint and Prettier, which help maintain clean, consistent, and well-structured code.

3.4.1 Design Overview

We have crafted an interface where users can easily interact with the application. They can view messages, mute or unmute the bot, and click the microphone icon for voice input. A restart button allows users to reset the app, while a text field enables them to type their queries. The messages are displayed in a scrollable element for easy viewing, and suggestive buttons help guide users through the app.

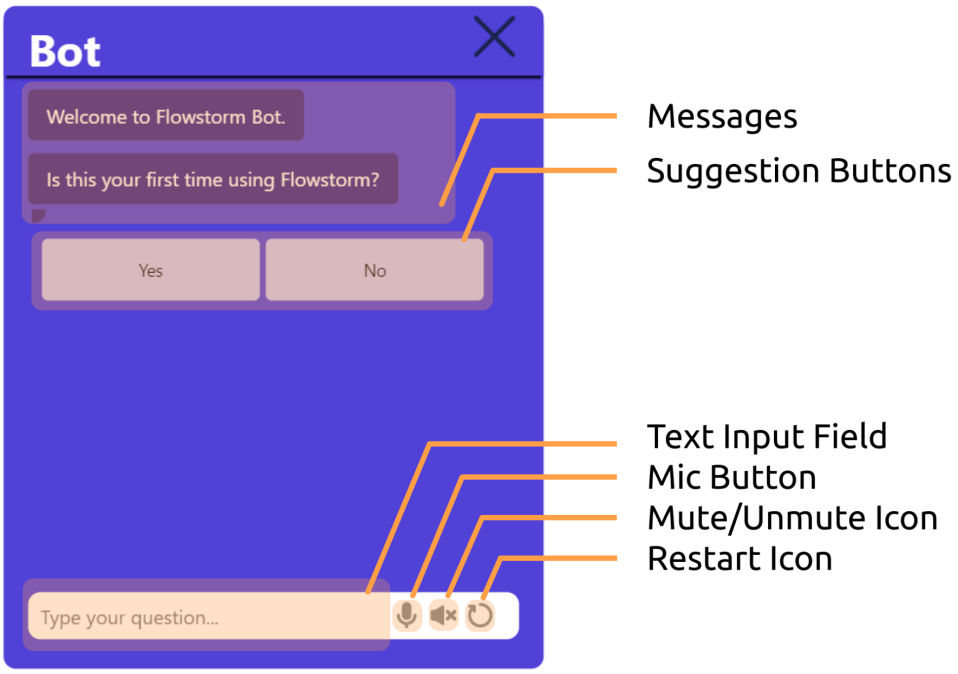


Figure 3.4: Displaying the design of the bot

3.4.2 UI Capabilities

Our UI capabilities extend to displaying a variety of content types, including PDFs, code snippets, Markdown text, and images. This versatility allows us to create rich, dynamic interfaces that cater to different user needs and preferences, whether they are reading documents, reviewing code, or browsing visual content.

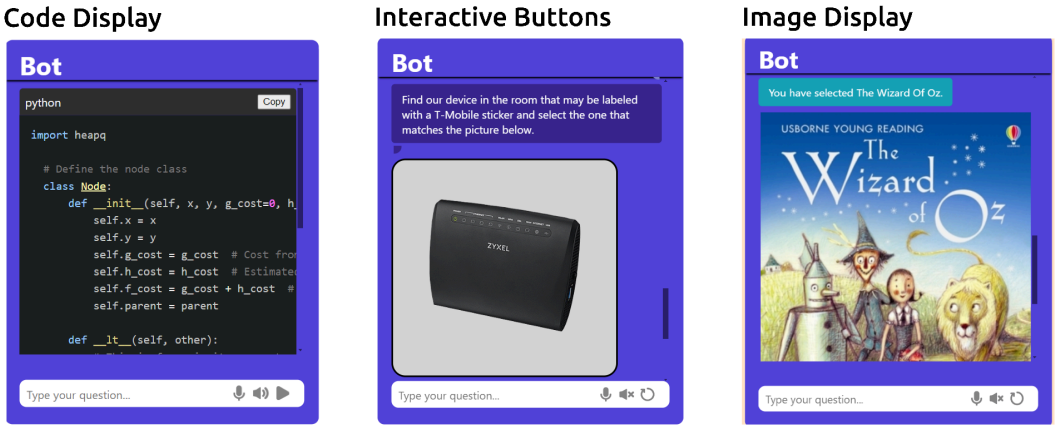


Figure 3.5: Showcasing different UI components of the bot

3.4.3 PDF Support

Our bot has the capability to view PDF files and navigate seamlessly through their pages. Users can easily switch between pages within a document to access specific sections of interest. Additionally, the bot can answer questions related to the PDF's content, guiding users to relevant pages that contain the information they seek. This streamlined functionality enhances the user's experience by making it simple to find and review pertinent data within PDF files.

3.4.4 Code Quality

We use ESLint and Prettier to enhance our code quality and consistency. ESLint helps enforce coding standards and best practices, while Prettier automatically formats code to maintain a uniform style. Together, they ensure our code is clean, readable, and adheres to established guidelines.

Figure 3.6: ESlint error about TypeScript “any“ type usage

3.4.5 Style Protection

Given that our web bot will be embedded in various environments with potentially differing stylesheets and frameworks, such as Bootstrap, it's crucial to safeguard our CSS values to ensure consistent styling and prevent interference from external styles. To achieve this, we've implemented a CSS rule that unsets all properties and applies a standardized font family to our bot's container and its child elements. This ensures that our bot maintains a consistent look and feel across different environments and remains unaffected by external styles or CSS frameworks.

```
.flowstorm-bot-container * {
  all: unset;
  font-family: -apple-system, BlinkMacSystemFont,
  "Segoe UI", Roboto, Oxygen, Ubuntu, Cantarell,
  "Fira Sans", "Droid Sans", "Helvetica Neue", sans-serif;
}
```

Figure 3.7: CSS adjustment to protect the styles of the bot

3.5 Responsive Design

Responsive design is a design methodology focused on creating websites or applications that seamlessly adjust to various screen sizes and devices. It ensures an optimal user experience by dynamically adapting layout, content, and functionality. In our project, we've embraced responsive design principles by implementing specific features tailored to different devices. For instance, we've introduced responsive fonts, which scale up on smaller devices to maintain readability. Moreover, our bot's sizing is responsive, maximizing screen space on mobile devices while remaining dynamic on larger screens. Additionally, users have the flexibility to resize the bot according to their preferences, offering a personalized experience across all devices. These responsive design enhancements contribute to a cohesive and user-friendly interface, catering to the diverse needs of our audience across various platforms and screen sizes.

3.5.1 Responsive Font Sizing

Font size plays a critical role in responsiveness, directly affecting readability across various devices. Adjusting font size based on screen dimensions is essential to ensure content remains clear and legible, enhancing user experience on different devices.

This adjustment of font size for responsiveness is typically achieved through CSS media queries, which allow developers to apply specific styles based on the characteristics of the user's device or viewport. By defining rules within media queries that target different screen sizes or resolutions, such as small screens for mobile devices or larger screens for desktops, developers can dynamically adapt font sizes to maintain readability across a range of devices. For optimal readability and efficient use of space, we should use larger font sizes on smaller devices (mobile phone) and smaller font sizes on bigger devices (PC, Tablet).

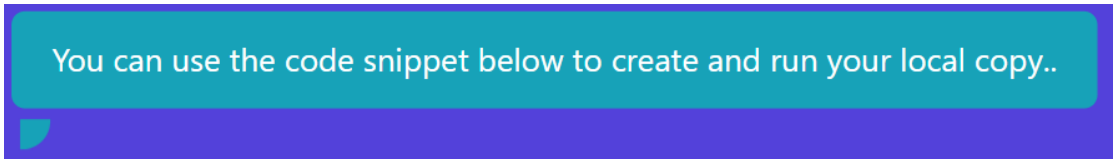


Figure 3.8: Message with smaller font size for bigger devices

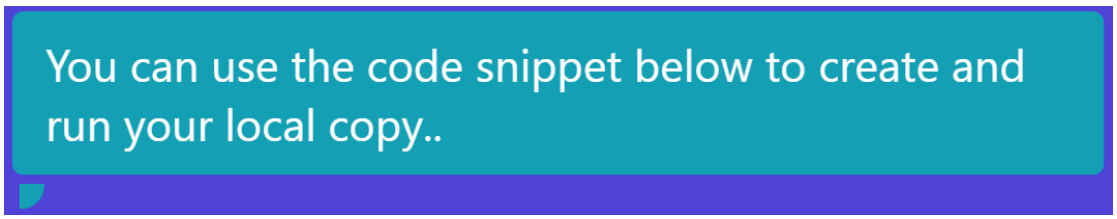


Figure 3.9: Message with bigger font size for smaller device

3.5.2 Responsive Size of The Bot

The sizing of the bot element is dynamically adjusted according to the screen size to optimize user experience across different devices. On smaller screens like mobile phones, the bot element spans the full screen, utilizing the available space efficiently and providing a seamless, immersive experience. This full-screen display ensures that users can interact with the bot comfortably without distractions. Conversely, on larger screens such as tablets or computers, the bot element maintains a dynamic size, adapting to the available space without spanning the entire screen. This approach ensures that the bot remains visually balanced and integrated with other content on the page, offering a responsive and cohesive interface across various devices and screen sizes.

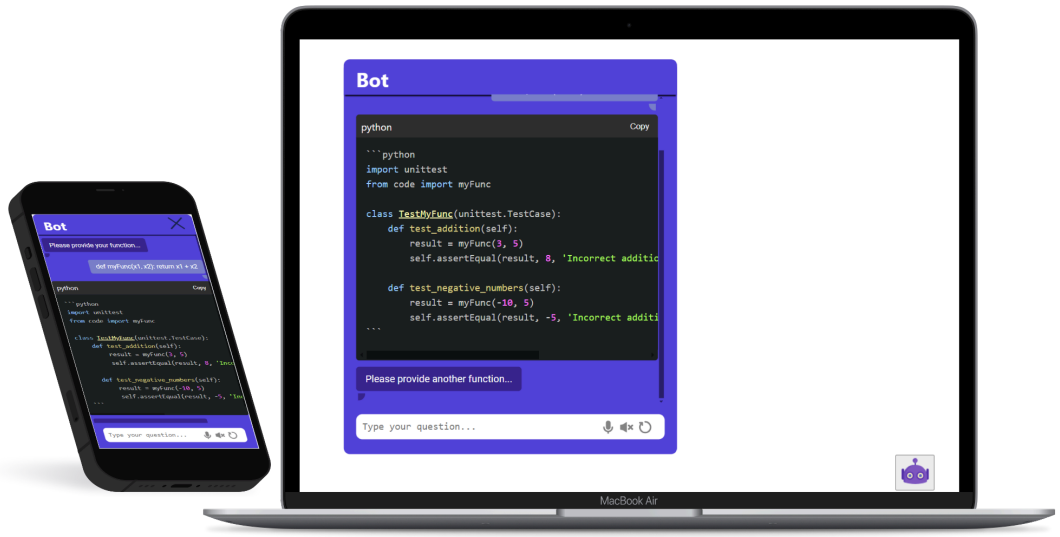


Figure 3.10: Web Bot adjusting itself to different screen sizes

3.6 Bot Configuration

Our application offers a wide range of configuration options, allowing users to personalize various aspects of their experience to suit their preferences. Users can adjust settings related to appearance, behavior, and functionality through our Settings configuration. They can customize the title of the bot, enable or disable sound, specify the initial position of the application, and choose from different themes. Additionally, users can fine-tune color schemes, font sizes, and component sizes to match their aesthetic preferences and optimize readability. The ability to configure custom components further enhances the user's ability to tailor the application to their specific needs. By offering such a diverse array of configuration options, we empower users to create a personalized and enjoyable interaction with our application, ultimately contributing to a positive UX.

Here is a quick rundown of all configuration options. You can also view the README.md file **[15]** to view the full list.

Basic Settings:

- ***Title***: Specifies the title of the bot
- ***Sound***: Enables or disables sound.
- ***Key***: Unique identifier for the application.
- ***Starting Position***: Sets the initial position of the application on the screen.
- ***Resizing***: Indicates whether resizing the bot is allowed.
- ***Dragging***: Indicates whether dragging the bot is allowed.
- ***Input Line Limit***: Determines the maximum number of lines the input field can expand to.

Theme:
Defines the theme of the application (default, dark, or light).

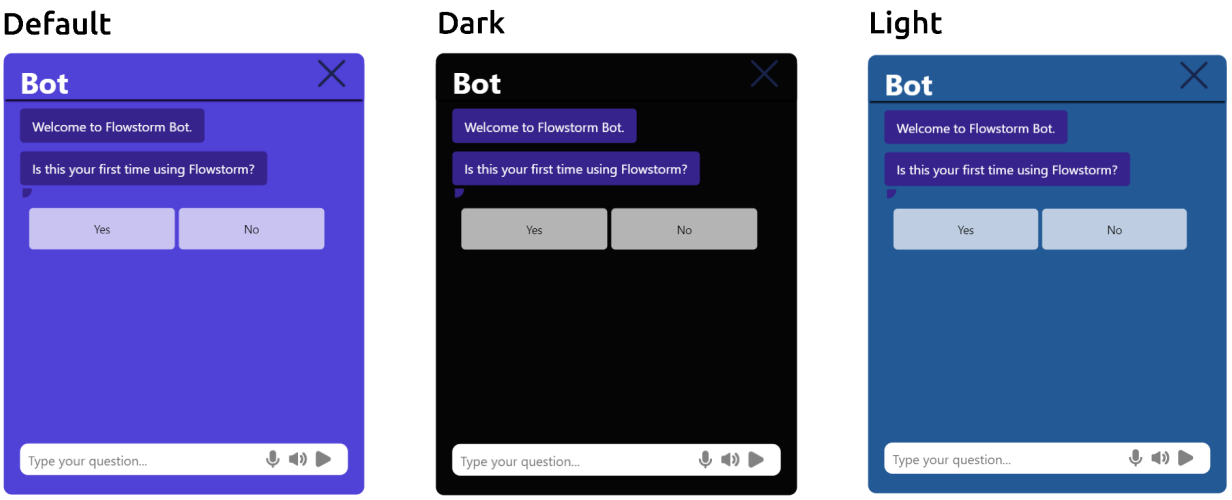


Figure 3.11: Displaying different themes of the bot

Colors:
Allows customization of various color aspects of the application, including bot messages, user messages, background color, title color, icon color, and suggestions.

Font Size:
Sets the font size of the text to small, standard, big, or biggest.

Size Options:
Defines size options for the bot as small, standard, big, or biggest.

PDF Options:
Provides options related to PDF functionality, including PDF ID, PDF scale, and enabling or disabling the file selector functionality.

Custom Components:
Specifies custom CSS for various components such as bubbles, messages, loading bar, suggestions, text input, video, image, button, icon, and bot.
Here is a schema showcasing how the configuration options interact with the components

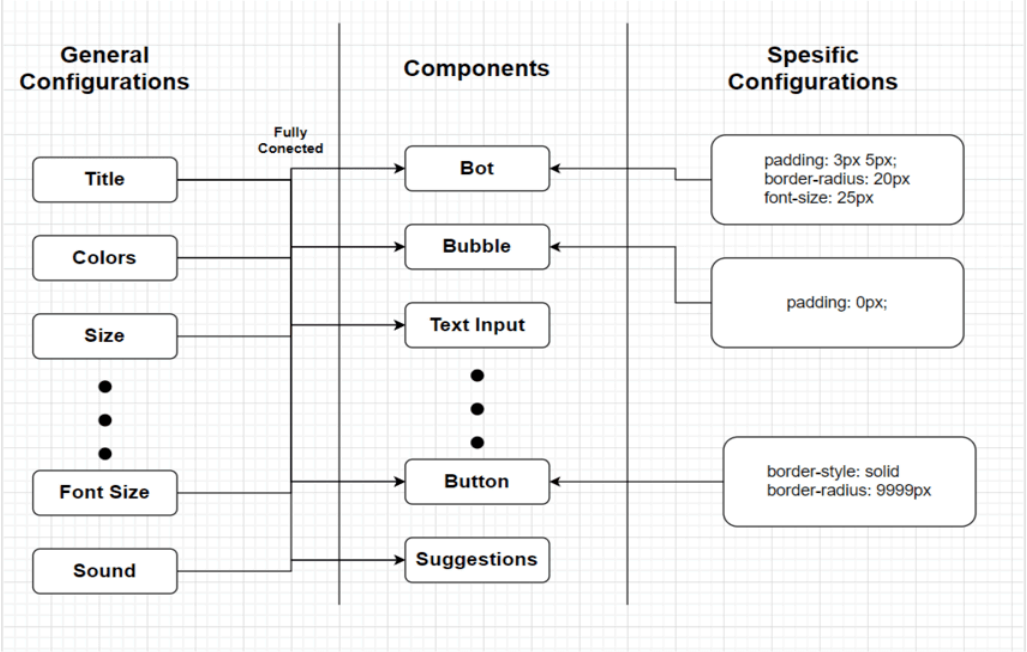
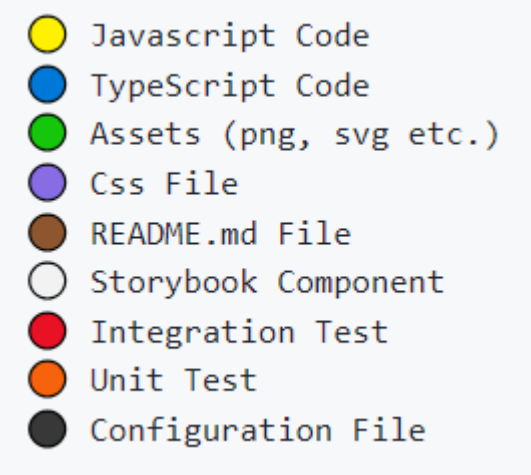


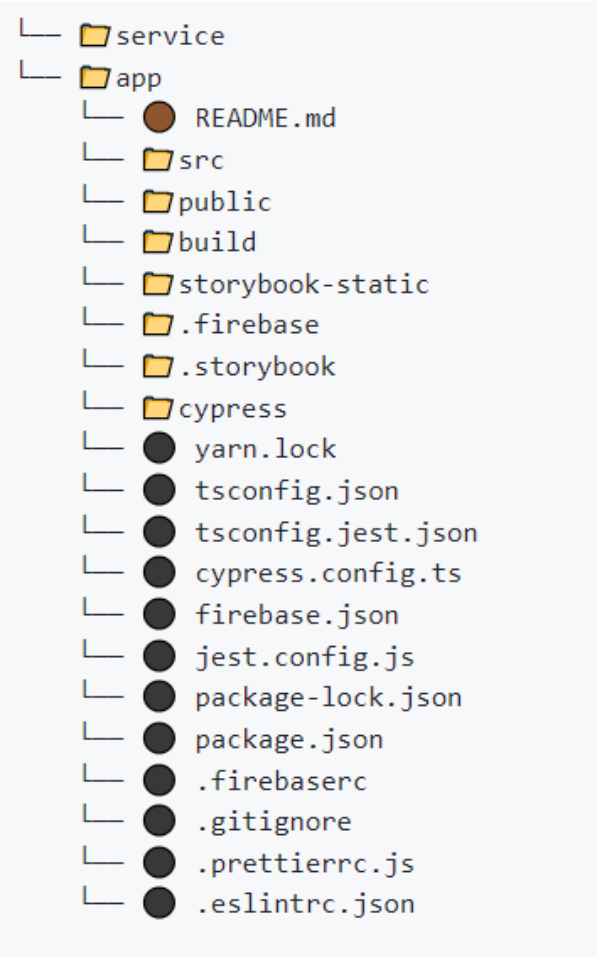
Figure 3.12: Explaining how specific (custom components) and general configuration options interact with the components

3.7 Project Structure



A project structure is an organized arrangement of files and directories within a project to facilitate efficient development and maintenance. In our project, we have two main folders: `**service**`, which is the cloned directory handling backend connections, and `**app**`, which is a React application serving as the frontend.

Figure 3.13: Explaining different kinds of files in the project



The `**app**` directory contains several subdirectories that organize our project files. The `**public**` directory holds static assets like images, index.html, and other files needed for the application to run. The `**src**` directory is where all of our source code resides, including components, utilities, and styles necessary for developing the application. The `**build**` directory is where the production-ready version of our app is stored after running the build process, ensuring that our code is optimized for deployment. Additionally, the `**storybook-static**` directory contains the static files for Storybook.

Figure 3.14: Displaying the root of the project

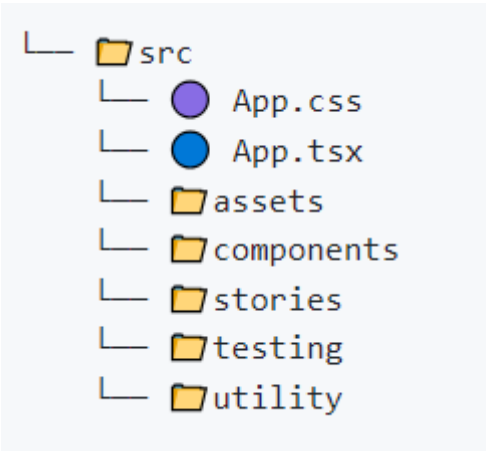


Figure 3.15: Displaying src folder

In the `**src**` directory, you will find the essential files and folders that drive the functionality and appearance of our React application. The main script for React is in `app.tsx`, while `app.css` provides the primary styling for the application. The `**assets**` folder is where we store various media files like images, icons, and other resources used throughout the application. The `**stories**` folder contains our React Storybook stories. All our application components reside in the`**components**` folder, providing the building blocks of our user interface. The `**testing**` folder houses tests for our code. Finally, the `**utility**` folder includes helper functions and utilities that support and streamline other parts of our codebase. Here are the contents of each folder mentioned above.

Assets and Components

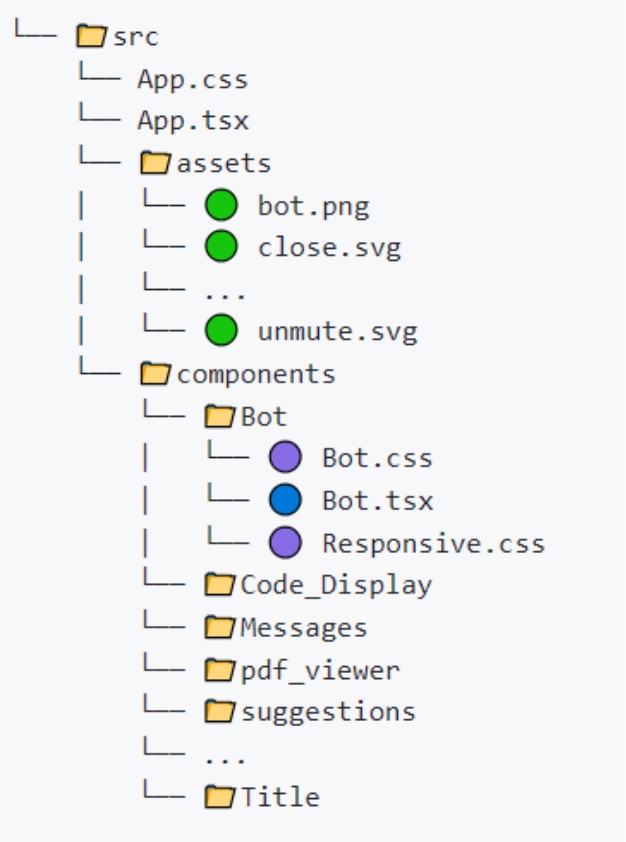


Figure 3.16: Displaying the assets and components directories

Testing directory

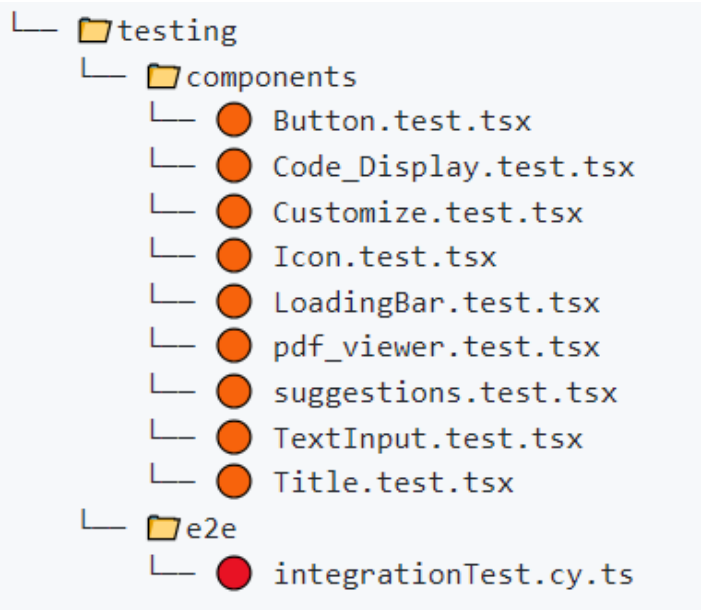


Figure 3.17: Displaying the testing directory

3.8 Flowstorm Server

Flowstorm is a block dialogue-building application that also serves the necessary files for our service connection. It simplifies the process of generating dialogues to meet our customers' needs and provides a visual interface for constructing and managing dialogues.

The Flowstorm server connection is managed through the `service` folder, which was cloned from a pre-existing codebase and is responsible for handling communication between our application and the Flowstorm server. This folder manages text-to-speech (TTS) functions and ensures that dialogue states are correctly transferred to the user interface.

By using Flowstorm script blocks with Kotlin, we can call our API for question-answering capabilities, handling user queries. This approach allows us to keep our API endpoints hidden and perform crucial calls server-side, enhancing security and protecting our backend infrastructure.

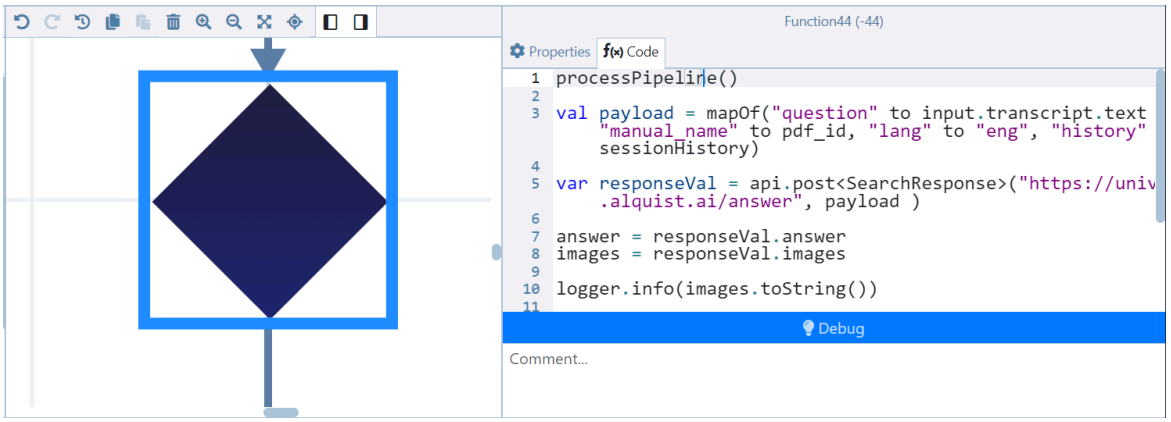


Figure 3.18: Displaying a function block from Flowstorm that calls the question answering API

4. Project Management

Our two-person team operates with defined roles, with me as the lead developer. Using GitLab for version control, we store our code and track project progress through its issue tracking system. Each issue is logged separately, allowing us to keep track of tasks. We employ a systematic versioning strategy, tagging stable bot releases for easy reference and deployment. Integration of CI/CD (Continuous integration / Continuous development) practices automates code integration, testing, and deployment. These practices uphold organization, efficiency, and reliability throughout the project lifecycle.

4.1 Version Control

Version control plays a pivotal role in the development process, and we have chosen GitLab as our preferred platform. GitLab provides us with robust version control capabilities, allowing us to track changes, collaborate effectively, and maintain the integrity of our codebase throughout the development lifecycle. Its intuitive interface and comprehensive set of features enable us to streamline our development workflow and ensure the reliability and stability of our project. The source code for the project can be found here **[16]**

4.1.1 Git Ignore

A *'.gitignore'* file specifies intentionally untracked files that Git should ignore, such as build artifacts, dependencies, cache files, and other miscellaneous files that do not need to be versioned. In our project, the *'.gitignore'* file is set up to exclude dependencies in the *'node_modules'* directory and other related files like *'.pnp'* and *'.pnp.js'* for dependency management. It also ignores testing coverage files and production artifacts such as the *'build'* and *'storybook-static'* directories.

4.2 Task Tracking

GitLab's issue tracking system is instrumental in managing tasks, bugs, and TODOs. Each item is logged separately, ensuring prioritization and progress tracking. To maintain stability, we create dedicated branches for each task, isolating changes and preventing disruption to the main branch. Once tasks are completed and tested, changes are merged back into the main branch, ensuring consistent stability.

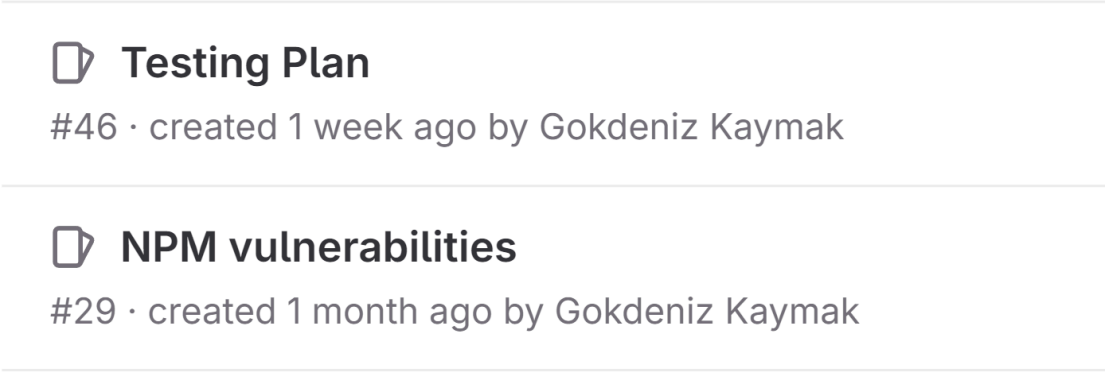


Figure 4.1: Gitlab issue tracking UI

5. Testing

We've adopted a dual approach to testing, incorporating both integration and unit testing methodologies. To facilitate unit testing, we've installed Jest, while for integration testing, we've implemented Cypress. Our testing scripts reside within the src/testing directory, enabling organized and comprehensive testing of our project's functionality and interactions.

5.1 Jest

We define individual test suites tailored to specific components. Within each suite, we orchestrate various scenarios to thoroughly assess the component's behavior, from basic rendering with different props to complex interactions and callback functions. For instance, in the case of our Button component, which serves as a clickable button with either text or images, we conduct tests to ensure proper rendering with both text and image props. Additionally, we verify that clicking the button triggers the expected callback function and that the component behaves appropriately in response to user interactions.

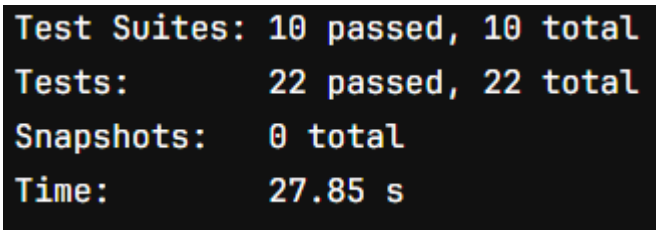


Figure 5.1: Displaying the results of the unit tests, ran on the CI / CD pipeline

5.2 Cypress

Cypress is a robust end-to-end testing framework tailored for web applications, renowned for its intuitive interface and extensive feature set. It provides developers with a seamless testing experience, offering automatic waiting, real-time feedback, and an array of assertion functions. With Cypress, developers can efficiently validate application functionality across various scenarios, ensuring reliability and consistency. Cypress simplifies the testing process by seamlessly integrating with popular JavaScript tools, making it a preferred choice for testing web applications.

5.2.1 Integration Tests

We conduct our tests on the Chrome environment, ensuring compatibility and reliability across this widely used browser platform. Additionally, to facilitate comprehensive testing, we've developed a mock server to simulate the Flowstorm backend, enabling us to evaluate the user interface independently of the server connection. This approach allows us to rigorously test the bot's functionality and responsiveness under various conditions without relying on external dependencies. Here are the tests we run:

1. ***Functionality of Icons***: We test the functionality of icons such as the shrink box toggle, cross icon (to close the bot), mute/unmute icons, play/restart icons, and microphone icon to ensure they perform their intended actions correctly.
2. ***Input Methods Testing***: We validate input methods such as text input and button clicks and typing text into the input field to ensure that the bot responds appropriately to user input.
3. ***Responsive Design Tests***: We assess how the application responds to different device sizes by testing its behavior and layout on various viewport sizes to ensure a consistent user experience across devices.
4. ***Dragging and Resizing Functionality***: We test the dragging and resizing functionality of elements within the application, such as draggable components, to ensure they function smoothly and as expected.
5. ***PDF Capabilities***: We evaluate the application's ability to display PDF files and navigate between pages by testing functionalities like loading PDF files, switching pages, and handling errors related to PDF rendering.
6. ***Code Display and Styling Tests***: We verify the display and styling of code snippets and other styled content within the application to ensure they are rendered correctly and adhere to design specifications.

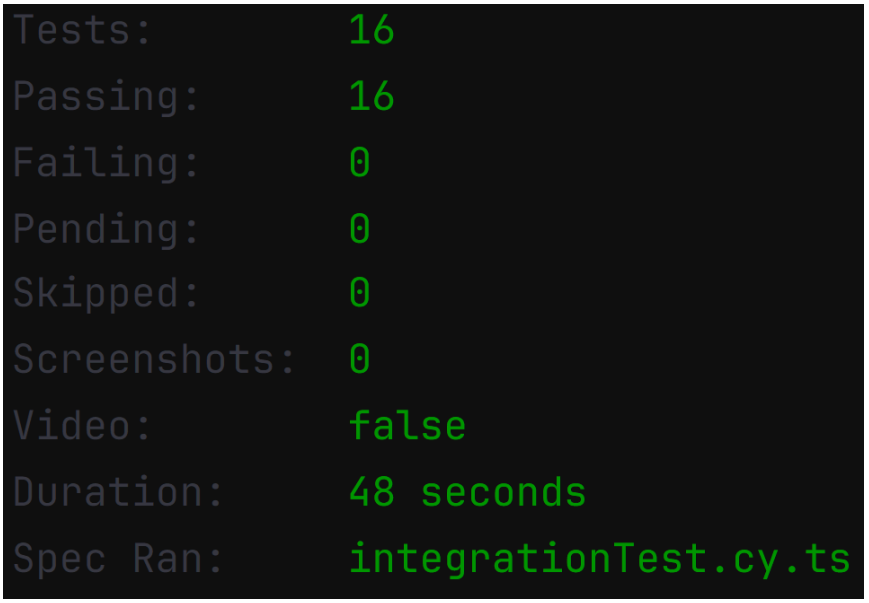


Figure 5.2: Displaying the results of the integration tests, ran on the CI / CD pipeline

5.2.2 Limitations

Unfortunately, due to current limitations, we are unable to test the voice input functionality using Cypress. This functionality requires manual testing or alternative testing methods outside of our automated tests.

5.3 Running The Tests

In our CI/CD pipeline, we execute both integration and unit tests before each deployment to ensure the integrity and quality of our application. Further details regarding our testing procedures in the CI/CD pipeline are provided in the Deployment section.

6. Deployment

Our deployment process for the project comprises several key steps to ensure efficiency and reliability. Firstly, we build the project to prepare it for deployment. Following this, our Continuous Integration/Continuous Deployment (CI/CD) pipeline kicks in, executing thorough testing procedures to validate code changes. Once the testing phase is successfully completed, the CI/CD pipeline deploys the updated code to Firebase. This automated process, facilitated by a YAML file configuration, guarantees that updates are swiftly and reliably delivered while maintaining high standards of code quality.

6.1 Building the Project

In React development, preparing applications for production deployment involves a series of crucial tasks orchestrated by `npm run build`. These include compiling JSX into standard JavaScript, bundling assets for optimized performance, and generating a production-ready build folder.

Here are some important features it has:

1. ***Transpilation***: It transpiles JSX code into plain JavaScript, ensuring browser compatibility and simplifying development.
2. ***Optimization***: It optimizes assets by bundling them together, reducing file sizes and improving load times for enhanced performance.
3. ***Production Build***: It generates a production-ready build folder with minimized and compressed files, suitable for deployment to web servers or hosting platforms.

6.2 CI / CD

GitLab's CI/CD capabilities play a pivotal role in our software development lifecycle. We capitalize on GitLab CI/CD pipelines to automate critical processes, particularly testing, in our workflow. These pipelines, configured through YAML files, offer customizable options to tailor testing procedures according to project needs. Prior to deployment to Firebase, our testing functions, as outlined in the testing section, are executed within GitLab CI/CD pipelines. This ensures that code changes undergo rigorous testing, to maintain quality.

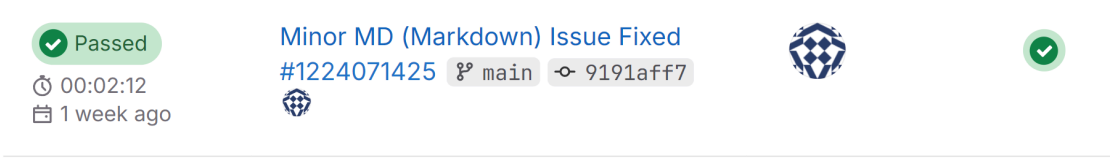


Figure 6.1: Gitlab’s CI / CD interface

6.3 Firebase Hosting

After building and testing our applications, we swiftly deploy them using Firebase Hosting. This ensures rapid and reliable delivery to production environments, supported by features like CDN integration and SSL support. With Firebase, we maintain high availability and deliver top-notch software efficiently. This deployment process marks the completion of our CI/CD pipeline, ensuring seamless integration, thorough testing, and efficient delivery of our applications using Firebase Hosting.

7. Documentation

Documentation is paramount in our codebase, serving as a crucial resource for understanding and maintaining the software. Key aspects of our documentation strategy include utilizing README.md files to provide an overview of the project, including installation instructions, usage guidelines, and important project information. Additionally, we embed code comments throughout the codebase to explain functionality, clarify complex logic, and provide context for future developers. Furthermore, we employ React Storybook Component Documentation methods to create interactive and visual guides for understanding our UI components. This approach accelerates the integration of future developers.

7.1 Readme File

A README file is a text document often found at the root of a software project's directory, providing essential information about the project to users and developers.

In our project, we've crafted a README file to offer guidance and insights into various aspects of our software.

Here are some sections of the README file:

- ***Project overview***: provides a brief summary of the project's purpose, goals, and features.
- ***Install and setup***: provides step-by-step instructions on how to run the code locally on your computer.
- ***Embedding***: explains how to integrate the bot into your website, expanding its functionality beyond standalone use.
- ***Configurations***: offers guidance on customizing the bot to suit specific preferences or requirements, enhancing its adaptability.
- ***Structure overview***: provides insight into the organization and layout of the bot's codebase, aiding developers in navigating and understanding its architecture.

These sections collectively cover a wide range of aspects. The full README.md file can be found here **[15]**

7.2 Comment Blocks

In our project, we've employed typedoc, a documentation generator for TypeScript projects, to automatically generate documentation from comment blocks. Typedoc extracts comments and compiles them into comprehensive documentation, complete with descriptions, parameter details, return types, and more. With typedoc, we can effortlessly maintain thorough documentation alongside our code, facilitating seamless collaboration.

```
/**
 * Adjusts responsive fonts of components based on screen width.
 * @param settings Settings object containing font sizes.
 * @param botRef Reference to the bot container.
 * @param width Current width of the screen.
 */
export const adjustResponsiveFonts = (
  settings: Settings,
  botRef: React.RefObject<HTMLDivElement>,
  width: number,
) => {
```

Figure 7.1: A TypeDoc comment block explaining a function

7.3 Storybook Component Visual Documentation

React Storybook is a powerful tool utilized in our project to create visual documentation for each component, enhancing understanding and facilitating collaboration among developers. By leveraging React Storybook, we can showcase individual components in isolation, providing interactive and visual representations of their functionality, appearance, and variations. This approach enables future developers to visually comprehend each component's purpose, usage, and behavior within the project's ecosystem. With React Storybook, we streamline the process of component exploration and documentation, fostering clarity, consistency, and efficiency in our development workflow. The visual representations of components are also deployed live on the internet. You can check them out here **[17].**

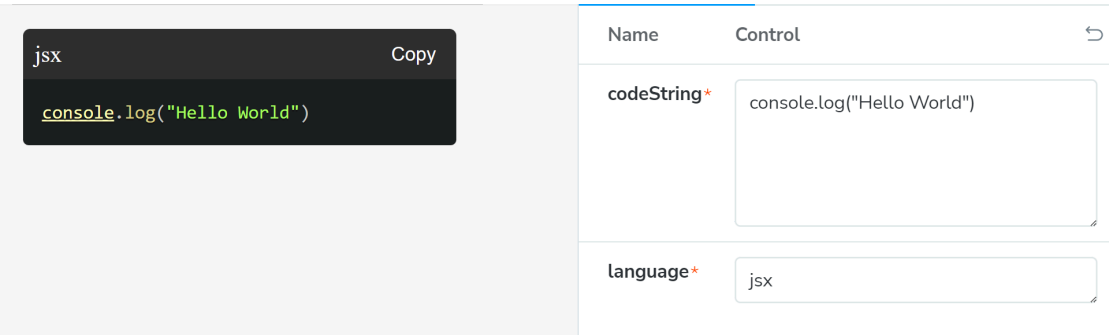


Figure 7.2: Displaying Storybook UI and the control panel of Code Display for a JSX script

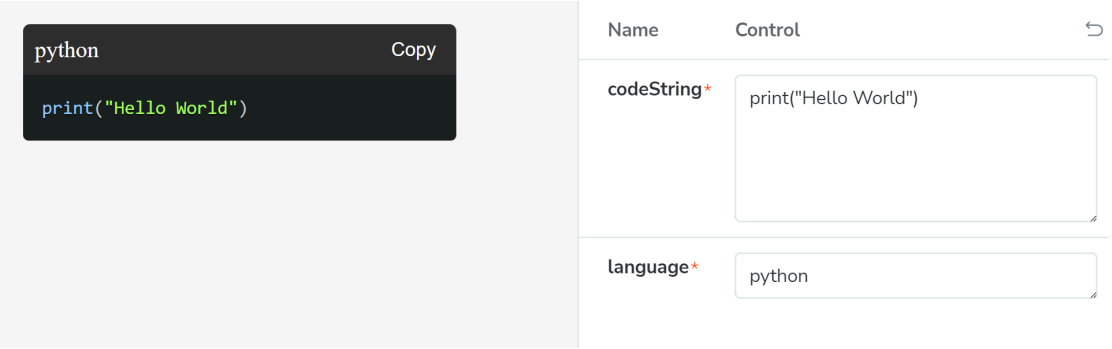


Figure 7.3: Displaying Storybook UI and the control panel of Code Display for a Python script

8. Results and discussion

The implementation of a React-based one-page application for question-answering bots demonstrated successful integration of various input file formats and best practices for user interaction. Comprehensive testing, including unit and integration tests, confirmed the application's efficiency and reliability across different scenarios. React's component-based architecture allowed for a streamlined user interface and enhanced user experience, with Storybook providing valuable visual documentation and testing capabilities.

The application's responsiveness across various devices ensured consistent performance and usability, while extensive configuration options enabled personalized user experiences.

The application is already in use by clients such as UPV and T-Mobile, showcasing its real-world applicability and effectiveness in diverse environments. This adoption demonstrates the software's potential for broader market success and establishes it as a viable solution for organizations seeking to enhance their customer support and internal communication processes.

Despite these successes, further improvements could include providing more customization options and expanding the range of UI components available for displaying customer data. Overall, the research validates the approach taken and provides a strong foundation for further enhancements and applications in the field of question-answering bots.

9. Conclusion

This thesis establishes a solid foundation for the development of React-based one-page applications dedicated to question-answering bots, offering seamless user interaction and efficient integration of various file formats. Looking ahead, as large language models (LLMs) continue to improve, we can replace Flowstorm with an LLM as the dialogue manager, enabling more sophisticated and natural interactions. By leveraging AI, we can generate custom elements tailored to customer needs, granting them unprecedented control over their experience. This versatile web bot holds potential for various applications, such as internal document management, where the bot can read and answer queries from PDFs, or customer support, assisting our clients' customers with their questions about the business. Overall, the application offers a robust platform that can be further enhanced to meet a wide array of user needs.

References

1. David Herbert. (November 13, 2023) ‘What is React.js? Uses, Examples, & More’ Hubspot Blog, [Online].
   Available: https://blog.hubspot.com/website/react-js.
   [Accessed December 17, 2023]

2. Sonoo Jaiswal ‘Pros and Cons of ReactJS’ Java Point [Online].
   Available: https://www.javatpoint.com/pros-and-cons-of-react.
   [Accessed December 17, 2023]

3. Nihar Raval. (November 9, 2023). ‘React vs Angular: Which JS Framework to choose for Front-end Development?’ Radix [Online].
   Available: https://radixweb.com/blog/react-vs-angular
   [Accessed December 18, 2023]

4. Jaydeep Patadiya. (November 9, 2023). ‘Next JS vs React : Which Framework to choose for Front end in 2024?’ Radix [Online].
   Available: https://radixweb.com/blog/nextjs-vs-react
   [Accessed December 18, 2023]

5. Kyle Cook. (July 9, 2022). ‘How To Structure React Projects From Beginner To Advanced’ blog web dev simplified [Online].
   Available: https://blog.webdevsimplified.com/2022-07/react-folder-structure/
   [Accessed December 24, 2023]

6. Pichl, Jan & Marek, Petr & Konrád, Jakub & Lorenc, Petr & Kobza, Ondrej & Zajíček, Tomáš & Šedivý, Jan. (2022). ‘Flowstorm: Open-Source Platform with Hybrid Dialogue Architecture.’ 39-45. 10.18653/v1/2022.naacl-demo.5.
   [Accessed January 23, 2024]

7. Coursera Staff. (November 29, 2023). ‘What Is GitHub and Why Should You Use It?’ Coursera, [Online].
   Available: https://www.coursera.org/articles/what-is-git
   [Accessed December 22, 2023]

8. Sten Pittet ‘The different types of software testing’ [Online]
   Available:
   https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing [Accessed December 28, 2023]

9. Shreerang Kolhe. (September 25, 2018) ‘What is Firebase? The complete story, abridged.’ Medium [Online]. Available:
   https://iamshreerang.medium.com/firebase-a-boon-for-backend-developer-c68957d2369c.
   [Accessed April 16, 2024]

10. OS System. (December 20, 2020). ‘The Ultimate Comparison: Firebase vs Other Platforms’. Available:
    https://os-system.com/blog/comparison-firebase-with-other-platforms/
    [Accessed April 16, 2024]

11. Tim Davidson. (May 8, 2023). ‘The Benefits and Frustrations of Using Storybook’, Clean Commit [Online]. Available: https://cleancommit.io/blog/the-benefits-and-frustrations-of-using-storybook/ [Accessed December 18, 2023]

12. Gökdeniz Kaymak. (October 1, 2023). ‘flowStormBot’, Github Repository. Available: https://github.com/protoss70/flowStormBot/tree/master/lib/service [Accessed December 20, 2023]

13. React. ‘Describing the UI’ *React.dev* [Online]. Available: https://react.dev/learn/describing-the-ui [Accessed April 10, 2024]

14. React. ‘Built-in React Hooks’ *React.dev* [Online]. Available: https://react.dev/reference/react/hooks [Accessed April 11, 2024]

15. Gökdeniz Kaymak. (2024). ‘Readme.md’ Gitlab Resource. Available: https://gitlab.com/alquist/ciirc-projects/client/react-bot/-/blob/main/README.md?ref_type=heads [Accessed April 10, 2024]

16. Gökdeniz Kaymak. (2024). ‘react-bot’. GitLab Repository. Available: https://gitlab.com/alquist/ciirc-projects/client/react-bot [Accessed April 21, 2024]

17. Gökdeniz Kaymak. (2024). ‘Components / Bubble - Bot Message · Storybook’ *react-bot-storybook.web.app* [Online] Available: https://react-bot-storybook.web.app/ [Accessed April 21, 2024]