UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

A web framework for collaborative code development

Diploma Thesis

Andromachi Kosmatou

Supervisor: Georgios Thanos

March 2024

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

A web framework for collaborative code development

Diploma Thesis

Andromachi Kosmatou

Supervisor: Georgios Thanos

March 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Δημιουργία περιβάλλοντος για συνεργατική ανάπτυξη κώδικα στο διαδίκτυο

Διπλωματική Εργασία

Ανδρομάχη Κοσμάτου

Επιβλέπων/πουσα: Γεώργιος Θάνος

Μάρτιος 2024

Approved by the Examination Committee:

Supervisor **Georgios Thanos**
Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

Member **Eleni Tousidou**
Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

Member **Athanasios Fevgas**
Laboratory Teaching Staff member, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Georgios Thanos, for his support, guidance, and invaluable feedback throughout the entirety of this research journey. I am also thankful to Eleni Tousidou and Athanasios Fevgas for their constructive criticism and for being part of this thesis committee.

I am indebted to my family for their encouragement, and belief in my abilities. This thesis is dedicated to all those who have supported me along the way. Thank you.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Andromachi Kosmatou

Diploma Thesis

A web framework for collaborative code development

Andromachi Kosmatou

Abstract

In the ever-changing environment of software development, efficient and collaborative code development has become paramount. Adding to the reasons that make the need for optimal solutions in this area imperative is the coronavirus pandemic. With the majority of companies and universities operating online during the quarantine period and even now, there has been an incentive to design new tools for collaboration between developers. This is the topic addressed in this thesis, in which a web application was designed and implemented to optimize collaboration and code sharing.

Specifically, in this research we tried to implement an application, the idea of which was generated by the lack of similar applications on the internet. Although there is a plethora of tools for code development, it was difficult to find one that combined a range of features that provided all the services a developer needs to complete his work seamlessly. We implemented an application in which any user can create a project and work on it by saving all the project files, sharing their work with other users of the application and much more.

We believe this research will help developers and students by improving their experience in collaborative code development. At the same time, it can become a useful tool in the hands of a person with no experience in programming and without the proper setup, to write code and take feedback for this code just with an internet connection. Hopefully, this paper will lay the groundwork for future research, which will take our work one step further.

Keywords:

collaborative code development, web application, Golang, React, Google Cloud, GitHub integration, web-sockets, PostgreSQL

Διπλωματική Εργασία

Δημιουργία περιβάλλοντος για συνεργατική ανάπτυξη κώδικα στο διαδίκτυο

Ανδρομάχη Κοσμάτου

Περίληψη

Στο συνεχώς μεταβαλλόμενο περιβάλλον της ανάπτυξης λογισμικού, η αποδοτική και συνεργατική ανάπτυξη κώδικα έχει καταστεί υψίστης σημασίας. Στους λόγους που καθιστούν επιτακτική την ανάγκη για βέλτιστες λύσεις στον τομέα αυτό, έρχεται να προστεθεί η πανδημία του κορωνοϊού. Με την πλειονότητα των εταιρειών και πανεπιστημίων να λειτουργούν διαδικτυακά την περίοδο της καραντίνας αλλά ακόμα και τώρα, δόθηκε κίνητρο να σχεδιαστούν καινούρια εργαλεία για την συνεργασία μεταξύ προγραμματιστών. Με αυτό το θέμα καταπιάνεται και η παρούσα διπλωματική, στην οποία σχεδιάστηκε και υλοποιήθηκε μια διαδικτυακή εφαρμογή με σκοπό την βελτιστοποίηση της συνεργασίας και της κοινής χρήσης κώδικα.

Συγκεκριμένα, στην παρούσα έρευνα προσπαθήσαμε να υλοποιήσουμε μια εφαρμογή, η ιδέα της οποίας μας δημιουργήθηκε από την έλλειψη παρόμοιων εφαρμογών στο διαδίκτυο. Αν και υπάρχει πληθώρα εργαλείων πάνω στην ανάπτυξη κώδικα, στάθηκε δύσκολο να βρούμε κάποιο που να συνδυάζει ένα εύρος χαρακτηριστικών, το οποίο να παρέχει όλες τις υπηρεσίες που ένας προγραμματιστής χρειάζεται για να ολοκληρώσει απρόσκοπτα το έργο του. Εμείς, υλοποιήσαμε μια εφαρμογή στην οποία κάθε χρήστης μπορεί να δημιουργήσει ένα project και να δουλέψει πάνω σε αυτό αποθηκεύοντας όλα τα αρχεία του project, μοιράζοντας την δουλειά του με άλλους χρήστες της εφαρμογής και πολλά ακόμα.

Πιστεύουμε ότι αυτή η έρευνα θα βοηθήσει τους προγραμματιστές και τους φοιτητές, βελτιώνοντας την εμπειρία τους στην συνεργατική ανάπτυξη κώδικα. Παράλληλα, μπορεί να γίνει ένα χρήσιμο εργαλείο στα χέρια ενός ανθρώπου χωρίς εμπειρία στον προγραμματισμό και χωρίς το κατάλληλο setup, να γράψει κώδικα και να λάβει αξιολόγηση από άλλους χρήστες για τον κώδικα αυτό, απαιτώντας από αυτόν μόνο σύνδεση στο διαδίκτυο. Ελπίζουμε, η παρούσα εργασία να θέσει το υπόβαθρο σε μελλοντικές έρευνες, που θα πάνε ένα

βήμα παραπέρα την δική μας δουλειά.

Λέξεις-κλειδιά:

συνεργατική ανάπτυξη κώδικα, διαδικτυακή εφαρμογή, Golang, React, Google Cloud, ενσωμάτωση με GitHub, web-sockets, PostgreSQL

Table of contents

Acknowledgements **ix**

Abstract **xii**

Περίληψη **xiii**

Table of contents **xv**

List of figures **xix**

Abbreviations **xxi**

1 Introduction **1**

1.1 Main objective 2

1.2 Thesis Structure 2

1.3 Related Work 3

1.3.1 Replit 3

1.3.2 Online IDE 3

2 Analysis **5**

2.1 Authentication 5

2.1.1 Log in 5

2.1.2 Registration 5

2.1.3 Log out 6

2.2 Project functionalities 6

2.3 File functionalities 6

2.4 GitHub Functionalities 7

2.5 Chat 7

3 Used Technologies **9**

3.1 Database 9

3.2 Back-end technologies 10

3.2.1 GORM 10

3.2.2 Gorilla web toolkit 10

3.2.3 JWT 10

3.2.4 SendGrid 11

3.3 Front-end technologies 11

3.3.1 React 11

3.3.2 JSX 12

3.3.3 DOM 12

3.3.4 React DOM and react-dom 12

3.3.5 react-router-dom 12

3.3.6 Asynchronous functions 13

3.3.7 Promises 13

3.3.8 Axios 15

3.3.9 Material UI 15

3.3.10 TanStack Query 15

3.3.11 monaco-editor 16

3.3.12 web-sockets 16

3.3.13 react-use-websocket 17

4 Technical Description **19**

4.1 REST API 19

4.2 Front-end 20

4.2.1 UI 20

4.2.2 Routing 22

4.2.3 State management 23

4.2.4 GitHub API 23

4.3 Back-end 24

4.3.1 Routing 24

4.3.2 Model-View-Controller 25
4.3.3 Models 25
4.3.4 Controllers 27
4.3.5 Chat 27

5 Deployment **29**
5.1 Database 29
5.2 Server 30
5.3 Storage 30
5.4 Front-end 31

6 Conclusions **33**
6.1 Summary and conclusions 33
6.2 Future Improvements 33

Bibliography **37**

List of figures

3.1 Callback function example 13
3.2 The different states of a promise 14
3.3 Promise chaining 14
3.4 Async/await syntax 15
3.5 Web-socket protocol 17

4.1 Login page layout 21
4.2 View of the project list 21
4.3 View of a Menu 22
4.4 MVC design pattern 25

6.1 Overview of how a pseudo-terminal operates 34

Abbreviations

API	Application Programming Interface
CLI	Command Line Interface
CSP	Communicating Sequential Processes
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
JSX	JavaScript XML
JWT	JSON Web Token
MVC	Model-View-Controller
ORM	Object-Relational Mapping
REST	REpresentational State Transfer
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Chapter 1

Introduction

In the current era of increasing remote work among programmers, the demand of new tools has never been more pressing. Our research into existing solutions reveals a gap in the market: no single application encompasses all the features essential to developers. While some apps enable code editing and execution, they lack file storage capabilities. Conversely, others function as comprehensive online code editors but they don't offer collaborative tools crucial for team projects. By asking a number of developers and from our own experience, we have identified key features a good online code editor should incorporate.

Initially, it should support multiple languages for running, building and compiling, complemented by a user-friendly terminal interface. Moreover, it should provide secure file storage for project components, seamless sharing capabilities within the application, integration with GitHub for remote repository management, and real-time chat functionality to facilitate team communication.

In subsequent sections, we will conduct a comparative analysis of similar web applications addressing this industry challenge. Furthermore, we will provide a comprehensive overview of our application's functionality and the methodologies employed in its development. We will also explain technical features and problems we faced and we will show the user interface of the online code editor. Finally, we will propose potential enhancements to optimize the application for widespread adoption within the industry.

1.1 Main objective

The concept of this web application is to provide a fully-featured environment for collaborative code development. It aims to offer a valuable tool to programmers or students that can write code and share their projects with fellow programmers with a minimal setup. Specifically, the application will do the following :

- Allow users to view and edit code in a multi-language, syntax-highlighting code editor
- Allow users to share their projects with others. Anyone with access to a project has permission to modify it
- Project collaborators can communicate in real time with our chat functionality, speeding up the development time
- Any user can search for public projects, view their contents, and even clone them as their private copy
- With GitHub integration, a GitHub-logged-in user can initialize a remote repository, commit changes to a remote repository, and can also clone a remote repository as a local copy

1.2 Thesis Structure

Chapter 1 provides an introduction to the topics we will discuss in the rest of this thesis, as well as a first insight into what the application is all about and what problems it aims to solve. Chapter 2 explains the application's functionality, guiding the user through the many things one can do with our code editor. Then, in Chapter 3, we refer to the technologies we used to implement the application and key concepts that will help the reader understand the development process. Chapter 4 delves deep into the technical features of the application, talking about the design and architecture. Chapter 5 discusses the deployment and the challenges we faced in deploying the various parts of our application. Finally, in Chapter 6, we summarize our findings and try to take a critical look at what we could have done better and what improvements we want to make in the future.

1.3 Related Work

1.3.1 Replit

Replit is an online IDE utilized with several programming languages. Replit facilitates collaborative coding by enabling multiple users to edit a shared repl, providing real-time edits across files, and offering instant messaging. Through a shared compute engine, code can be executed and displayed uniformly to all users within a Repl. The platform incorporates built-in source control using Git across all Repls, allowing users to switch branches, push files, and revert code as needed. Additionally, it supports the integration of code from GitHub repositories and the linkage of Repls to GitHub repositories. Repls written in Java, Python, Node.js, and C++ feature debugging and unit testing support. Furthermore, Repl offers secrets management, allowing users to conceal sensitive values from public view. [1]

1.3.2 Online IDE

The Online IDE, powered by the ACE code editor, presents an extensive platform tailored for programming practice. It facilitates the entire programming life-cycle, from learning and building to running and testing programs. Users have the flexibility to import code from their local environment and continue development within the IDE. Additionally, the option to download both code and output is available, enhancing accessibility and sharing capabilities. While collaborative features and Git integration are absent, the IDE nonetheless provides a straightforward and efficient interface for coding and execution purposes. [2]

Chapter 2

Analysis

Our application aims to facilitate the software development process during a pivotal era of technological advancement. By integrating essential functionalities, such as a file storage and a real-time chat, users can seamlessly code, without using a bunch of different applications. With our online code editor serving as a centralized hub, developers have access to all necessary tools and resources conveniently consolidated within a single platform.

2.1 Authentication

2.1.1 Log in

Upon entering our application, the login page prompts the users to enter their credentials, comprising their email address and password. Having completed the registration process, wherein their email address is verified, and if their credentials match the ones in our database records, the users gain access to the application. If the users have forgotten their passwords, they can recover access to their accounts by setting a new password. The application sends an email to the user's mail with a code, which then the user can fill in the appropriate form and set a new password.

2.1.2 Registration

When new users want to create an account, they are prompted to provide a valid email address and a secure password. After submitting this information, our system generates a verification email and sends it to the user's email. This email provides a verification code,

which they will enter in a specific form to verify their email. Once the process is complete, the users will be redirected to the login page.

2.1.3 Log out

The application lets users log out from every page of the application. Whenever they log out, they are redirected to the login page.

2.2 Project functionalities

Our application gives users the freedom to do almost anything with their projects. The main functions are listed below.

- Create a project by specifying its name, description, and visibility (public or private)
- Access a project and alter its contents, by adding files and folders
- View the contents of a public project with read-only permissions
- Invite users to collaborate with you on a project
- Delete projects that you have access to
- Search public projects

2.3 File functionalities

A project in our application consists of files and folders, which serve as the fundamental components of the user's workspace organization. Therefore, our application provides users with extensive capabilities to manage the building blocks of their projects. Users with editing privileges for a particular project can perform the following actions.

- Create and delete files and folders
- Upload files and folders inside the projects' hierarchy
- Preserve alterations made to files by pressing Ctrl + S

- Read the contents of the files within the code editor interface, which provides syntax highlighting for easier codebase navigation and debugging
- Edit the contents of the files within the code editor interface

2.4 GitHub Functionalities

Our application provides GitHub integration to streamline the development workflow, by reducing the need to switch between different platforms, as well as providing access to version control features. Here is a list of GitHub features

- Users can log in with their GitHub account in addition to traditional credentials, granting access to further functionalities.
  When a user logs in with GitHub, they are redirected to the GitHub authentication page, where they need to authenticate using their GitHub credentials. Subsequently, they are requested to grant permissions to our application, allowing access to their repositories, both public and private, as well as their user information.
  Upon successful authentication by GitHub, users are redirected back to the main page of the application, where they can:
- Download a repository from their GitHub account. This process, typically referred to as ”cloning” the repository, creates a local copy of the repository in the user’s account, allowing them to work with the project files locally.
- Initialize a new repository, again specifying a name, and a description and selecting visibility, either public or private
- Create commits for the changes to the project files and update the main’s branch head to the new object, similar to git push

2.5 Chat

In the project page, the user can see the collaborators of the specific project and if any, they can chat with them. The chat is implemented in order to facilitate real time communication between the developers, speeding the development time. The real time chat doesn’t save the

messages for later viewing, but only supports live chatting. In the chat room, all collaborators can chat, not just a specific number and every project has its own chat room.

Chapter 3

Used Technologies

In the process of choosing tools for developing a web application, it is wise to think about numerous aspects. In our case and considering the specific functionalities of our app, we decided to reject some languages and frameworks in favor of others. We weighed the popularity of various frameworks and libraries as a larger community increases the likelihood of finding solutions to encountered issues. During our research, we paid attention to the time of the latest publication of a tool. A considerable gap since the last update may indicate a lack of support, potentially leading to package deprecation. Finally, we evaluated whether a tool contributes to or diminishes the complexity of the application, making wise decisions based on whether the added benefits justify potential challenges.

3.1 Database

A database is an ordered collection of structured information, essential to a web application for storing persistent data. Without a database, we couldn't keep the users' authentication credentials, the project details, and much more information vital for our applications' functionality. Relational databases store the information in tables with rows and columns, making it easier to comprehend how different data structures relate to one another. PostgreSQL is an open-source relational database management system, compliant with SQL. It offers powerful features, such as support for complex queries, transactions, and data integrity constraints. [3]

3.2 Back-end technologies

The back-end communicates with the front-end, by sending and receiving data to render web pages. We developed it mostly with Go, a statically typed, compiled high-level programming language. Go shares syntactical similarities with C while offering features such as memory safety, garbage collection, structural typing, and CSP-style concurrency. CSP, an acronym for Communicating Sequential Processes, provides a way to deal with concurrent processes by communicating via channels rather than accessing shared memory. [4]

3.2.1 GORM

To speed up the development time we used GORM as an abstraction layer between the application and the database. ORMs, one of which is GORM, are often used in computer science to query and manipulate data from a database using an object-oriented paradigm. In this way, we avoid writing much SQL code and instead manage to interact directly with an object in our language of preference, which is Go. GORM has helped us save time since it forces us to write MVC code, thus making the codebase cleaner and it abstracts the database system, so we can change it whenever we want. [5]

3.2.2 Gorilla web toolkit

In our application, we made great use of the Gorilla web toolkit, which provides many useful packages for writing HTTP-based applications. We used the package gorilla/mux that implements a request router and dispatcher for matching incoming requests to their respective handler. We also made use of the gorilla/handlers and specifically the CORS function that provides Cross-Origin Resource Sharing middleware and finally the gorilla/websocket package to implement websocket connections in several parts of our application. [6, 7, 8]

3.2.3 JWT

As a Go implementation of JSON Web Tokens, we used the jwt-go package. JWT is an open standard, accessible and usable by anyone, that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This object typically consists of a header, identifying the algorithm used to generate the signature, a

payload, containing a set of claims, and a signature, which securely validates the token. In that way, the information contained in the token is verified and can be trusted.

By creating and returning a JWT when a user logs in, we achieve the authentication of an application. JWTs also can authorize a client who wants access to protected routes or resources. Simply by sending the JWT to the server and the server validating it, a user can access protected resources. Since the JWTs are self-contained, all the necessary information is there, reducing the need to query the database multiple times. [9] [10]

3.2.4 SendGrid

SendGrid is an SMTP service that allows emails to be delivered via their servers instead of our own. By providing an email template, we were able to send through SendGrid, emails to our application's users to reset their passwords and verify their email addresses. [11]

3.3 Front-end technologies

The front-end consists of the visual and interactive elements of a website that users interact with directly. It's a combination of HTML, CSS, and JavaScript, where HTML provides the structure, CSS the styling and layout, and JavaScript the dynamic behavior and interactivity.

3.3.1 React

For building our user interface, we chose React, an open-source JavaScript library, which is mainly concerned with rendering components to the DOM. A key advantage of React is that it only rerenders those parts of the page that have changed, avoiding unnecessary rerendering of unchanged DOM elements. React code is mostly made of components, which are modular and reusable. The components are rendered to a root element in the DOM using the React DOM library. When rendering a component, values are passed between components through props. Values internal to a component are called its state. [12]

3.3.2 JSX

React uses JSX as a syntax expression for JavaScript that lets you write HTML-like markup inside a JavaScript file, facilitating the creation of dynamic user interfaces. JavaScript XML, shortly known as JSX, is a syntactic sugar for JavaScript and is transpiled into nested JavaScript function calls to be executed in the browser.

3.3.3 DOM

The DOM, short for Document Object Model, is the data representation of the objects that comprise the structure and content of a document on the web. The DOM depicts the document as nodes and objects, facilitating the interaction with the page with a scripting language, such as JavaScript. Like that, we can access these nodes and update their styles, attributes, and content.

3.3.4 React DOM and react-dom

In React, however, you don't interact directly with the browser DOM, but with the virtual DOM, it's copy stored in memory. The virtual DOM is the reason React renders fast and efficiently. The update of the browser DOM is time-consuming since it lays out the elements and repaints the screen for the user to see changes. When using React, it updates firstly the virtual DOM, then compares the browser DOM and the virtual DOM to check for any updates made to the virtual DOM that must be reflected in the browser DOM. If there are any, it updates the browser DOM to match the virtual DOM. For the purpose of accessing the virtual DOM and modifying it, we use the react-dom package which serves as an entry point to the DOM. It is installed with npm. [13]

3.3.5 react-router-dom

Since React doesn't engage with routing, we must choose a package to enable client-side routing in our application. For this role, we used react-router-dom, which makes possible the navigation among views of various components in a React application by allowing the changing of the browser URL and keeping the UI in sync with the URL. react-router-dom provides various ready to use components and hooks that are very useful in web development. One of those is <Link> that lets the user navigate to another page by clicking it. [14]

3.3.6 Asynchronous functions

Often in your code you will come across asynchronous functions that are initialized at one time but are not finished until a specific job has been carried out. Thus, the results or errors of these functions are not available immediately after the call. There are two ways to handle these functions, promises and callbacks.

Callbacks are functions passed as arguments to other functions to be executed after a certain task has been completed. They are often used when handling asynchronous events such as a file I/O or a network request. They are quite handy in that they allow you to perform one task while another is waiting for a response, creating a better user experience and utilizing CPU time.

Figure 3.1: Callback function example

The previous piece of code shows a typical callback function. setTimeout() method sets a timer which executes a function or specified piece of code once the timer expires. In this case, the program will run the console.log() after waiting for 1000 ms.

3.3.7 Promises

Promises, on the other hand, offer a more elegant way to handle asynchronous code. A promise represents the eventual completion or failure of an asynchronous operation and its resulting value. In some cases where the value to be returned from a function or request is not known immediately, the asynchronous method returns a promise to provide the value or the error when it will be available.

A Promise is in one of three states:

- pending: initial state
- fulfilled: operation resolves successfully and returns a value
- rejected: operation failed with an error

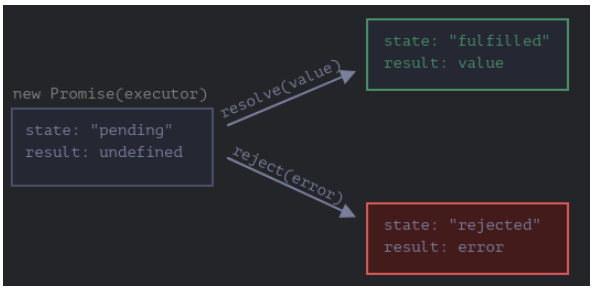

Figure 3.2: The different states of a promise

then/catch

When a promise is fulfilled, you can access the resolved data in the then() method. Then() in turn returns a new Promise, allowing you to chain calls to other promise methods. That way you can execute more than one asynchronous operations the one after the other, with the subsequent starting when the previous has succeeded and its result is available.

When you want to catch rejected promises, you can use the catch() method.

```
1  fetch('/users/user.json')
2      .then(response => response.json())
3      .then(user => console.log(user.name))
4      .catch(error => console.error(error))
5
```

Figure 3.3: Promise chaining

The code initiates a network request to the URL /users/user.json and returns a promise. The promise resolves with the response object when the remote server responds with headers, but before the full response is downloaded. To read the full response, we call the method json(), which in turn returns a promise that resolves with the result of parsing the body text as JSON. This code structure allows for promise chaining. If at any point of the promise chain, an error occurs, the catch() method will catch the error.

async/await

The async/await syntax is a simpler way to work with promises, as they resemble synchronous execution of code. The await keyword must be used inside an async function and makes a JavaScript function wait until the promise has resolved, and then resumes its execution with the response. That's powerful when you don't want to waste CPU cycles and instead your code can run other jobs while the async function is waiting for the promise to

be resolved.

```
async function fetchData() {
    try{
        const response = await fetch('/users/user.json')
        const user = await response.json()
        alert(user.name)
    } catch (error) {
        alert(error)
    }
}
```

Figure 3.4: Async/await syntax

As you can see, it is crucial to add the await code inside a try/catch block for proper error handling. This way if an error occurs at any stage of the promise chain, the catch() method will catch the error and the provided error handling function will be executed.

3.3.8 Axios

For making HTTP requests to the server, we used Axios, a promised-base HTTP Client, with many benefits over native JS interface, Fetch. One very powerful feature is the interceptors, middleware functions that allow you to modify requests and responses in a centralized manner. In our case, it was widely used for including an authentication token in every request's headers and refreshing the authentication token automatically after expiration date. [15]

3.3.9 Material UI

Material UI is an open-source React component library, ready to use for development and production. It's very easy to start working with it, just by importing the component you want to use. Among the components you can choose from, there is plethora for inputs, data display and navigation. [16]

3.3.10 TanStack Query

To handle requests to the server, we used TanStack Query, a library that facilitates fetching, caching, synchronizing, and updating server state. By server state, we refer to the data fetched from a back-end server or an API. TanStack Query works with any function that returns a Promise and uses the stale-while-revalidate caching strategy. TanStack Query offers features such as caching, retries, polling, data synchronization, prefetching, and many more

3.3.11 monaco-editor

Monaco Editor is a browser-based code editor powered by Visual Studio Code with many powerful features. It provides syntax highlighting, which was very important for our project and supports a variety of languages, including JavaScript, HTML, CSS, Java, etc. For our application, we used @monaco-editor/react, which is a wrapper for easy integration with React applications. [17]

3.3.12 web-sockets

Web-sockets allow for simultaneous bidirectional communication between client and server over a persistent, single-socket connection, allowing real-time data transfer from and to the server. Hence, they are widely used for live chats, terminals, to broadcast real-time events, to facilitate multiplayer collaboration and much more.

Web-sockets work over TCP connections that are kept open for data to be passed back and forth. The process of working with web-sockets is the following.

1. Open web-socket connection

   The handshake begins with an HTTP upgrade request from the client to the server, followed by the corresponding HTTP upgrade response from the server to the client.

2. Data transmission over web-sockets

   Server and client exchange messages over the persistent web-socket connections.

3. Close web-socket connection

   The web-socket connection can be terminated whenever either part of the communication wants to by sending a close message to the other side

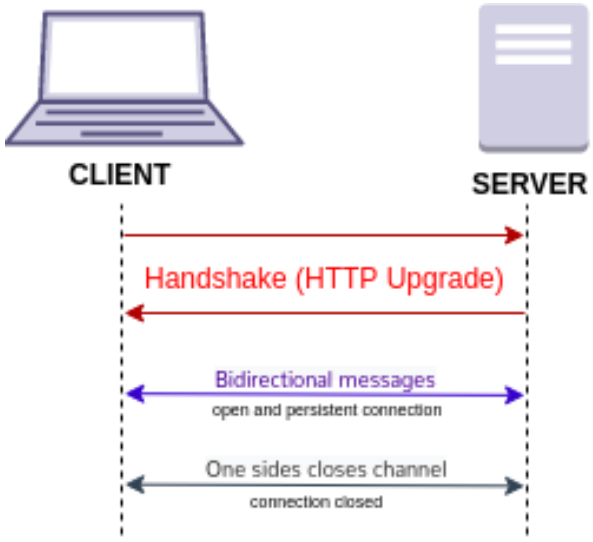

Figure 3.5: Web-socket protocol

3.3.13 react-use-websocket

We used the react-use-websocket package, which provides react hooks for web-socket integration. It simplifies the process of opening and closing a web-socket connection without the developer having to delve deep into the web-socket API, and it also offers features such as reconnection if the connection is lost and multiple components using a single web-socket.

Chapter 4

Technical Description

In this chapter, we aim to provide a comprehensive overview of the application's architecture and functionality. After reading this section, we are sure that even the unrelated reader, will understand how our application functions and how anyone can use it.

4.1 REST API

REST, which stands for Representational State Transfer, is a software architecture pattern that guides the design and development of the World Wide Web's architecture. APIs allow computer programs to communicate without requiring knowledge of system details. REST APIs combine these two concepts. REST APIs use HTTP requests to communicate and perform standard database functions, such as creating, reading, updating and deleting records within a resource, known as CRUD operations.

REST APIs rely on specific architectural constraints to work as predicted.

- Uniform Interface

  REST APIs provide a standard for client-server interaction, it's like a contract that everyone has agreed to work on to ensure smooth communication. Resources are identified by the URI standard and the operation on that resource is described by an HTTP method. Also, to separate the client from the application-specific URI structure, you may use hyperlinks and URI templates.

- Client-server decoupling

  Client and server apps need to be totally independent of one another in the REST API

standard. The client application should not communicate with the server application in any other way; it should only have knowledge of the URI of the requested resource. Similarly, the server application should only deliver the requested data to the client application via HTTP.

- Statelessness

  Each request must contain the information needed to process it, as the server application should not retain any data in relation to a client request.

- Cacheability

  Resources should be stored in the cache either on the client or the server side. Caching, makes the client side faster and more efficient, while enhancing the scalability of the server side.

- Layered system architecture

  Neither the server nor the client can tell whether it is communicating with the end application or an intermediary. There can be many intermediaries between the client and the server, without either side being aware of it.

4.2 Front-end

4.2.1 UI

Authentication Pages

Each application has one page for login, one for registration, and pages for password reset and email confirmation. These pages consist mainly of a form where the user fills in the appropriate fields. On the registration page, we add validity checks for invalid or already existing email addresses and a small password. On the login page, we handle and display an error in case of an invalid email or password or if the user is not verified.

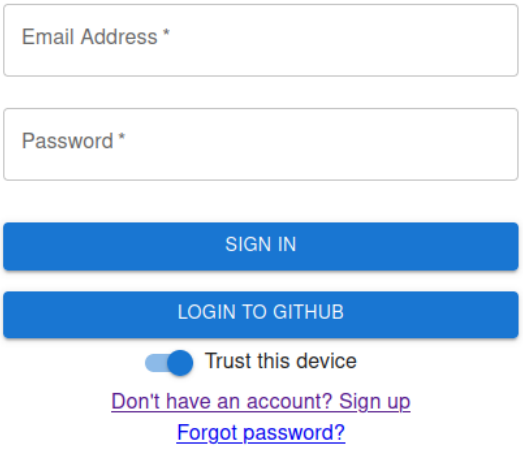

Figure 4.1: Login page layout

Card layout

In many parts, we make use of the Card component, which is a surface that displays content and actions on a single topic. We used it to display the projects of each user in their main page, to display information about the user, and also to create elegant views for the email confirmation and the reset password pages.

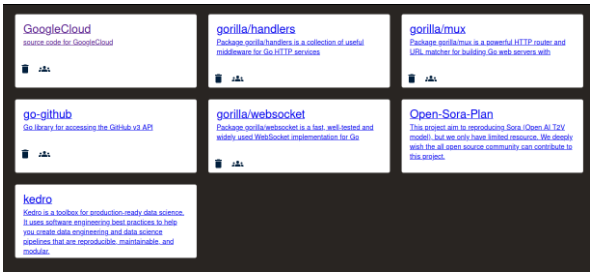

Figure 4.2: View of the project list

Menu

A lot of times, we utilized the Menu component, which displays a list of choices on temporary surfaces. With the MenuItem component, we were able to customize our Menus the way we needed.

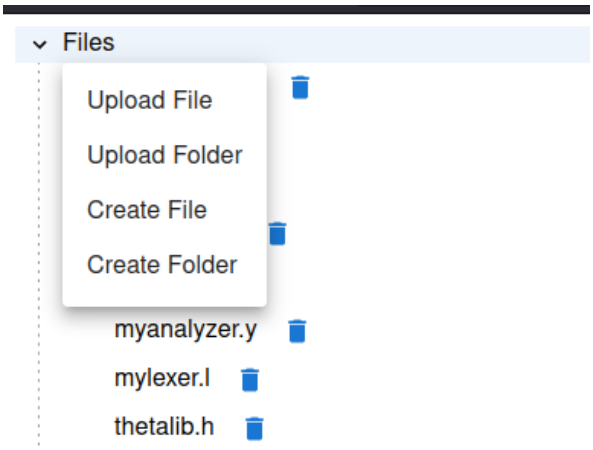

Figure 4.3: View of a Menu

4.2.2 Routing

Routing is a vital part of a web application's architecture and design. It's the reason that users can move between different pages and view the appropriate content linked to a specific path.

Nested Routes

With react-router-dom, we can create nested routes in our web application for authentication strategies, but also for better design and architecture and to improve performance.

- authentication

  We have created protected routes by wrapping the private components to an authenticating component that checks if the user is logged in and then allows access to the protected resources, thus providing security. Moreover, we have implemented persistent login authentication, where our app is able to remember the identity of a user between refreshes, reloads, or revisits unless the user logs out or the refresh token has expired. This is a task also accomplished with the help of nested routes.

- performance

  Nested routes improve performance by rendering only the required parts of the page and not rerendering the entire page, making the web application faster and more responsive.

- clarity

We have included all routes associated with authorization in a shared parent route, allowing for reusable components in many routes or sections of the app.

4.2.3 State management

While developing our application, we realised that some states need to be global while others need to be used by a plethora of components. In this case, we use two very helpful state managing tools.

TanStack Query

TanStack Query is a powerful tool for async state management. We use it to fetch data by defining a query key that uniquely identifies our query, so as long as we call the query with the same key in two different places, we will get the same data. Since React Query manages the server state, it assumes that the front end doesn't own the data. Thus, frequent fetching is needed to keep our data in sync with the back end. React Query handles this by letting you choose when to refetch your data depending on your application's needs. In our app, we have used TanStack Query hooks multiple times, and it has helped us with data fetching, caching, performance optimizations, updating outdated data in the background, and much more.

Context

There are cases where passing props to children components can become a very repetitive and tedious process, especially if there are many components between the parent and the children. With Context, the state becomes available to any component in the tree below the parent, without having to pass it explicitly. In our app, we used Context to pass the selected file to the code editor to display its contents and in Delete file and folder to delete the corresponding file or folder. Moreover, we utilized Context to make the access token and the GitHub token available to any component that needed it to authenticate the user.

4.2.4 GitHub API

In our app, we provide GitHub integration that complements and extends the user's workflow. A user logged in with GitHub can initialize a GitHub repository, commit changes, and download a remote repository to the application to use as a local copy. To accomplish these

functionalities, we use GitHub REST API. This API provides endpoints defined by an HTTP method and a path specific to the resource you want to retrieve or modify. Any request for a repository or any other information must be preceded by GitHub authentication. [18]

4.3 Back-end

4.3.1 Routing

The back-end uses RESTful routes to send and receive resources to the front-end. By mapping HTTP verbs and CRUD actions together, we are able to create a intuitive and simple API that the front-end can communicate with. We can organize our endpoints to teams that access the same resources such as :

- projects

  The routes associated with projects access the corresponding resources and perform related operations. These operations include creating a new project, retrieving the projects a user has access to, and copying a public project as your own private project, among others.

- files

  These routes manage file and folder-related functions. They perform necessary operations for the functionality of the code editor, such as creating a file, uploading a folder, saving the contents of a file, etc.

- authentication

  For authentication purposes, we use routes that enable user login, registration, logout, and the refresh of the access token. They are vital for the operation of any web application as they ensure that a user has an account and specific resources associated with this account.

- GitHub

  Since we offer GitHub integration, we have routes for login with GitHub, initialize a repository, commit changes to a repository, etc.

- invitations

  A user can invite collaborators to a private project and can accept or reject an invitation from another user. These are accomplished by the invitation routes.

- sockets

  This route handles the web-socket communication in the chat rooms.

4.3.2 Model-View-Controller

In the design of our application, we used the Model-View-Controller design pattern, that divides the program logic into three interconnected elements. The model, which contains all the data logic, the view, which is the front-end in our case, and the controller which links these two.

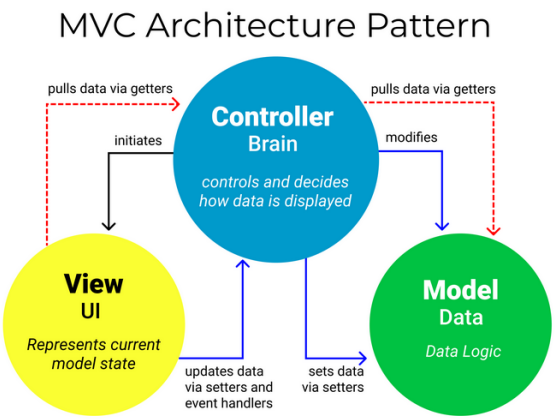

Figure 4.4: MVC design pattern

4.3.3 Models

The model contains and manages all the data in the application. The data can be stored in a database or retrieved from an API, in our case models are tables in our database.

User

The User model represents the identity of a user in our application.

- Id: The primary key
- Email: The email of the user, must be unique

- Password: The password used for login
- RefreshToken: The refresh token is created once the user logs in and is stored in the database until the user logs out or it expires. It is a JWT token that provides information about the id of the user and offers authentication when the access token expires
- GithubToken: It is the access token provided by the GitHub REST API to an authenticated user to access the specific user's resources
- Verified: It is a boolean value that shows whether the user has verified the email entered at registration
- VerificationToken: It is sent to the user's email and the user has to fill it in the appropriate form to verify the email
- ResetToken: It is sent to the user's email and the user has to fill it in the appropriate form to reset the password
- ResetTokenExpires: It shows if the reset token has expired. We initialize it to 1 hour. We don't want to last longer because it has access to sensitive data

Project

The Project model represents the projects that the users have access to.

- Id: The primary key
- Name: The name of the project. It doesn't have to be unique but it can't be null
- Description: The description of the project. It can't be null
- Public: A boolean value. If false, only the collaborators of the project have access to the project. If true, any user in the application has read-only access to the project, while the collaborators have permissions to modify the project as well

Access

The Access model shows who has access to which project and also the status of the invitations. This model has a compound key. A compound key is similar to a composite key in that two or more fields are needed to create a unique value. However, a compound key

is created when two or more primary keys from different tables are present as foreign keys within an entity. The foreign keys are used together to uniquely identify each record.

- UserId: It is a foreign key to the User's model id. With the ProjectId, they uniquely identify each row in the access table
- ProjectId: It is a foreign key to the Project's model id.
- Status: Shows the status of the invitations. It is an enum type, which allows us to limit the value chosen from a list of permitted values. In our case the two permitted values are accepted and pending. When an invitation has been sent and the user has not accepted it yet, it is pending. Once they accept it, or when a user creates a project, the status is accepted.

4.3.4 Controllers

The controllers' responsibility is to handle incoming user requests, execute the code defined by these requests, and return the appropriate data. Our controllers are quite similar to our route logic, in the sense that they are divided to project, file, authentication, invitation, GitHub and chat controllers.

The controller is the linking part between the model and the view. First, the front-end sends a request to the controller, then the controller interacts with the model to send and receive data and the appropriate data are sent to the view to render them.

Controllers handle errors that may occur while interacting with the models or if the data sent by the user were invalid. The controllers return informative messages and status codes that help in the debugging in development and in the display of the appropriate messages to the user in the front-end in production.

4.3.5 Chat

It is worth saying a few words about the chat implementation. Essentially, we didn't create a simple chat that everyone has access to, but chat rooms. There is one chat room for each project, so as the collaborators can communicate while developing. If a project has only one collaborator, we choose not to display a chat component, because it wouldn't make any sense.

When the server starts, we create a chat server which will manage the chat rooms. Essentially, the chat server is a struct that contains a map of rooms, a map of clients and a register

and an unregister channel. After the creation of the chat server, it is ready and listening for clients who want to register or unregister from a chat room. Whenever a user sends a message to the chat it identifies the room and the user id, if the requested chat room hasn't been created, it is instantly created and if the user is not registered as a client to the chat room, is instantly registered. Following, a message sent by the user will be redirected to every other user who has access to the chat room immediately.

Chapter 5

Deployment

Software deployment is the process of making an application available for use. Generally, the development of a software service is divided into three fundamental stages: development, staging, and production. Development focuses on coding and feature design and development. Staging involves rigorous testing and quality assurance. Finally, production makes the software ready for use. Deployment is the process of taking the code from development and moving it through the following stages until it reaches production. We want to devote an entire chapter to the deployment of our application since it was a demanding task that caused us to change part of the design and architecture of our application.

We will go through the process from the bottom level, which we consider the database, to the top level, the front-end.

5.1 Database

Our application uses PostgreSQL as its primary database management system, which is relational and SQL compliant. For deployment, we used the Azure Database for PostgreSQL Flexible Server. This service offers scalability, security, and the ability to monitor performance, and it is much more efficient for an application in production than hosting your database on your local system.

While setting up the database, we configure the compute and storage resources we need and the networking options, such as the firewall rules, which allow specific IP addresses to connect to the database. We also choose the region to host our database, which should be as close as possible to the server's location to reduce latency. After configuring these

choices, an SQL server instance is created, ready to connect to our back-end. The connection is accomplished by authenticating with a connection string that contains information about the database, such as port, db name, and credentials for the authentication of the server, such as user and password. [19]

5.2 Server

We chose to deploy our back-end to Google Cloud Run because it provides source-based deployment for the Go language, relieving us of the need to build and deploy a container image. Cloud Run has powerful features like a unique HTTPS endpoint for every service. It manages TLS for us, relieving us from the need to create certificates, and includes support for web-sockets. However, Cloud Run has a disposable container file-system, meaning it does not persist when the container shuts down.

The whole development process is done through a shell command. It is necessary to first install the Google Cloud CLI and then, while in your source directory, you run the following command: gcloud run deploy SERVICE –source . , where SERVICE is the name of your service, and the automatic deployment will begin. When the deployment is complete, you will see your service URL. Cloud Run offers continuous deployment from GitHub and is a fairly easy-to-use cloud service.

On the website, you can see the logs of your application, which helps with debugging, and you can deploy and run a new revision of your service instantly by changing the configuration. Before deploying a revision, we declare the environment variables we need for our server. [20]

Here is our backend url: `https://code-editor-eusldaqlhq-zf.a.run.app`

5.3 Storage

Our application relies on persistent data storage because the users upload and create files and folders and they need to be able to access them every time they log in. When we moved our server to Google Cloud Run, we realized it didn't offer persistent data storage. It was then a matter of urgency to find a cloud storage solution that would integrate with our existing back-end. We chose Cloud Storage because it allows us to store and retrieve any amount of

data at any time. And since we were already using a tool from the Google Cloud environment, it seemed only logical to use storage from the Google Cloud as well.

We used Cloud Storage to store the different projects of the users in buckets. We decided to use just one bucket for the whole application, to keep things simple, and within that bucket, we created a directory for each project, containing the relevant files and folders. [21]

In making this decision, we faced the challenge of changing a lot of the code we had already written. In most requests from the front-end, we would either retrieve or modify data from the local file-system. Now we had to convert that code to retrieve and modify data from the cloud storage. Moreover, although we had already implemented a terminal that users could interact with and compile their code, by moving our server to Google Cloud and using Cloud Storage instead of the local file-system, we could no longer use it as it was originally planned. We analyze this issue in more detail in the following chapter.

5.4 Front-end

Our application's front-end was deployed in Vercel, a cloud platform. The deployments are initiating automatically by pushing to the main branch in GitHub, enabling for continuous deployment and ensuring an up-to-date live version of our app. [22] Our application can be accessed via the following link: `https://thesis-frontend-snowy.vercel.app/`

Chapter 6

Conclusions

Now that we have taken the reader through the process of thinking, designing, and developing the application, it is time to reflect on the knowledge and experience we gained through this thesis and to be critical of what we could have done better. Overall, it has been a fascinating journey that allowed us to delve into many aspects of web development while also triggering us to explore topics we couldn't have expected when we started this thesis, such as concurrency and parts of the operating system.

6.1 Summary and conclusions

To sum up, this application was an interesting piece of work that combined a variety of technologies, such as React, Go, PostgreSQL, and TanStack Query. It was a great lesson that taught us to thoroughly research the technologies before using them and whether they are a good match for the functionality of an application. This application has achieved its goal of designing and developing a functional web application that does what it promises. It provides a reliable code editor and a collaborative environment for users to get the most out of it. With GitHub integration, real-time chat, search functionality, and project sharing, it is a worthwhile choice for an experienced programmer or someone taking their first steps into coding.

6.2 Future Improvements

Our most exciting future development is a terminal for the users to compile and run their code. We aim to provide a fully-featured experience for our users and we believe that creating

a terminal would do the magic.

In fact, a terminal was implemented utilizing multiple technologies. We used xterm.js which provided the UI of a real terminal in the front-end and worked like a bash where a user would type commands and see the results of those commands. Web-sockets handled the communication between the front-end and the back-end by sending messages back and forth. In the back-end, we had each user associated with their own terminal for security and isolation, and we managed the user connections with a global struct that held information about the number of open connections and the number of active clients.

A function was always running to read messages from the web-socket connection and write the message to the pseudo-terminal, and vice versa to read output from the pseudo-terminal and write it to the websocket connection. Go-routines and channels were used to ensure concurrent execution of these actions, and in cases where we wanted isolation, such as when writing to the pty, we used mutex locks.

A pseudo-terminal was used instead of the real local terminal because it allows for asynchronous, bi-directional communication between two or more processes. It is ideal for applications that want to use the shell to execute commands as the pseudo-terminal consists of two parts, the master and the slave. Any data written to the slave side is forwarded to the output of the master side. Conversely, any data written on the master side will be forwarded to the output of the slave side as shown in Figure 5.1. The master is not a pseudo-terminal in itself, it only allows messages to be received and sent to the slave.

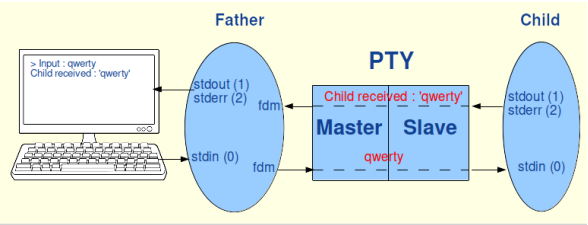

Figure 6.1: Overview of how a pseudo-terminal operates

However, our approach was deemed insufficient when we deployed the back-end of our application to Google Cloud Run and used Google Cloud Storage as the persistent storage where we store our projects. This means that it didn't have direct access to the project it needed to compile or run code from, as it did when it was stored on the local file-system. This problem made it impossible for the shell to work properly in our application. However, it is in our very near future developments, to correct this feature and make it available to our

users.

Bibliography

[1] Replit. https://replit.com/.

[2] Online ide. https://www.online-ide.com/.

[3] Postgresql. https://www.postgresql.org/.

[4] Golang. https://go.dev/.

[5] Gorm. https://gorm.io/index.html.

[6] Gorilla/handlers. https://pkg.go.dev/github.com/gorilla/handlers.

[7] Gorilla/mux. https://pkg.go.dev/github.com/gorilla/mux.

[8] Gorilla/websocket. https://pkg.go.dev/github.com/gorilla/websocket.

[9] Jwt. https://jwt.io/.

[10] jwt-go. https://pkg.go.dev/github.com/golang-jwt/jwt/v5.

[11] Sendgrid. https://sendgrid.com/en-us.

[12] React. https://react.dev/.

[13] react-dom. https://www.npmjs.com/package/react-dom.

[14] react-router-dom. https://github.com/remix-run/react-router/tree/main/packages/react-router-dom.

[15] Axios. https://axios-http.com/.

[16] Material ui. https://mui.com/material-ui/.

[17] @monaco-editor/react. https://www.npmjs.com/package/@monaco-editor/react.

[18] Github rest api. https://docs.github.com/en/rest?apiVersion=2022-11-28.

[19] Azure database for postgresql-flexible server. https://learn.microsoft.com/en-us/azure/postgresql/.

[20] Cloud run. https://cloud.google.com/run?hl=en.

[21] Cloud storage. https://cloud.google.com/storage?hl=en.

[22] Vercel. https://vercel.com/docs/deployments/overview.