

UNIVERSITY OF ZAGREB

**FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING**

BACHELOR THESIS ASSIGNMENT No. 309

**SYSTEM FOR WRITING AND ORGANISING  
PROGRAM DOCUMENTATION**

Fran Jurinec

Zagreb, May 2021

## **BACHELOR THESIS ASSIGNMENT No. 309**

Student: **Fran Jurinec (0036516020)**  
Study: Electrical Engineering and Information Technology and Computing  
Module: Computing  
Mentor: assoc. prof. Igor Mekterović

Title: **System for writing and organizing program documentation**

Description:

Make an overview of common functionalities and patterns in systems for writing and managing program documentation (eg Gitbook, Docusaurus, VuePress). Create a web application that will allow the user to write program documentation and subsequently generate static web pages for that documentation. Documentation is provided in the markdown format, where it is necessary to enable more advanced properties such as images, formulas, tables, etc. The web application must allow multiple users to work on common documentation, and in addition to managing user accounts, it is necessary to provide support for simultaneous work which can result in multiple versions of the documentation. The user needs to be able to customize the structure and layout of the generated web pages to some extent. Implement the search facilities in both phases - when writing documentation and in the generated pages. Consider the distribution of the web application in the form of an Electron application. Provide an assessment of the approach taken and guidelines for future development.

Submission date: 11 June 2021

Zahvaljujem se svom mentoru, prof. dr. sc. Igoru Mekteroviću,  
na savjetima i pomoći pri izradi ovog završnog rada.

# Contents

Introduction .....	1
1 Documentation Tools .....	2
1.1 GitBook .....	2
1.2 Docusaurus.....	4
2 User Requirements .....	6
3 Used Technologies .....	9
3.1 Core Client Technologies.....	9
3.1.1 Node.js .....	9
3.1.2 Electron.js.....	10
3.1.3 TypeScript .....	12
3.1.4 React.js .....	13
3.2 Extended Client Technologies .....	14
3.2.1 Redux (react-redux).....	14
3.2.2 Markdown-it & plugins .....	15
3.2.3 Tailwind CSS.....	16
3.2.4 Pug.js .....	17
3.3 Client Technology Overview .....	18
3.4 Remote Host Technologies.....	19
3.4.1 Node.js .....	19
3.4.2 Express.js.....	19

3.4.3	JWT (JSON Web Token).....	19
4	Solution Overview .....	20
4.1	Visual Identity.....	20
4.1.1	Name.....	20
4.1.2	Color scheme .....	20
4.1.3	Logo and branding.....	23
4.2	Client Architecture.....	24
4.3	Core Client Functionality .....	26
4.3.1	Project Management Screen .....	26
4.3.2	Main Editor .....	28
4.3.3	Search .....	30
4.3.4	Static Export .....	31
4.4	Server Architecture and Functionality .....	33
5	Reflection and further development .....	36
	Conclusion .....	37
	References.....	38

# Introduction

Writing project documentation is a key element of any software development project. As a key source of truth for the project, documentation is often a collaborative process which involves the various members of both the development team as well as administration. A project's documentation can serve many purposes, and a single project can rely on multiple types of documentation. For example, technical documentation can guide the development process and help developers keep the code base consistent and in line with established principles and methodologies implemented in that project. On the other hand, customer-oriented documentation can serve as a beginner guide for onboarding new customers, and as a reference for existing users of the product.

The goal of this bachelor thesis is to develop a tool which will assist the creation and management of project documentation. The tool would allow the users to manage multiple project's documentations from a single interface. The tool would also include a component for remote hosting of documentation, which would allow easy collaboration and concurrent access to the documents. To allow easy distribution of a specific version of the documentation, the tool will also allow users to export a static, read-only, version of the documentation. This simple export will be cheap and easy to host. To increase compatibility for a larger user base, the tool will be developed as a cross-platform computer program, which can operate on Windows, Linux and MacOS. The documentation will be written in Markdown format, which is a popular syntax for writing formatted text, already used in many existing project documentations.

The remote host will be a simple hosted solution which will allow users to host their own collaborative project in a remote location. Most likely, it will be created in the form of a Node.js project, which is flexible and easy to deploy. The main goal of the remote host is to allow for remote and concurrent work on the documentation.

# 1 Documentation Tools

Documentation tools, for the lack of a more common term, refers to a variety of software solutions which allow individuals and groups to create and manage their documentation more easily. These tools generally allow for the creation and editing of rich formatted documents, displaying them in an organized manner, and easy navigation.

## 1.1 GitBook

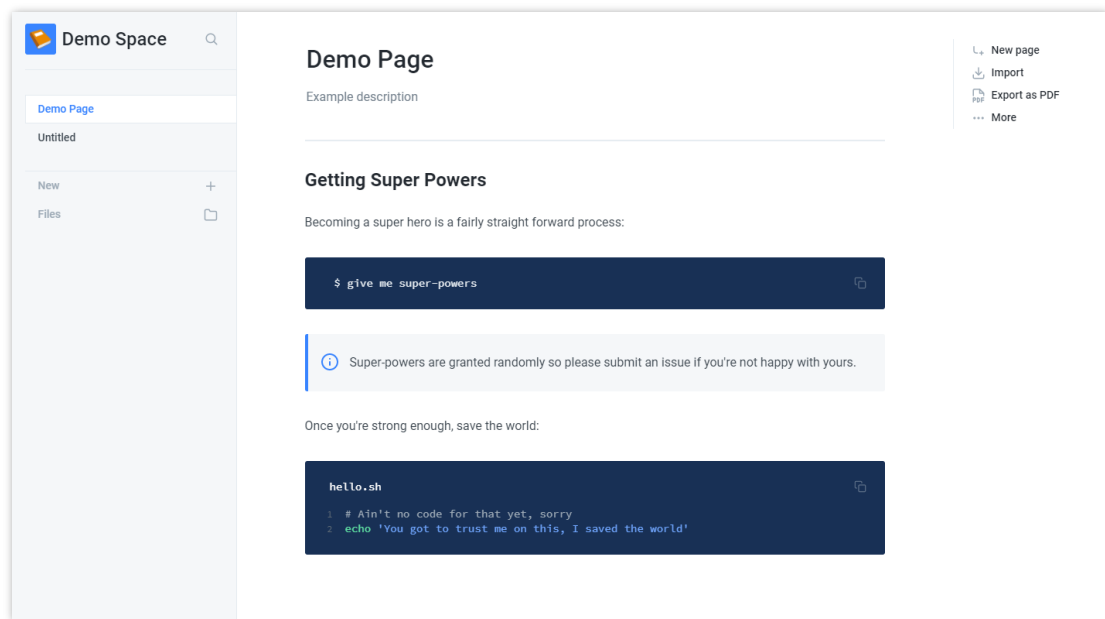
Among the most popular documentation writing solutions today is GitBook. GitBook is a web application which allows the development of documentation in the cloud [1].

GitBook allows users to create separate projects, which can be private or public. Collaboration is smooth, and changes are committed in a git-like fashion, where each change must be first saved then merged into the project. Documents are edited in their formatted form. Creating headings and other rich elements can be achieved either through the user interface, or by typing the expected markdown, roughly adhering to CommonMark specifications [2]. The main editor is shown in Picture 1.1.

---

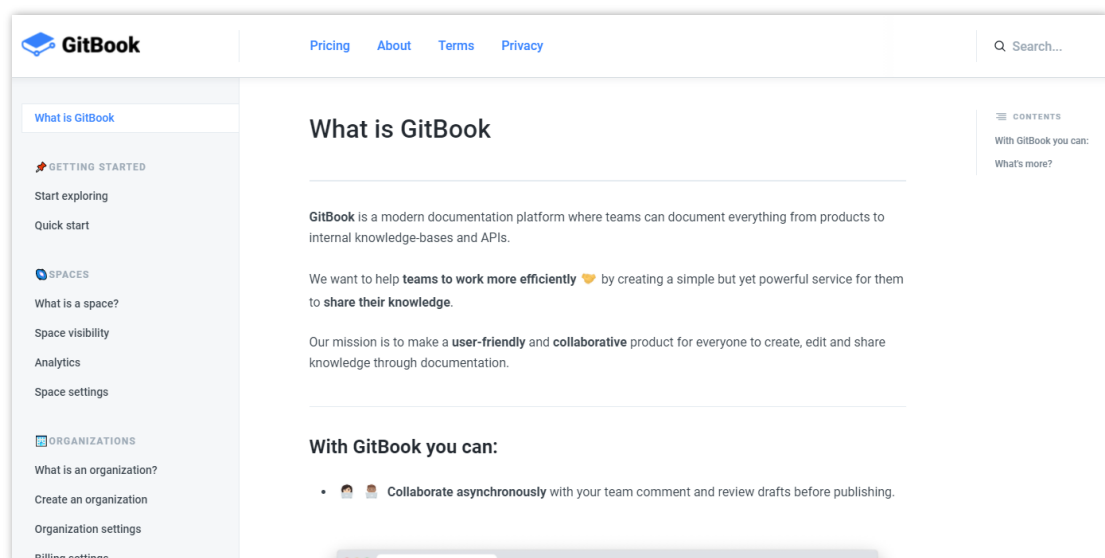
<sup>1</sup> GitBook Documentation, *What is GitBook*, <https://docs.gitbook.com/>, 10.06.2021

<sup>2</sup> GitBook Documentation, *Markdown*, <https://docs.gitbook.com/editing-content/markdown>, 10.06.2021



Picture 1.1 GitBook Editor GUI

GitBook allows you to publish your documentation in the form of a publicly accessible web page, however limiting the visibility and controlling access to it are among the paid features. Free users can only choose between completely private or public documentation instances. The read-only documentation generated by GitBook can be seen in Picture 1.2.



Picture 1.2 GitBook Generated Webpage <sup>[3]</sup>

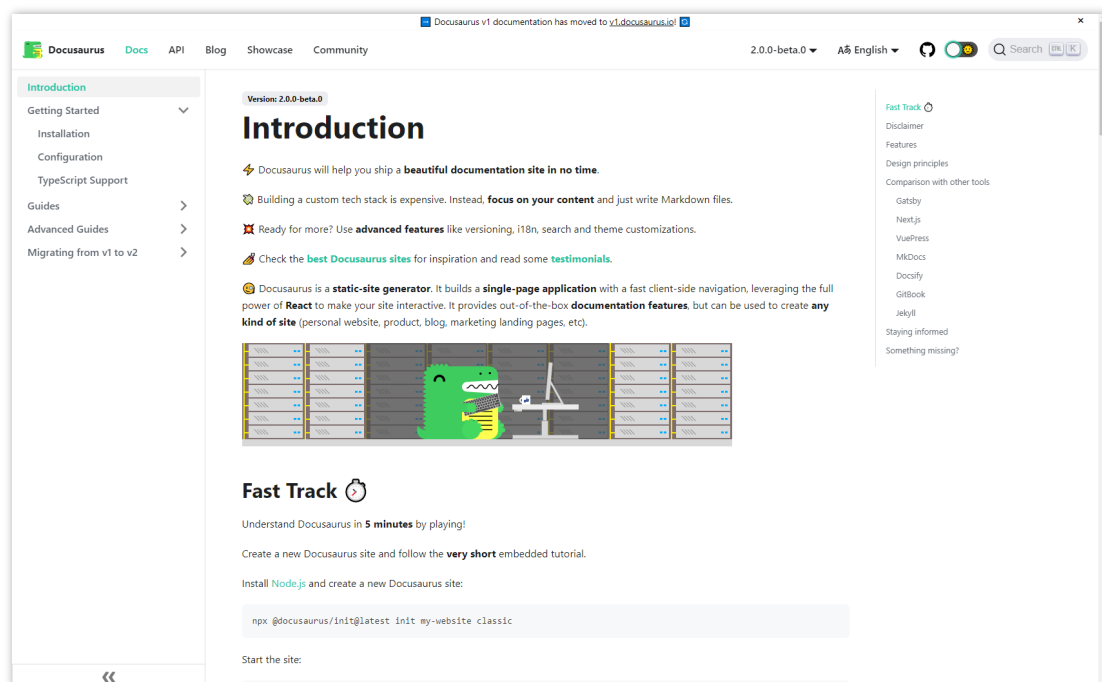
<sup>3</sup> GitBook Documentation, *What is GitBook*, <https://docs.gitbook.com/>, 10.06.2021



## 1.2 Docusaurus

While GitBook has a large focus on ease of use and high-level features, it forces the user to rely on their service for hosting the contents. Some features are also limited by a paywall. On the other hand, Docusaurus is a completely free and open-source solution developed by Facebook Open Source. Docusaurus is a much more robust tool, focused more on the display and navigation of the documentation, than on the process of creating it.

In Docusaurus' own words, it is a static-site generator which uses React to build a Single Page Application (SPA) with out-of-the-box documentation features [4].



Picture 1.3 Docusaurus Page Interface

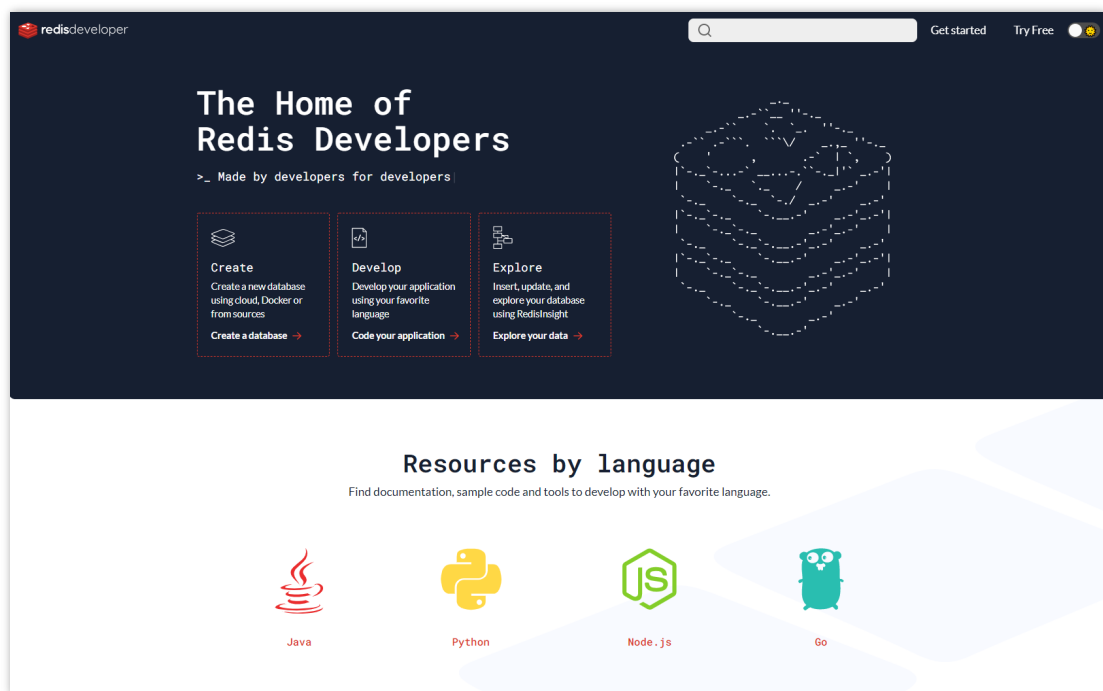
Docusaurus pages must be created manually as markdown documents in the user's editor of choice, and then Docusaurus must be configured to correctly display these documents as a web page. While the configuration of Docusaurus requires the users to possess significantly more technical

---

<sup>4</sup> Docusaurus, *Introduction*, <https://docusaurus.io/docs/>, 10.06.2021

knowledge than GitBook, it also allows for more customization through plugins, as well as offering a more structured website for browsing the documents, with a dedicated landing page, advanced search, and tabs to give more structure to the project. A general document display interface can be seen in Picture 1.3.

As mentioned, when configuring Docusaurus, users will have to manually configure many things, this can include technical work such as configuring a variety of settings in “.json” configuration files, as well as write some custom elements in JavaScript and React. However, the amount of customization is nearly unlimited. An example of a highly customized landing page built with Docusaurus can be seen in Picture 1.4.



Picture 1.4 Docusaurus powered Landing Page <sup>[5]</sup>

---

<sup>5</sup> Redis Developer Hub, *The Home of Redis Developers*, <https://developer.redislabs.com/>, 10.06.2021

## 2 User Requirements

The product, which was later named “Documint”, is comprised of the main desktop application and an optional remote server. The main application treats all users equally, while the server requires an additional person responsible for its configuration.

This system recognizes two different user roles:

- User – standard application user
- Remote Host Administrator – person responsible for configuring the remote server

Because the solution is a desktop application, any user has full access to all its features, and as such, no additional roles are introduced. The basic overview of client functionality is shown on Diagram 2.1.

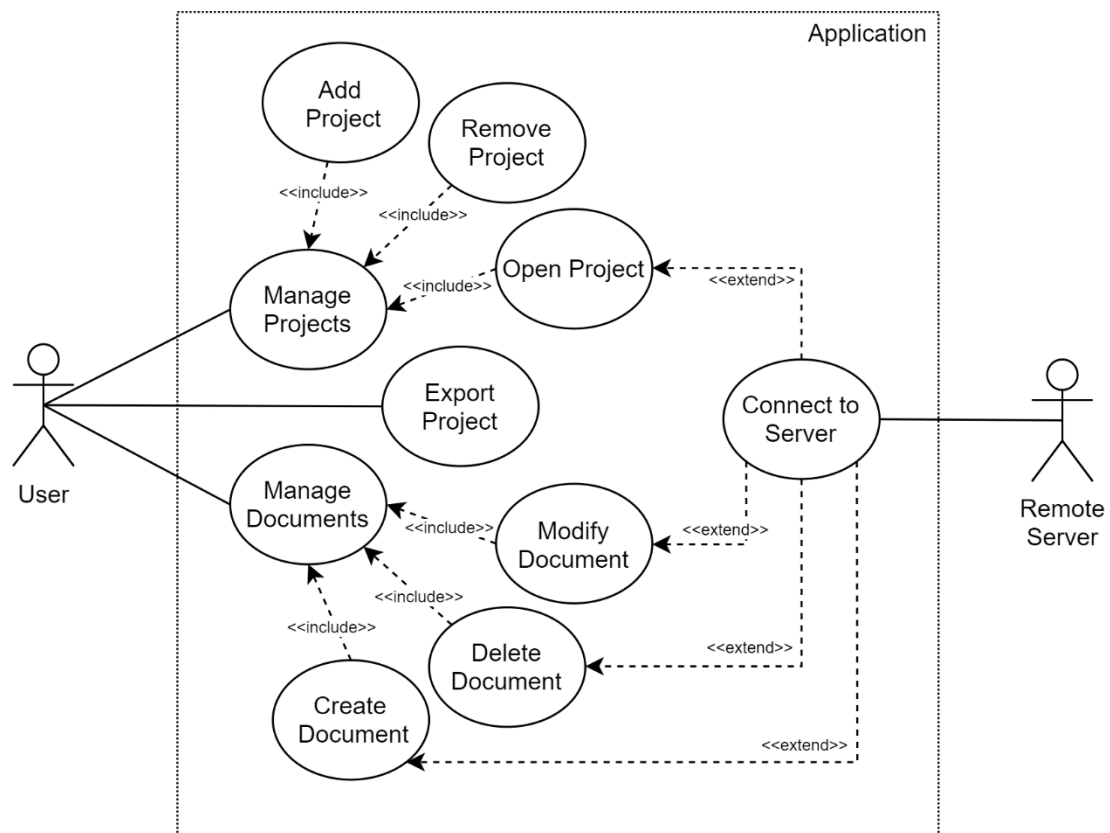


Diagram 2.1 User Use Case Diagram

The main functional requirements for the user are the following:

1. Adding and removing projects from the main menu
2. Browsing the contents of local or remote projects
3. Creating and deleting documents in a project
4. Editing documents using the built-in editor
5. Publishing the project as a static website

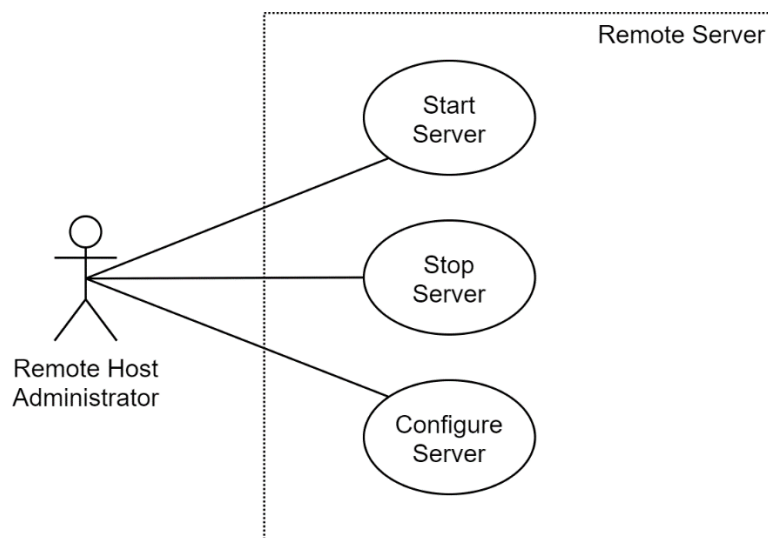
Extended functional requirements also include:

1. Performing content-aware search queries to find documents
2. Customizing the look of the exported website

The user must have the ability to completely develop the documentation within the application. This means that managing projects and documents must be possible and easy to perform. It also means that the application must contain a functional built-in editor which allows the user to write markdown syntax documents which are then rendered by the app. Finally, once the documentation is ready for publishing, the user must be able to perform an export of the project into a standalone static website. To navigate between documents more easily, a content-aware search should also be included. Exporting the project should allow the user to specify a certain degree of custom properties to tailor the exported website to their needs.

The remote server is another key element of the system for users who wish to collaborate on a project. The remote host administrator, the person configuring a project's remote server must be able to perform the following basic tasks as shown in Diagram 2.2:

1. Configuring the remote server (port number and access password)
2. Starting and stopping the server



*Diagram 2.2 Remote Host Administrator Use Case Diagram*

In terms of configuration, the server has to enable the Remote Host Administrator to define the listener port as well as the password used to access the documentation project. The server also has to prevent unauthorized access for users without the password.

## 3 Used Technologies

To meet the goal of cross-platform use, the program was developed using Node.js and Electron. This means that the client application is developed using a web technology stack for easier cross-platform deployment. The packaging of the application as a desktop app allows it to function without depending on any remote service, making it completely functional for users without incurring any additional costs from webhosting or otherwise.

The standalone app can be expanded using a simple web server. The server is also developed on top of Node.js, and it is using Express.js for simplicity reasons.

### 3.1 Core Client Technologies

#### 3.1.1 Node.js

Node.js is an asynchronous event-driven JavaScript runtime, enabling JavaScript execution outside of a browser <sup>[6]</sup>. It also includes API calls which access some lower-level native features. It is a de-facto industry standard for JavaScript app development.

Node.js serves as the foundation for all other components and frameworks used in the project.

---

<sup>6</sup> Node.js, *About*, <https://nodejs.org/en/about/>, 10.06.2021

### 3.1.2 Electron.js

Electron is an open-source framework maintained by GitHub [7]. It allows the development of cross-platform desktop applications using the traditional web stack of HTML/JavaScript/CSS. The Electron framework renders its contents using Chromium, a lightweight open-source version of the Google Chrome browser, under the hood. The key element in this process is that Electron exposes Node.js functionality to the client-side code running in Chromium. This allows for a single layer of JavaScript which can access both the UI elements and the DOM, as well as native low-level features, allowing the development of a complete desktop application.

The Electron runtime is separated into two process categories. The first of which is a single “main” process which has access to additional low-level electron functions. This process is used to run and control multiple instances of the child “renderer” process, as well as the Chromium instances.

On the other hand, the “renderer” process is much like the classic “window” of a desktop app. One renderer process is connected to each application window, allowing them to operate asynchronously from the main process. It also has access to the complete set of Node.js functions, as well as limited access to Electron functionality.

The processes can communicate between each other in a synchronous or asynchronous manner using a standardized form of Inter Process Communication (IPC). The communication relies on “main” and “renderer” processes’ event emitters and receivers. This IPC communication can be used to query for low level data from the main process.

---

<sup>7</sup> Electron, *Build cross-platform desktop apps with JavaScript, HTML, and CSS*, <https://www.electronjs.org/>, 10.06.2021

Once the app is loaded, the main process script is executed, and it is used to open a new window, and rendering specified content. Usually, the first item rendered by the main script is the root document which loads all the other necessary components of the application. In this project, the main entry point to the rest of the web app is a static index.html document. This HTML document includes the Electron renderer script, which is then executed inside of the browser. Once index.html is loaded, the rest of the application logic and rendering is handled through in the renderer process. The basic structure of the Electron app architecture can be seen on Diagram 3.1.

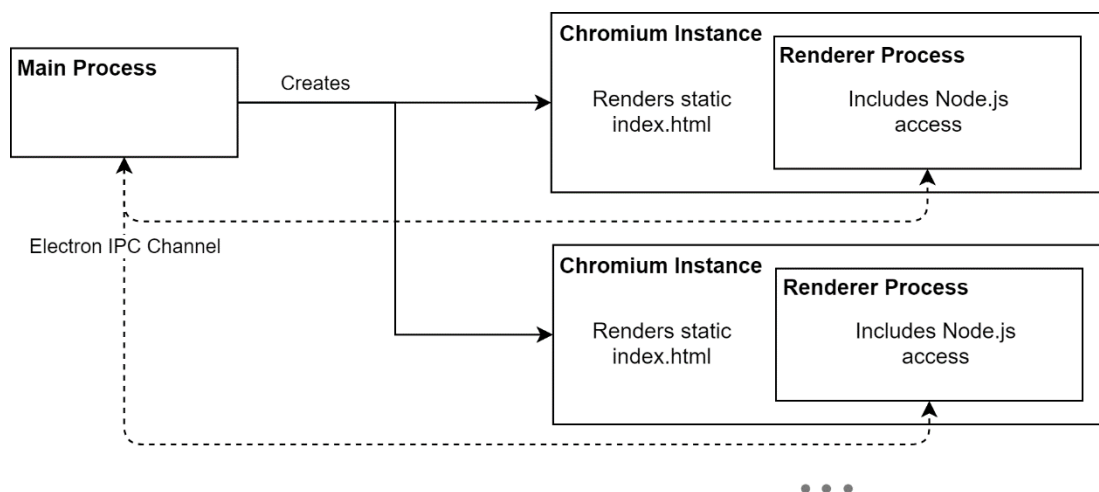


Diagram 3.1 Electron App Architecture



### 3.1.3 TypeScript

An increasingly popular alternative to writing pure JavaScript is TypeScript. Developed by Microsoft, TypeScript is an open-source language offering a more verbose syntax for writing type-safe web code [8]. This project is using TypeScript to keep the development process more streamlined and to reduce the possibility for runtime errors. TypeScript's rich IntelliSense allows for a richer overview of a project's source code. Instant insight into available functions and variables, as well as the expected function parameter and return types allow for arguably faster development.

It is important to note, however, that TypeScript must be compiled to JavaScript before interpretation by the Node.js runtime. For this purpose, the project relies on the official TypeScript Compiler (TSC), although other options such as Babel are available. This choice was made to reduce additional overhead, as TSC already offers React (.tsx) compiling out of the box.

Because of the required compilation step, the 'build' and 'start' scripts have been created in the package.json (the Node.js project descriptor file) to make compilation and starting of the app easier.

---

<sup>8</sup> TypeScript, *Typed JavaScript at Any Scale*, <https://www.typescriptlang.org/>, 10.06.2021

### 3.1.4 React.js

React is one of the most popular UI development frameworks, and it was ranked 2<sup>nd</sup> among all web frameworks in the Stack Overflow Developer Survey 2020 [9]. Relying on TSC allows easy use of React, as no additional compiler is necessary. React files are labeled with the “.tsx” extensions and are compiled into executable JavaScript by the TSC. React allows the creation of a responsive and stateful UI components. These components are written using the TSX syntax (TypeScript alternative of JSX, the standard React syntax), which is a combination of TypeScript and HTML source code. React typically offers two types of components, class-based components, and functional components. This project relies on functional components, as they offer a cleaner source code, and they have excellent integration with TypeScript.

As briefly mentioned earlier, React is integrated into the application by being executed in the Electron renderer process. Being rendered directly in the browser allows React to directly render its components onto the DOM of the index.html document rendered by Electron on startup.

A single function call is used in the renderer script, which renders the <App> React component inside of the “root” element present inside of index.html. This script is shown in Code Snippet 3.1.

```
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>,  
  document.getElementById('root')  
)
```

*Code Snippet 3.1 Electron Renderer React Call*

This script renders the React application, and delegates all further rendering logic onto the React framework. The <Provider> tag is explained in 3.2.1.

---

<sup>9</sup> Stack Overflow, *Stack Overflow Developer Survey 2020*, <https://insights.stackoverflow.com/survey/2020>, 10.06.2021

## 3.2 Extended Client Technologies

### 3.2.1 Redux (react-redux)

Normally, React states are limited to the scope of a single component, making the transfer of information between different. Redux is the answer to this problem. Redux offers the ability to store a global state, serving as a source of truth to all components that need it <sup>[10]</sup>. Accessing the global state is done via selectors, which can gain read-only access to the select variables that they need. An example of selectors can be seen in Code Snippet 3.2.

```
const currentProject =
  useSelector<GlobalState,GlobalState["currentProject"]>(state =>
    state.currentProject)
const currentFile =
  useSelector<GlobalState, GlobalState["currentFile"]> (state =>
    state.currentFile)
```

*Code Snippet 3.2 <ProjectDisplay> Selectors*

Modifying the global state is performed through an event-based system of actions which are dispatched from various React components and executed asynchronously by the Redux handler. An example of a dispatched action and its reducer can be seen in Code Snippet 3.3.

```
// From src/react/components/ProjectSelector.tsx - action is dispatched
const dispatch = useDispatch()
...
dispatch(setOpenProject(project))

// From src/redux/reducer.ts - action is received and state is updated
const projectReducer = (state: GlobalState = initialState, action: Action) => {
  ...
  case "SET_OPEN_PROJECT":
    return { ...state, currentProject: action.payload }
  ...
}
```

*Code Snippet 3.3 Dispatch Example*

---

<sup>10</sup> Redux, *Redux*, <https://redux.js.org/>, 10.06.2021

This set of accessors and dispatched actions offers a very robust and structured system for global state management. It also allows effortless and safe sharing of information between separate React components.

### **3.2.2 Markdown-it & plugins**

Markdown-it is a fast and powerful Markdown to HTML compiler <sup>[<sup>11</sup>]</sup>. It is used to convert source markdown content into rendered HTML. The markdown-it library follows the CommonMark specification for interpreting markdown, however it can also be extended using plugins, which often follow the extended pandoc specification.

---

<sup>11</sup> GitHub, *markdown-it*, <https://github.com/markdown-it/markdown-it>, 10.06.2021

### 3.2.3 Tailwind CSS

The first CSS framework used in the development of this project was SASS. While SASS is a powerful framework for developing modular and more dynamic styles, it did not fit the constraints of this project. It still relied on custom styling for all components, and with the limited time frame for this project, it was drastically slowing down the development process.

The replacement was Tailwind CSS, a lightweight and powerful utility-first framework for rapid UI development [12]. Tailwind completely removes contact with CSS from the development process, instead relying on a system of modular classes which can be used to quickly style HTML elements. In addition, it offers out of the box support for responsive styles, and a very clean set of styles. An example of Tailwind styles applied in React components using the “className” property can be seen in Picture 3.1

```
/* List saved projects */
<div className="flex flex-col gap-2 justify-center xl:border-l border-mint pl-10 py-10 w-96">
  <div onClick={() => setShowAddProjectModal(true)}
    className="p-3 bg-mint rounded font-bold text-white text-center hover:bg-opacity-60 active:bg-
    <i className="fas fa-plus mr-1"></i>Add Project
  </div>
  {projects.map((project, index) => {
    return (
      <div className="flex flex-col p-3 gap-1 bg-gray-200 rounded hover:bg-opacity-60 active:bg-op
        key={index}
        onClick={() => onOpenProject(project)}>
          <div className="flex flex-row justify-between gap-2">
            <div className="flex-grow truncate">{project.name}</div>
            <div className="text-xs font-semibold">{project.type}</div>
          </div>
          <div className="flex flex-row justify-between gap-2">
            <div className="flex-grow text-xs truncate">{project.path}</div>
            <div className="text-gray-500 hover:text-gray-700"
              onClick={(e) => {e.stopPropagation(); removeFromProjectsFile(project).then(updateP
            </div>
          </div>
        </div>
      )
    )
  })}
</div>
```

Picture 3.1 Tailwind Styling in React

---

<sup>12</sup> Tailwind CSS, *Rapidly build modern websites without ever leaving your HTML*, <https://tailwindcss.com/>, 10.06.2021

### 3.2.4 Pug.js

Pug.js is a simple and lightweight templating engine [13]. It was a late addition to the project, serving as the main framework through which the exported static website is generated. Pug allows the project to have a single defined template for an HTML page, and the function to generate the final HTML can be pre-processed so that multiple pages using the same template and variables can be generated very quickly.

Pug syntax is also very simple, it is essentially a simplified version of the HTML syntax with additional support for code elements such as variables and loops. It is also modular and supports imports and mixins, making it a very powerful tool overall. An example of the Pug.js syntax can be seen in Picture 3.2.

```
div(class="flex flex-row justify-between gap-1 w-full items-center text-lg pb-2 mb-4 border-b □border-gray-900")
  div(class="font-medium select-none") #{projectTitle}
  div(class="■text-gray-400 □hover:text-gray-600" id="search-button")
    i(class="fas fa-search")

include searchContext.pug

each fileName in uncategorisedFileNames
  a(href=fileName+'.html')
    div(class="flex flex-row gap-1 select-none w-full items-center □hover:text-gray-400 truncate") #{fileName}

each element in categoryFileNames
  div(class="w-full border-t mt-2 ■border-gray-300")
    div(class="pt-4 pb-2 w-full font-semibold select-none") #{element.category}
    for fileName in element.fileNames
      a(href=fileName+'.html')
        div(class="select-none □hover:text-gray-400 flex-grow truncate") #{fileName}
```

Picture 3.2 Pug.js Example

---

<sup>13</sup> Pug, *Getting Started*, <https://pugjs.org/api/getting-started.html>, 10.06.2021

### 3.3 Client Technology Overview

Diagram 3.2 shows a complete overview of how the different key technologies interact with each other, as well as containers and key files in the process.

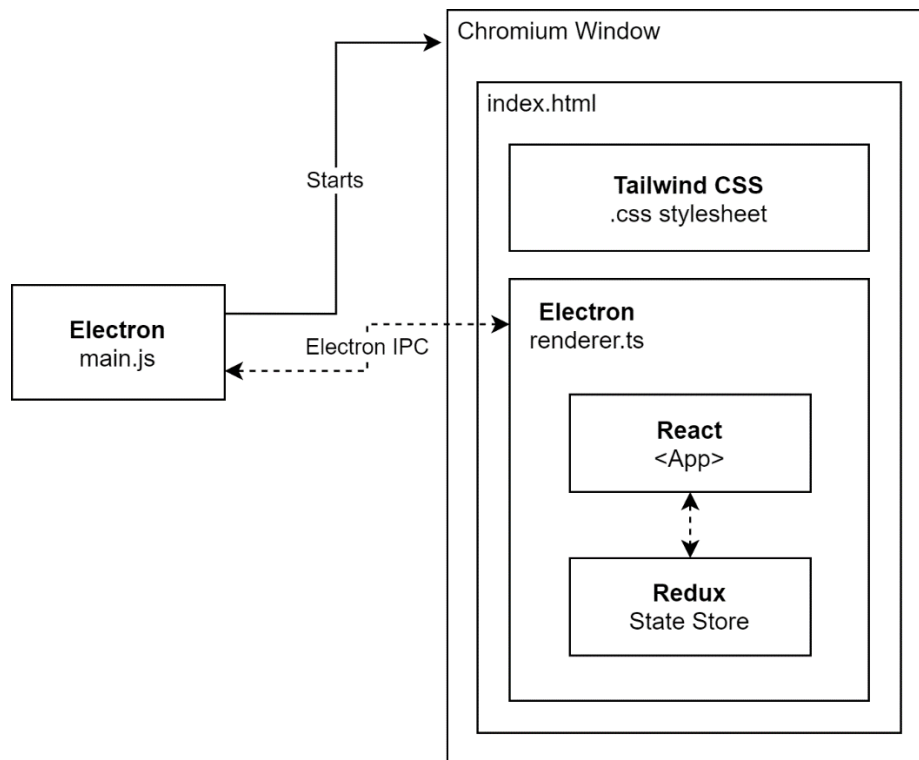


Diagram 3.2 Technology Overview

The main Electron process begins with `main.ts`, the starting point of the application. The main process then creates a browser window, which renders `index.html`. The contents of this HTML file are crucial, as they include the Tailwind CSS stylesheet, the `renderer.ts` script and “root” element for attaching React to the DOM. From there, the `renderer.ts` script calls the React code, which handles all further logic. React also activates Redux, as well as other handler functions which contain the heavier app logic.

## **3.4 Remote Host Technologies**

### **3.4.1 Node.js**

Node.js is also used as the server runtime. More details about Node.js are explained in 3.1.

### **3.4.2 Express.js**

Express.js is one of the most popular robust web application frameworks. Express allows us to easily set up a web server featuring all the required API endpoints as well as authentication.

### **3.4.3 JWT (JSON Web Token)**

The server implements a very simple form of authentication. The mechanism uses a password defined in the server config. Users who know this password can use it to generate a JSON Web Token granting them access to the project and keeping track of their identity.

The json-web-token npm package is used to provide JWT encoding and decoding functionality.



## 4 Solution Overview

### 4.1 Visual Identity


At the start of the project, an exploration was conducted into certain elements which would guide the later UI development and product design. The basic visual identity parameters which were explored were the name, logo, and color scheme of the product.

#### 4.1.1 Name

The product was named “Documint” as a play on words between document and mint. The name contains a strong descriptive element, while still being a discrete, personalized term. The “mint” element guided the logo and color scheme development.

#### 4.1.2 Color scheme

Through the process, multiple color schemes have been explored. The explored palettes are shown on Picture 4.1, Picture 4.2, and Picture 4.3.




Color 1		Color 2		Color 3		Color 4		Color 5	
HEX	CAD2C5	HEX	84A98C	HEX	52796F	HEX	354F52	HEX	2F3E46
RGB	202, 210, 197	RGB	132, 169, 140	RGB	82, 121, 111	RGB	53, 79, 82	RGB	47, 62, 70
HSB	97, 6, 82	HSB	133, 22, 66	HSB	165, 32, 47	HSB	186, 35, 32	HSB	201, 33, 27
CMYK	3, 0, 6, 17	CMYK	21, 0, 17, 33	CMYK	32, 0, 8, 52	CMYK	35, 3, 0, 67	CMYK	32, 11, 0, 72






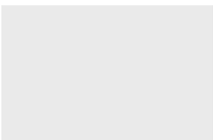


Picture 4.1 Palette Option 1 [<sup>14</sup>]

---

<sup>14</sup> Generated on <https://coolors.co/generate>

									
Color 1		Color 2		Color 3		Color 4		Color 5	
HEX	1F2421	HEX	216869	HEX	49A078	HEX	90C5A1	HEX	DCE1DE
RGB	31, 36, 33	RGB	33, 104, 105	RGB	73, 160, 120	RGB	156, 197, 161	RGB	220, 225, 222
HSB	144, 14, 14	HSB	181, 69, 41	HSB	152, 54, 63	HSB	127, 21, 77	HSB	144, 2, 88
CMYK	13, 0, 8, 85	CMYK	68, 0, 0, 58	CMYK	54, 0, 25, 37	CMYK	20, 0, 18, 22	CMYK	2, 0, 1, 11

Picture 4.2 Palette Option 2 [<sup>15</sup>]

 <p><b>Jungle Green</b> Light Shade 01</p> <p>#42C4A4 rgb(66, 196, 164) Pantone 7465 C</p>	 <p><b>Jungle Green</b></p> <p>#1FA37F rgb(31, 163, 127) Pantone 7473 C</p>	 <p><b>Jungle Green</b> Dark Shade 01</p> <p>#078B64 rgb(7, 139, 100) Pantone 3288 C</p>	 <p><b>Jungle Green</b> Dark Shade 02</p> <p>#057351 rgb(5, 115, 81) Pantone 341 C</p>
 <p><b>White</b></p> <p>#FFFFFF rgb(255, 255, 255) Pantone 7436 C</p>	 <p><b>White</b> Dark Shade 01</p> <p>#EAEAEA rgb(234, 234, 234) Pantone 663 C</p>	 <p><b>White</b> Dark Shade 02</p> <p>#D5D5D5 rgb(213, 213, 213) Pantone 427 C</p>	 <p><b>White</b> Dark Shade 03</p> <p>#BABABA rgb(186, 186, 186) Pantone Cool Gray 4 C</p>

Picture 4.3 Palette Option 3 [<sup>16</sup>]

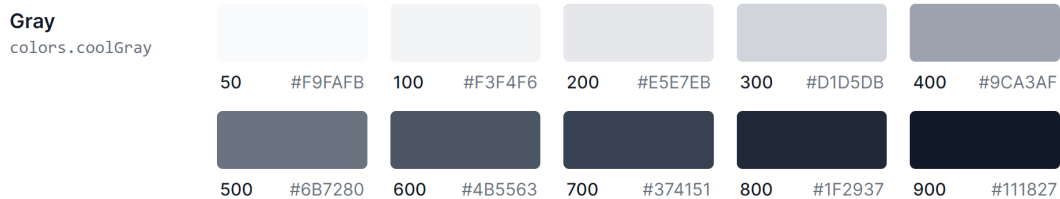
<sup>15</sup> Generated on <https://coolors.co/generate/>

<sup>16</sup> Generated on <https://pigment.shapefactory.co/>

In the end, the final color scheme comprised of just a pair of colors inspired by mint leaves, the primary accent color #1FA37F, and a darker variant #057351, both of which shown on Picture 4.2 Main Colors. The rest of the color scheme relied on shades of grey for all other needs. The specific shades used are supplied by the Tailwind CSS framework and can be seen on Picture 4.3 Tailwind Grey.



Picture 4.4 Main Colors



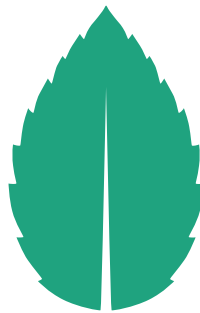
Picture 4.5 Tailwind Grey <sup>[17]</sup>

---

<sup>17</sup> Tailwind CSS, *Customizing Colors*, <https://tailwindcss.com/docs/customizing-colors>, 10.06.2021

### 4.1.3 Logo and branding

A custom logo was developed in vector format representing a mint leaf (Picture 4.4). This logo serves as the main application icon and as the base for all GitHub repository banners (Picture 4.5 and Picture 4.6) which were created for this project. The complete set of branding resources can be seen below:



*Picture 4.6 Documint Logo*



*Picture 4.7 Small Git Banner*



*Picture 4.8 Large Git Banner*

## 4.2 Client Architecture

Because the application was developed with Electron in mind from the start, it does not follow the traditional Web-app architecture. Electron apps do not have the same level of maturity and best practices and architecture examples are often outdated. Hence, a custom architecture was implemented with the goal of creating a clean codebase while maintaining the advantages Electron has to offer.

One of the main differences stems from the fact that React can be connected directly to the front-end. This means that React is not rendered and then sent to a browser, instead it is compiled and executed directly inside of the browser itself. Because of this, effectively all logic is handled within the same process, and a custom defined barrier had to be defined to separate the UI code from the backend and application logic code.

In the codebase, this is handled by separating all non-UI logic into separate 'utility' scripts. These utility scripts primarily include asynchronous methods which can be called from the UI code to execute more complex logic and actions involving other libraries. For example, handlers currently handle logic such as filesystem access, markdown compilation, server communication, Electron IPC communication, search, etc.

This leaves only UI logic in TSX (TypeScript React Syntax). This contains the handling of user actions and the application state. The global application state is controlled via the Redux framework, which offers another separate logic category. (Component-specific state is still implemented through React's useState hook)

Redux actions form their own logical group, as they are tightly connected to the Redux structure, depending on a global action dispatcher and asynchronous handling by the Redux framework. The Redux state is also accessed through a system of selectors, meaning that UI elements are completely separated from directly managing the state.

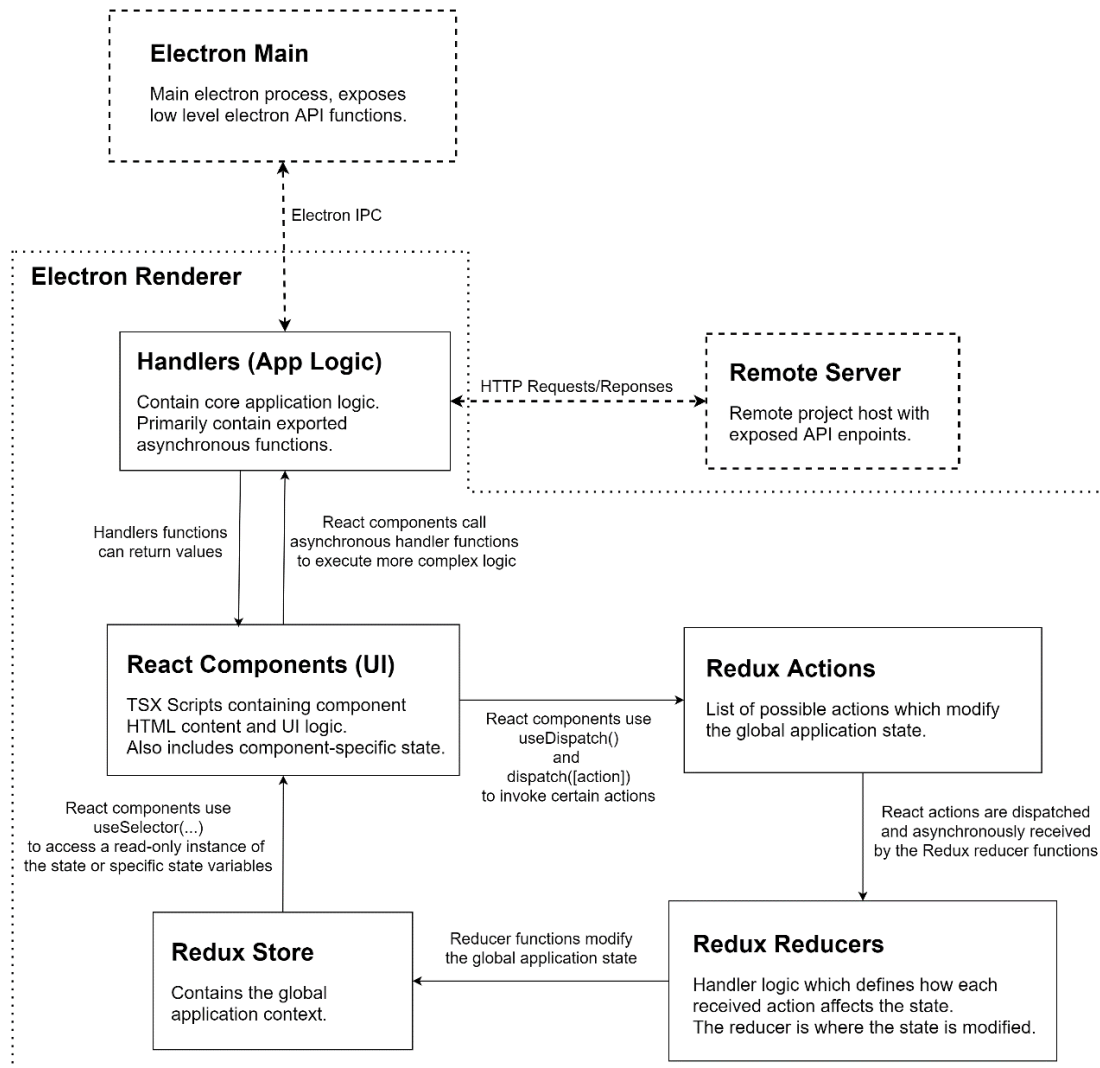


Diagram 4.1 Client Architecture

Diagram 4.1 shows how these groups interact with each other. This architecture limits access to specific resources to specific functions and channels, increasing code flow readability as well as helping to reduce possible points of failure when debugging a specific issue.

## 4.3 Core Client Functionality

The client implements all its functionality without relying on any remote sources. This means that the client is completely functional in an offline environment, and its features will first be explained in the local context, not relying on the remote host.

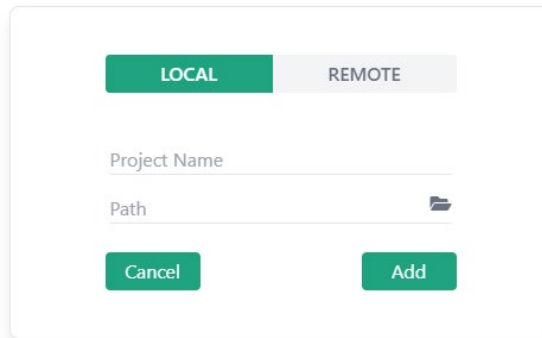
### 4.3.1 Project Management Screen

The first screen of the program is the project manager screen, seen in Picture 4.7. This screen allows the user to see a list of accessible projects. The user can open a project, delete a project, or add a new project to the list using through the UI.



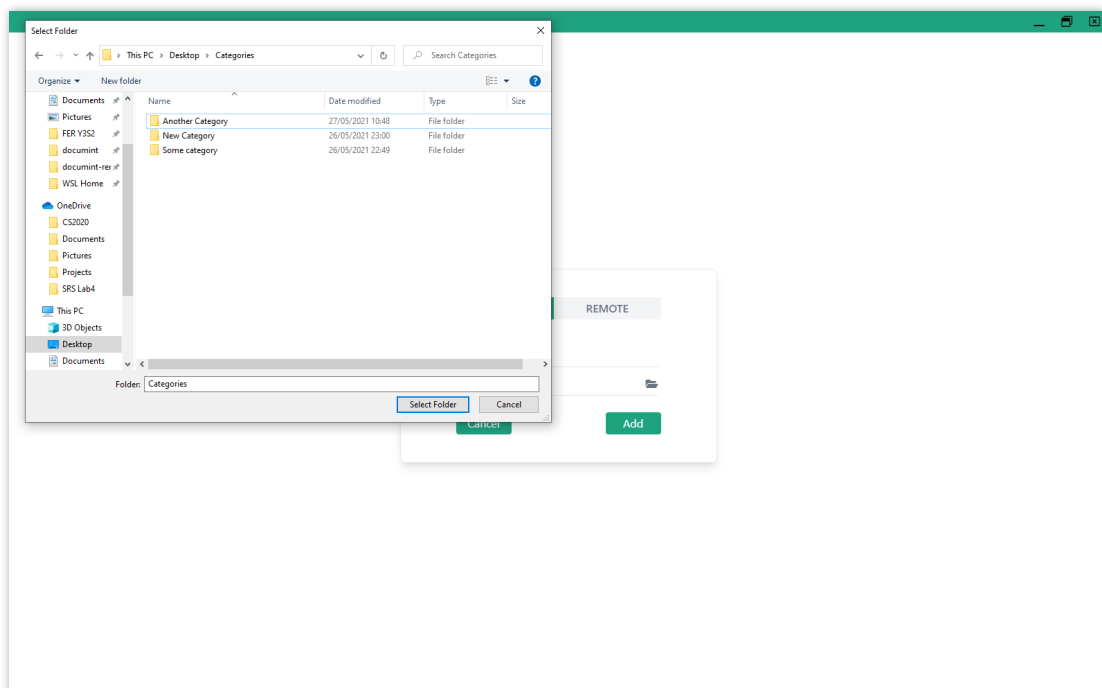
Picture 4.9 Documint Project Manager

The “add project” prompt, shown in Picture 4.8 allows the user to define whether they are adding a local or remote project, showing appropriate fields for each type.



Picture 4.10 Documint Add Project

The directory icon allows the user to use whatever native file browser is offered by his operating system. For example, on Windows, it allows the user to select a folder using File Explorer as seen in Picture 4.9. This functionality comes from Electron.js’ ability to interact with OS-native functions.

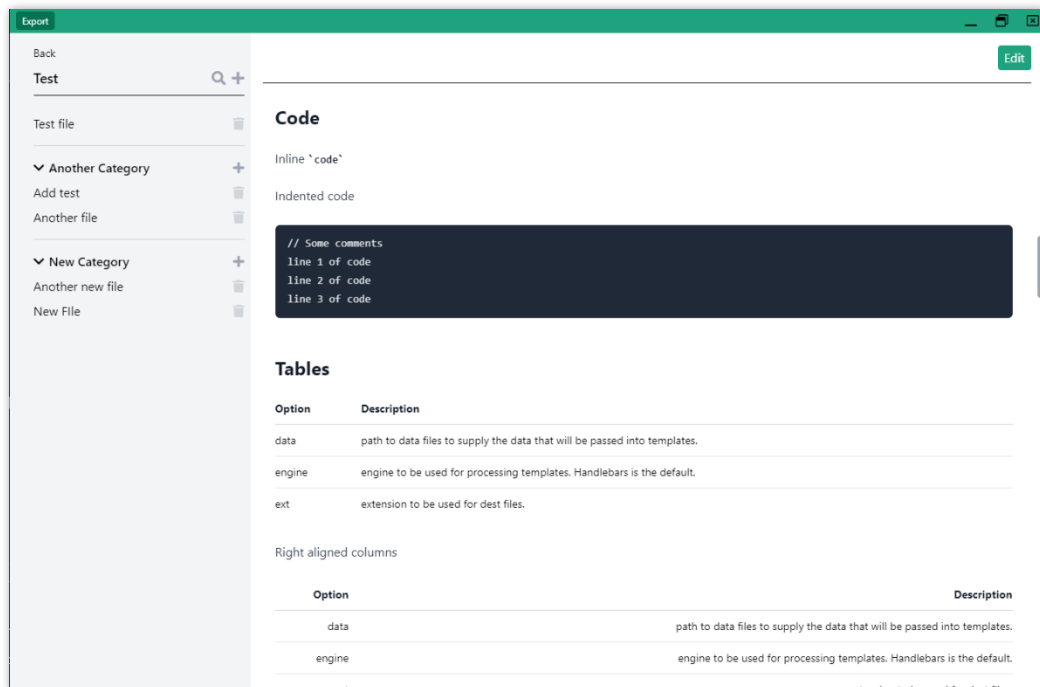


Picture 4.11 Documint Folder Selection



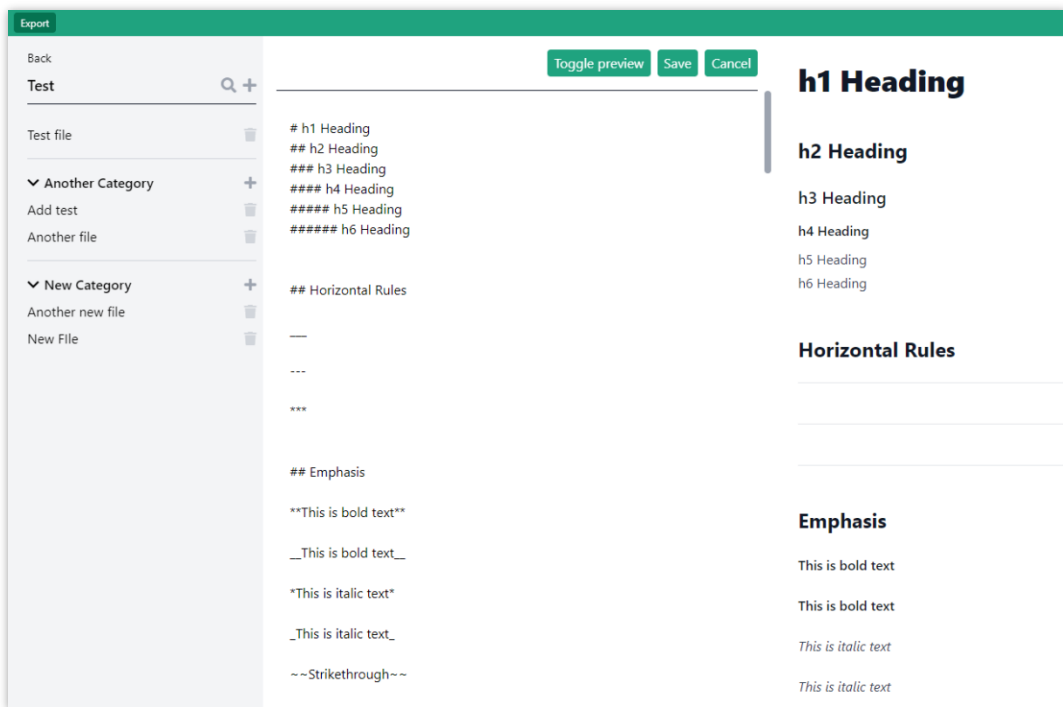
Once a project has been added, it can be opened, loading the files, and displaying the main editor shown in Picture 4.10. The files are loaded by processing all .md files within the root folder and any .md files in first level subfolders, where the subfolder is considered as the category. The loading system work for any user-specified directory, and it relies on Node.js' asynchronous fs.promises package for filesystem access.

### 4.3.2 Main Editor



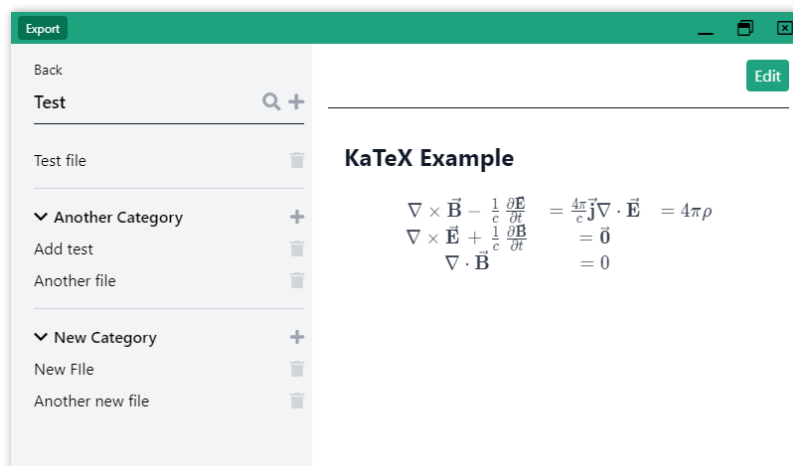
Picture 4.12 Documint Main Editor

Once loaded, the main editor allows the user to browse through the file using the navigation on the left, as well as using the content-aware search bar through the search icon. Content is shown as rendered HTML, while the source files are written in markdown syntax following the CommonMark specification. The edit button allows the user to enter edit mode, which lets them edit the markdown source content. While in edit mode, they can enable a toggleable live preview, as seen in Picture 4.11. The edit mode allows the user to preview rendered HTML while writing the markdown.



Picture 4.13 Documint Live Preview Edit

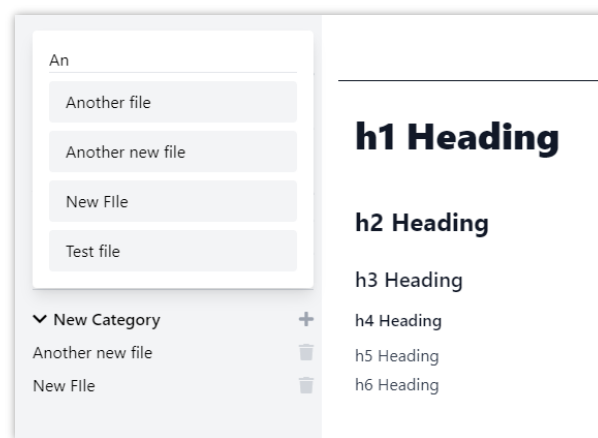
As mentioned, the markdown source is parsed according to most of the CommonMark specification, with some additional syntax options from pandoc such as KaTeX math which can be seen in Picture 4.14.



Picture 4.14 KaTeX Example

### 4.3.3 Search

The search functionality relies on a custom Implementation of a very primitive search algorithm. Because the content-aware search needs to process all the documents and their contents, they are all loaded into memory and then searched using a regular expression match. The results are sorted primarily by the number of matches in the document title, and secondarily by the number of matches in the contents. An example of the search interface can be seen in Picture 4.15.



Picture 4.15 Search Example

The search is performed on every character entry. This search method is very unoptimized and uses up a lot of memory to cache the whole project.

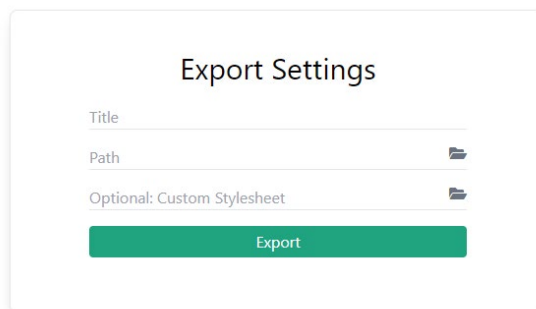
To slightly reduce frequency of file access operations, as well as the duration for which the memory is being used to store the content of all files, this file caching mechanism works like a debounced function. This way, the contents are dropped from memory 90 seconds after the last search. The contents are cached again on the next user search or keypress.

The search is one of the weaker points of the finished product and offers room for further improvements. Despite this, the per-character search works surprisingly well, and it produces the expected results, making it normally usable on medium or better performing devices.

### 4.3.4 Static Export

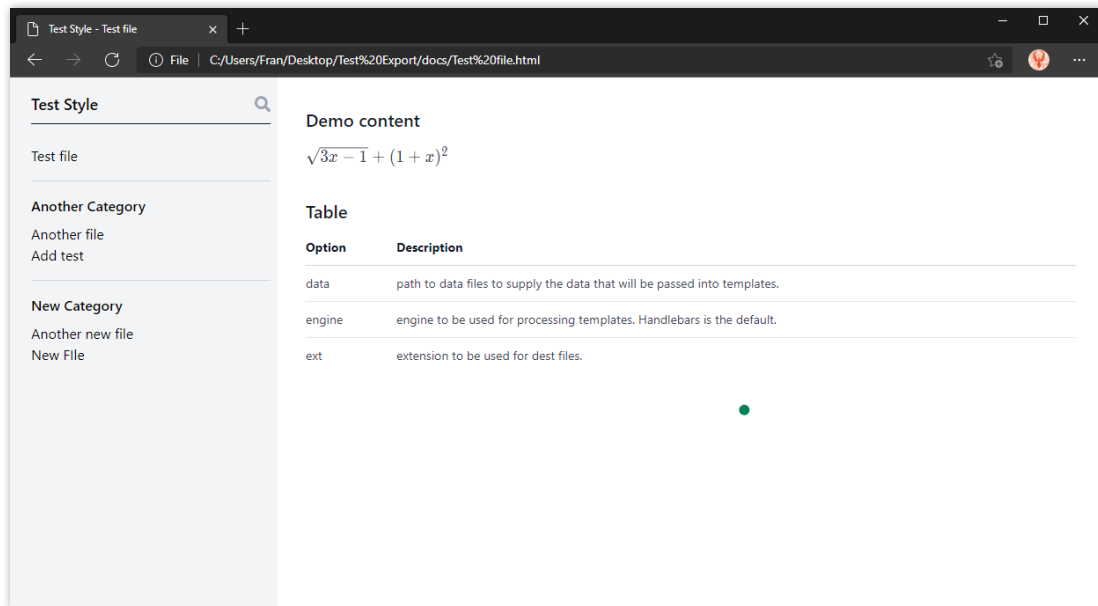
Once the user is satisfied with the documentation, an export can be made to generate a static website containing the documents. A static website has been chosen as the output format as it does not require setting up a web server and can be hosted for free via services such as GitHub Pages, allowing developers and teams to easily add rich documentation to their GitHub Projects.

The export can be started by pressing the export button in the top left corner, which brings up the prompt shown in Picture 4.12, allowing the user to configure the output directory, title of the exported documentation, and optionally include a custom stylesheet.

A dialog box titled "Export Settings" with a light gray background and rounded corners. It contains three input fields: "Title" (a simple text box), "Path" (a text box with a folder icon on the right), and "Optional: Custom Stylesheet" (a text box with a folder icon on the right). Below these fields is a green button with the text "Export" in white.

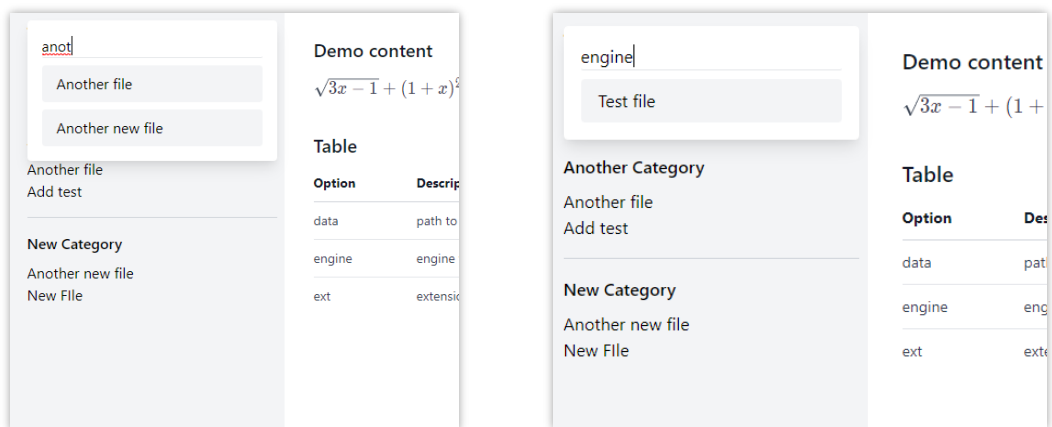
*Picture 4.16 Documint Export Prompt*

An example of the exported website can be seen on Picture 4.13.



Picture 4.17 Static Export

The exported website also features the navigation sidebar as well as a complete content-aware search as seen in Picture 4.14.



Picture 4.18 Content-aware Search

## 4.4 Server Architecture and Functionality

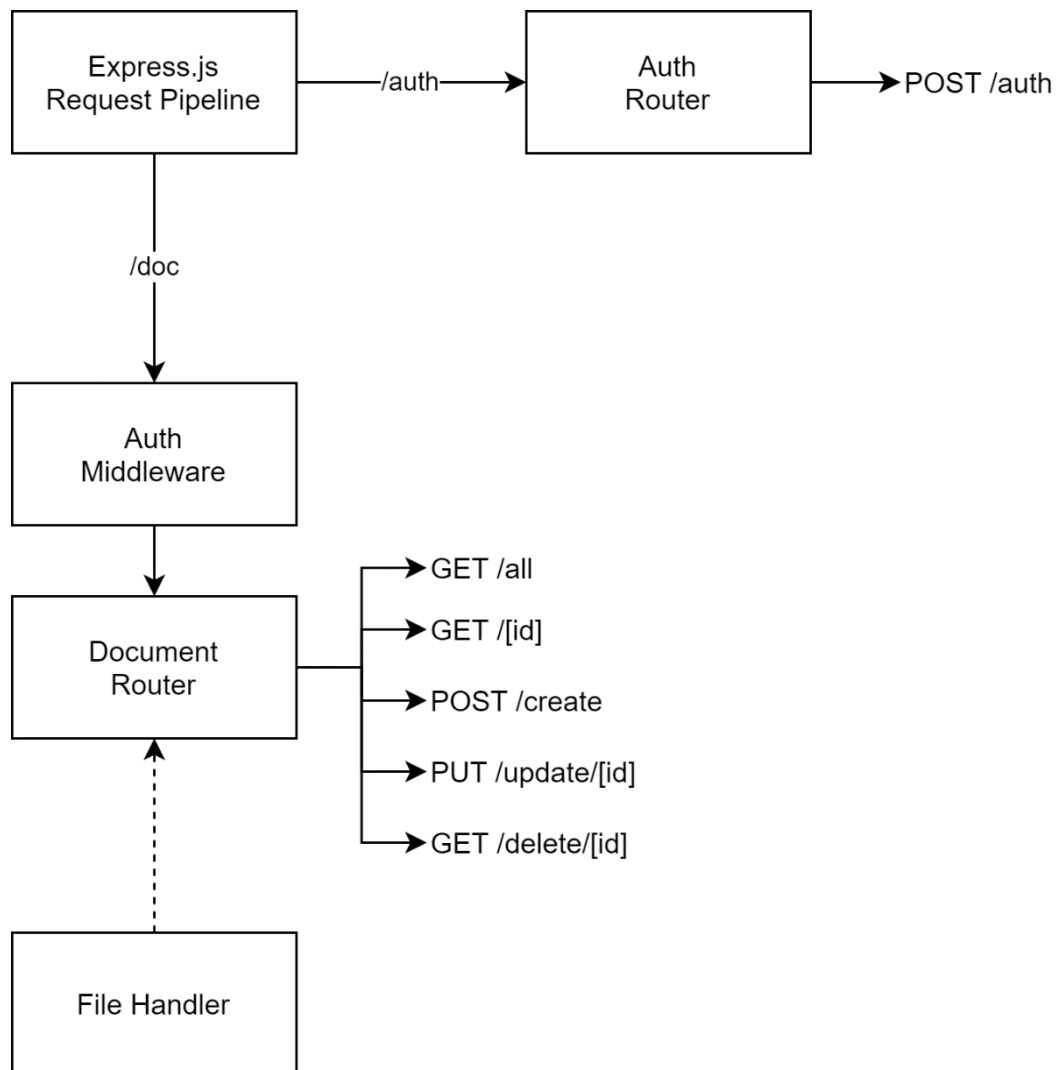


Diagram 4.2 Server Architecture

The server is essentially a barebones implementation of the necessary functions to handle collaborative work on a documentation project.

To unauthorized users, the only available endpoint is the `/auth/token` endpoint which takes in the configured password to provide users with an authorizing JSON web token. The users then use the token to gain access to `/doc/...` endpoints, which are used to modify the project.

A complete list of server endpoints and request/response forms can be seen in Table 4.1.

Route	Request Body	Response Body
POST /auth/token	{ user: string, password: string }	{ auth: bool, token: string }
GET /doc/all *	-	{ id: string, name: string, category: string }[]
GET /doc/[id] *	-	{ name: string, category: string, content: string, lastUpdated: number }
POST /doc/create *	{ name: string, category: string }	{ success: bool }
PUT /doc/update/[id] *	{ content: string, readTimestamp: number }	{ success: bool }
GET /doc/delete/[id] *	-	{ success: bool }

\* These routes require the JWT token in the “x-access-token” header.

*Table 4.1 Remote Endpoints*

In addition to the “id” parameter from the route and the body of each request, the server also has info about the “user” name, as it is contained within the JWT token and decoded in the auth middleware.

The update endpoint implements a simple conflict detection system to deal with asynchronous concurrent editing of documents. In the scenario that two people are editing the same document at the same time, the latter one to publish his changes will create a separate document instead of overwriting the unseen changes made by the other person. The separate document is titled the same as the original plus the name of the user who submitted the update. This leaves the final conflict resolution to the users.

An example of conflict detection can be seen in the following series of events:

1. First user, "Ron", begins editing the "Introduction" document.
2. Second user, "Tammy", begins editing the same document.
3. Tammy writes "Welcome!" in the document and saves it.
4. The server receives and updates the Introduction document.
5. Ron writes "Goodbye!" in the document and saves it. (his local copy was made before Tammy's changes)
6. The server detects that Ron is trying to update a document using an outdated reference, and instead, creates a separate new document titled "Introduction - Ron".
7. This way, both of their additions were stored on the server, and no work was overwritten by subsequent changes.
8. It remains for the users to agree on the final version of the document, updating it and deleting the extra which was created in the conflict.

This method is robust, and any number of subsequent document updates based on outdated versions of the document will lead to new documents being created. The method prioritizes saving all of the work.



## 5 Reflection and further development

The development process has been a very valuable experience, and an encounter with a whole variety of new frameworks and technologies. It was a valuable learning experience, and a way to put pre-existing skills to the test. During the process I attempted to emulate a real-life development process, keeping in mind non-coding elements of development such as issue tracking on GitHub, constructing a visual identity for the product, and treating the project as if it were being built for real world use.

There were certainly hick ups in the process, for example in user interface and styling, which is among my weaker skills. At one point of the project, I was effectively forced to scrap SASS from the technology stack and replace it with Tailwind to speed up the UI development. This required me to completely remake the existing UI elements, however it saved a lot of time in the long run, allowing for much faster development after the switch was made.

Apart from this, the technology stack held up very well. While it was constructed in a way which would reduce the need for extra packagers such as webpack or compilers such as babel, the chosen frameworks were able to perform together very well. The final Electron distribution was 68.8MB for the .msi installer, 53.8MB for the portable .exe executable, and 196MB unpacked. This is a large file size for such a small program, however most of the file size comes from the fact that Electron distributions include a packaged version of the Chromium browser, whose full size is 171 MB [18]. As such, Electron introduces a lot of overhead in addition to the application contents, however for larger projects the additional ~170MB of size could be negligible, especially when considering the benefits of portability and cross-platform support that Electron offers.

---

<sup>18</sup> The Chromium Projects, *Download Chromium*,  
<https://www.chromium.org/getting-involved/download-chromium>, 10.06.2021

TypeScript Compiler's native support for React TSX scripts allowed for easy React integration into the project, and with a little bit of tinkering, it was easy to set up a complete system incorporating React into Electron's browser window. Additional libraries, such as the markdown compiler as well as Pug.js for export templating have also performed as expected, resulting in a very functional package. Using Redux as a package manager has proved to be a very useful addition to the stack as well, allowing for a very structured state management system, at the acceptable cost of a little extra boilerplate code.

## Conclusion

The goal of this project was to develop a complete tool which would assist in the creation of project documentations from start to finish. This means that the user does not have to rely on any external tools in the process, and that the result is a functional and easily shareable documentation. The final product includes the necessary functionality to satisfy these criteria. The program allows for writing documents from scratch and organizing them into logical categories. It allows users to include rich content such as formatted text, tables, and even mathematical expressions in their documents, which are rendered by a powerful and fast markdown compiler.

Further development is needed to push the product towards practical use. The interface requires a lot of updates, and features such as the search in exported websites have a lot of room for optimization. But overall, I am very satisfied with the result given the time constraints and the wide scope of the development process.

# References

1. GitBook Documentation, *What is GitBook*,  
<https://docs.gitbook.com/>, 10.06.2021
2. GitBook Documentation, *Markdown*,  
<https://docs.gitbook.com/editing-content/markdown>, 10.06.2021
3. Docusaurus, *Introduction*,  
<https://docusaurus.io/docs/>, 10.06.2021
4. Tailwind CSS, *Customizing Colors*,  
<https://tailwindcss.com/docs/customizing-colors>, 10.06.2021
5. Redis Developer Hub, *The Home of Redis Developers*,  
<https://developer.redislabs.com/>, 10.06.2021
6. Node.js, *About*,  
<https://nodejs.org/en/about/>, 10.06.2021
7. Electron, *Build cross-platform desktop apps with JavaScript, HTML, and CSS*,  
<https://www.electronjs.org/>, 10.06.2021
8. TypeScript, *Typed JavaScript at Any Scale*,  
<https://www.typescriptlang.org/>, 10.06.2021
9. Stack Overflow, *Stack Overflow Developer Survey 2020*,  
<https://insights.stackoverflow.com/survey/2020>, 10.06.2021
10. Redux, *Redux*,  
<https://redux.js.org/>, 10.06.2021
11. GitHub, *markdown-it*,  
<https://github.com/markdown-it/markdown-it>, 10.06.2021
12. Tailwind CSS, *Rapidly build modern websites without ever leaving your HTML*,  
<https://tailwindcss.com/>, 10.06.2021
13. Pug, *Getting Started*,  
<https://pugjs.org/api/getting-started.html>, 10.06.2021
14. Tailwind CSS, *Customizing Colors*,  
<https://tailwindcss.com/docs/customizing-colors>, 10.06.2021
15. The Chromium Projects, *Download Chromium*,  
<https://www.chromium.org/getting-involved/download-chromium>, 10.06.2021

# **System for Writing and Organizing Program Documentation**

## **Summary**

This project follows the development of a cross-platform documentation writing a publishing tool developed on top of the Electron.js framework. It also focuses on the development process as well as the different technologies which were used in the process. The final product includes the main application and a lightweight remote server for hosting collaborative projects.

## **Keywords**

Documentation, Productivity, Markdown-it, Electron.js, Node.js, React.js, Redux.js, Tailwind CSS, Pug.js

# **Sustav za pisanje i organizaciju programske dokumentacije**

## **Sažetak**

Ovaj projekt prati razvoj više platformskog alata za pisanje i objavu dokumentacije, razvijenog pomoću Electron.js-a. Također se fokusira na sam proces razvoja, te razne uključene tehnologije. Konačno rješenje uključuje glavnu aplikaciju i lagani web server koji služi za posluživanje kolaborativnih projekata odnosno dokumentacije.

## **Ključne riječi**

Dokumentacija, Produktivnost, Markdown-it, Electron.js, Node.js, React.js, Redux.js, Tailwind CSS, Pug.js