Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Kasper Nilssen
Marko Stančić
Tor Anders Trondsgård
Daniel K. Bjørdal

Configuration Management for Complex Cloud Operations

Bachelor's thesis in Engineering in Computer Science
Supervisor: Rune Volden
May 2022

NTNU
Norwegian University of
Science and Technology

Kasper Nilssen
Marko Stančić
Tor Anders Trondsgård
Daniel K. Bjørdal

Configuration Management for Complex Cloud Operations

Bachelor's thesis in Engineering in Computer Science
Supervisor: Rune Volden
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

ABSTRACT

In the rapidly evolving world of microservices and multi-tenant applications, the demand for secure and scalable solutions for data management is higher than ever. This project aimed to meet that demand by developing a full-stack web application for managing configuration files used in distributed data pipelines. We used a range of modern technologies such as Next.js, NextAuth, Prisma, tRPC, and ChakraUI, to create a reliable and fast solution that fits the project's requirements.

Next.js, a React-based web framework, was used for the front-end development of the application. Its powerful features, such as server-side rendering, ensured a fast and responsive user interface. NextAuth was used for authentication, ensuring the security of the application. Prisma facilitated the management of the database, while tRPC was used for efficient communication between the front-end and back-end of the application. Finally, ChakraUI, a modern UI framework, was used to build a visually appealing and user-friendly interface, enhancing the overall user experience.

Our application's main function was to validate JSON files with JSON schemas, ensuring that the data used by our services was safe and in line with our customers' needs. We also made it easy to create, maintain, and validate configuration files, reducing the chances of human error or misconfigurations.

The project resulted in a reliable and scalable system for managing configuration files in a distributed data pipeline environment. While the solution requires additional work before it can be used in a production environment, the project laid a solid foundation for further development into a more complete solution fit for enterprise standards. The results of the project offer valuable research insights and a reliable product that can help developers store, edit, and validate JSON configuration file in cloud-based environments.

SAMMENDRAG

I dagens verden med mange mikrotjenester og flerbrukerapplikasjoner, er det økende behov for sikre og skalerbare måter å håndtere data på. Vi har nettopp jobbet med et prosjekt som møter dette behovet ved å lage en full-stack webapplikasjon som lar deg administrere konfigurasjonsfiler for distribuerte datarørledninger. Vi brukte en rekke moderne teknologier som Next.js, NextAuth, Prisma, tRPC og ChakraUI, for å skape en pålitelig og rask løsning som passer for prosjektets krav.

Next.js, som er basert på React, ble brukt for å bygge front-end av applikasjonen. Den har mange kraftige funksjoner, inkludert server-side rendering, som gir en rask og responsiv brukeropplevelse. NextAuth ble brukt for autentisering, slik at applikasjonen ble trygg og sikker. Prisma håndterte databasen, og tRPC gjorde kommunikasjonen mellom front-end og back-end effektiv. ChakraUI ble brukt for å gi et pent og brukervennlig grensesnitt som gjorde hele opplevelsen bedre.

Applikasjonen vår hadde som hovedfunksjon å validere JSON-filer med JSON-skjemaer, noe som sikret at dataene som ble brukt av tjenestene våre, var trygge og i samsvar med kundenes behov. Vi gjorde også at det var lett å opprette, vedlikeholde og validere konfigurasjonsfilene, slik at det var mindre sjanse for menneskelige feil eller misconfigurations.

Resultatet av prosjektet vårt er en pålitelig og skalerbar løsning for administrering av konfigurasjonsfiler i et distribuert datapipeline-miljø. Mens løsningen krever ekstra arbeid før den kan brukes i et produksjonsmiljø, la prosjektet et solid grunnlag for videre utvikling til en mer komplett løsning tilpasset bedriftsstandarder. Resultatene av prosjektet tilbyr verdifull forskningsinnsikt og et pålitelig produkt som kan hjelpe utviklere med å lagre, redigere og validere JSON-konfigurasjonsfilen i skybaserte miljøer.

ACKNOWLEDGMENTS

We would like to express our gratitude to our supervisor, Rune Volden, for providing us with guidance, feedback, and encouragement throughout the project. His support has been appreciated and has helped us greatly in achieving our goals.

We would also like to extend our thanks to FiiZK, and especially to Gerhard Melseth, Marius Lundbø, and Erik Mauring, for their cooperation and commitment to this project. Their invaluable insights, suggestions, and availability have been instrumental in the success of our work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **ABAC**: Attribute-Based Access Control
- **AJV**: Another JSON Schema Validator
- **API**: Application Programming Interface
- **CDN**: Content Delivery Network
- **CI/CD**: Continuous Integration and Continuous Delivery
- **CPU**: Central Processing Unit
- **CRUD**: Create, Read, Update, Delete
- **CSS**: Cascading Style Sheets
- **CSV**: Comma-Separated Values
- **DB**: Database
- **DMMF**: Data Model Meta Format
- **DOM**: Document Object Model
- **HTTP**: Hypertext Transfer Protocol
- **JSON**: JavaScript Object Notation
- **JWT**: JSON Web Token
- **MFA**: Multi-Factor Authentication
- **MVP**: Minimum Viable Product
- **OOP**: Object-Oriented Programming
- **ORM**: Object-Relational Mapping
- **RBAC**: Role-Based Access Control

- **SDG**: Sustainable Development Goals
- **SQL**: Structured Query Language
- **SSO**: Single Sign-On
- **tRPC**: TypeScript Remote Procedure Call
- **UI**: User Interface
- **URL**: Uniform Resource Locator
- **UX**: User Experience
- **VM**: Virtual Machine
- **XML**: Extensible Markup Language
- **YAML**: YAML Ain't Markup Language

CHAPTER

ONE

INTRODUCTION

1.1 Background

Configuration files are an essential component of software applications, providing a way to store and manage settings and options that determine how the application behaves. Configuration files can be used to specify parameters such as database connections, network settings, application behavior, and other parameters that can be modified without changing the application's code.

Configuration files can be implemented in various formats, such as XML, YAML, or JSON. JSON (JavaScript Object Notation) has become a popular choice due to its simplicity, flexibility, and compatibility with many programming languages and platforms.

While configuration files are a useful tool for managing application settings, they can also present challenges. One of the main challenges is managing the complexity and size of configuration files, which can grow rapidly as an application becomes more complex. Additionally, it can be challenging to ensure that configuration files are properly formatted, validated, and secured, to avoid errors, misconfigurations, or security vulnerabilities.

1.2 Problem statement

FiiZK is a company in the aquaculture industry that relies on JSON files for configuring its products. As the number of products and customers grows, FiiZK faces challenges in organizing, maintaining, and validating configuration data. The current manual approach for managing JSON configuration files is ineffective and can result in human errors and misconfigurations, leading to potential product failures and decreased customer satisfaction. In addition to that, managing JSON configurations with the manual approach is a boring and frustrating task, leading to a decrease in the employees motivation at work.

Therefore, FiiZK has tasked us with developing a web application that addresses

these challenges by providing a safe, scalable, effective and reliable solution for managing JSON configuration files. The application should allow for creation, maintenance and validation of JSON files, with a user-friendly interface to improve efficiency and reduce the risk of misconfigurations and human errors. The user interface should contain a editor where properties and values can be added and edited in a simple and easy way, and validate the configurations using a JSON schema containing the required properties and corresponding types. If the configuration is not aligned with the requirements of the schema, the user should be given an error message explaining whats wrong. The users input should be formatted according to the JSON syntax and stored in a JSON file, which is ready to be downloaded and deployed whenever the user wants.

1.3 Objectives and scope

The main objective of this project was to improve the efficiency and minimize the risk of errors when managing JSON configuration files. To be more specific, we were supposed to create a user-friendly web application that allows the user to create JSON configurations in a simple and effective way, and to give the user a way to check if the configuration is valid according to a JSON schema defining the requirements of the configuration. The app should provide a way to upload a JSON schema and create configurations that will be validated according to the selected schema.

The scope of the project included the development of a full-stack web application, where we are responsible of both the front-end and back-end of the application, including the user-interface, database solution, data validation logic and so on. Due to the fact that the objectives of the project could be achieved without the need for sensitive or confidential information from the company, we decided to not have a non-disclosure agreement with the client. As a result of this, the goal was to make a more generalized solution to the problem instead of a tailored solution according to their specific needs.

To achieve these objectives, we utilized modern technologies such as Next.js, NextAuth, Prisma, tRPC, and ChakraUI. These technologies allowed us to build a secure, scalable, and reliable application that met the project requirements. Our focus was on delivering a high-quality application that met the needs of our target users while remaining flexible and adaptable to changing requirements.

1.4 Report structure

In the following chapters, we'll conduct an in-depth examination of our development process and results. A detailed explanation of the technical details and how we worked collaboratively as a team will be provided. This deep dive will be organized in the following structure:

Chapter 2 - Theory: The theory chapter serves as a foundational introduction to the technical aspects and collaborative approaches of our solution. It covers relevant concepts and theories that underpin our development approach and will provide readers with the necessary background to understand the subsequent chapters.

Chapter 3 - Methods: The methods chapter details the methodology and materials used in the project, explaining the implementation of relevant technologies and concepts. It provides insight into the decision-making process behind the selection of methodologies and materials, offering a comprehensive understanding of the project's execution.

Chapter 4 - Results: The fourth chapter will present the final product of our development process, including the achieved outcomes and the decision-making behind them. It will provide a detailed description of the solution and its technical specifications.

Chapter 5 - Discussion: This chapter provides an analysis and reflection on the results presented in the previous chapter. It examines the strengths and weaknesses of the development process and offers insights for potential future improvements

Chapter 6 - Conclusion: The conclusion summarizes the findings and insights presented in the previous chapters. It provides a concise overview of the project's accomplishments, and outlines possible avenues for future development and improvement.

Chapter 7 - Societal Impact: The final chapter examines the potential economic and social implications of the project. It explores how our solution can contribute to addressing relevant societal issues and how it may benefit different sectors or communities.

CHAPTER

TWO

THEORY

2.1 Monolithic vs microservices vs serverless

2.1.1 Monolithic

Monolithic architecture is an approach to building software applications that involves building the entire application as a single, self-contained unit. This approach is characterized by its simplicity, as all the components of the application are integrated into a single executable. This makes it easy to develop and deploy, as everything is contained in a single package, that can be installed on a server or in the cloud. Additionally, because all the components are tightly integrated, there is no need for complex communication between different parts of the application, which can simplify development and testing. [1]

But, due to the application being all in one piece, it can become complex and difficult to maintain as it grows. Any changes or updates to one part of the application can have a cascading effect on other parts, making it difficult to implement new features. A single error in any component can bring down the entire application, making it less sturdy than other architectures. As a result, monolithic architecture may not be the best choice for large or complex applications where flexibility, scalability, and stability are critical.

2.1.2 Microservices

The architecture of a microservice based software involves building an application where many small independent services communicate with each other through APIs. Each service is responsible for a specific task, and can be modified independently from the other services. This method provides a way better flexibility and scalability compared to the monolithic architecture, allowing for easier implementation of new features. [2] The application itself is also much more resilient, as a fault in one service is less likely to bring down the entire application.

However, this method can also have its challenges. Due to the application being composed of many small services, there is a greater need for communication and

coordination between them. This can make the development process a lot more difficult, as each service must be designed to work with each other. Designing an application using microservices might therefore not be the right choice for smaller applications, where the added complexity outweighs the benefits.

2.1.3 Serverless

Serverless architecture is a cloud-based approach to building and deploying applications where the cloud provider manages the infrastructure. The developers focus solely on writing code for functions that can be executed independently in response to events. This approach offers benefits such as scalability, cost-effectiveness, and reduced management overhead, since developers only pay for the actual execution time of their functions. [3]

One of the main challenges with this architecture, is that developers must design their code to be stateless, since functions are executed independently and do not share state between executions. This can lead to additional complexity in the development process, as developers must manage stateful information in external data storage services. And since the code is broken down into small functions, one must also manage and deploy these functions individually, which requires additional effort from the developers. Despite these challenges, serverless architecture offers a compelling option for those that require high scalability and cost efficiency.

2.2 Modern web applications

Compared to traditional websites, modern web applications have advanced significantly. In the past, websites were often static and unengaging, whereas modern web apps are much more interactive and dynamic. They work seamlessly across various devices such as desktops, tablets, and smartphones, making them highly adaptable and user-friendly.

The development of modern web applications heavily relies on the use of front-end frameworks like React, Angular, and Vue [4]. They offer developers a wide range of tools to create user interfaces and experiences that are both engaging and dynamic. These frameworks also include additional features such as client-side routing, state management, and server-side rendering, which greatly enhance the performance and usability of web applications.

Modern web applications are also heavily dependent on APIs, which provide a standardized mode of communication between web applications and external services. [5] This, in turn, enables developers to build applications that are modular, flexible, and scalable.

The use of cloud computing services, such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform, is another crucial aspect of modern web application development. Cloud computing facilitates the easy deployment and scaling

of web applications without the burden of managing physical infrastructure.

Overall, modern web applications offer users an immersive and dynamic experience, and the capacity to scale and adapt to changing needs. The combination of modern front-end frameworks, APIs, and cloud computing services has revolutionized the way web applications are developed, and continues to advance the field of web application development.

2.3 JSON

JSON (JavaScript Object Notation) is a lightweight and widely-used data interchange format. It provides a simple and human-readable way to represent structured data, making it popular for data storage, configuration files, and communication between systems. JSON data is written as name/value pairs, with data separated by commas. Curly braces are used to hold objects, while square brackets are used to hold arrays, making JSON syntactically identical to the code for creating JavaScript objects. [6, 7] This allows for easy conversion between JSON data and native JavaScript objects.

```
{
  "name": "John",
  "age": 30,
  "hobbies": ["reading", "swimming", "hiking"],
  "address": {
    "street": "123 Main St",
    "city": "Springfield",
    "state": "CA"
  }
}
```

Figure 2.3.1: Example of a JSON object demonstrating name/value pairs, arrays, and objects

JSON's simplicity and flexibility make it highly compatible with different programming languages and platforms. Its textual format facilitates easy parsing and generation, allowing for seamless data exchange between systems. JSON's widespread adoption, particularly due to the rise of web technologies, like JavaScript, and widespread tooling support make it a versatile choice for various applications, including web APIs, client-server communication, and data serialization. This has made it a popular alternative to other data interchange formats of the past such as CSV (Comma-Separated Values) and XML (eXtensible Markup Language). [8]

2.3.1 JSON schema

JSON Schema is a powerful tool for validating and describing the structure of JSON data. It provides a standardized way to define the expected properties, types, and constraints of JSON objects. [9] JSON Schema acts as a blueprint or contract that allows developers to enforce data integrity and ensure that JSON documents adhere to specific rules.

JSON Schema follows a JSON-based syntax for defining schemas, making it easy to understand and work with. Schemas can include various keywords and validation rules, such as "type" to specify the data type of a property, "required" to define mandatory fields, and "pattern" to validate against regular expressions. Using these keywords, developers can express complex constraints within the JSON data.

2.4 Object oriented programming

Object-oriented programming (OOP) is a programming paradigm that organizes code into reusable objects that encapsulate data and behavior. It provides a structured approach to software development by modeling real-world entities as objects, which can interact with each other through defined interfaces. The fundamental principles of OOP revolve around the concepts of encapsulation, inheritance, and polymorphism. [10]

One of the key advantages of OOP is its ability to enhance code readability and maintainability. By encapsulating data and behavior within objects, OOP promotes the concept of modularization, allowing developers to focus on individual objects and their interactions. This leads to cleaner, more understandable code, making it easier to debug, modify, and extend software.

2.5 Functional programming

Functional programming is a programming paradigm that emphasizes the use of pure functions, immutability, and declarative programming style. In functional programming, functions are treated as first-class citizens, meaning they can be passed as arguments, returned as values, and stored in variables. [11] Pure functions produce the same output for a given input, without causing any side effects. Immutability ensures that data remains unchanged once created, leading to code that is easier to reason about and less prone to bugs.

Functional programming promotes code that is concise, modular, and easier to test and maintain. It encourages the use of higher-order functions, such as map, filter, and reduce, which operate on collections of data. These functions promote code reusability and enable developers to write more expressive and readable code.

2.6 Hooks

Hooks in React have revolutionized the way developers manage state and lifecycle in React applications. Introduced in React 16.8, hooks provide a functional approach to component logic and allow developers to encapsulate and reuse stateful logic without the need for class components. React offers a set of built-in hooks, such as useState, useEffect, and useContext, which cover common use cases for managing state, performing side effects, and accessing context within functional components. [12]

In addition to the built-in hooks, developers can create custom hooks to encapsulate reusable logic specific to their application's needs. Custom hooks enable the extraction of shared functionality into separate modules, promoting code reusability and maintainability. By following the naming convention and utilizing hooks as a convention, developers can create custom hooks that provide specific functionality and can be easily integrated into different components. [13]

Furthermore, numerous libraries and frameworks have embraced the concept of hooks and provide their own custom hooks as part of their APIs. These hooks extend the functionality of React and offer specialized features for handling their specific use cases.

2.7 Relational databases

Relational databases are a popular type of database management system that organizes data into tables with columns and rows, where each table represents an entity or concept, and each row represents a specific instance of that entity or concept. The relationships between tables are established through the use of foreign keys, which create links between tables and enable the retrieval of data from multiple tables using complex queries. [14]

Relational databases use Structured Query Language (SQL) to manipulate the data in the tables. SQL provides a rich set of commands for creating, modifying, and querying the tables, including commands for selecting, inserting, updating, and deleting data.

One of the main advantages of relational databases is their ability to ensure data integrity. This is achieved through the use of constraints, such as primary keys, foreign keys, and check constraints, which enforce rules on the data in the tables. Relational databases also support transactions, which allow multiple operations to be grouped together as a single unit of work that either succeeds or fails as a whole, ensuring that the database remains in a consistent state.

The relational model provides a high degree of data integrity and consistency, as well as powerful querying capabilities, making it well-suited for applications that require complex data structures and relationships. [15]

2.8 NoSQL databases

NoSQL databases are a type of non-relational database that are designed to handle large volumes of unstructured, semi-structured, and structured data. NoSQL databases are often used in applications that require high scalability, availability, and performance, and that deal with large volumes of data.

Unlike traditional relational databases, NoSQL databases do not use the structured query language (SQL) for data manipulation and retrieval. Instead, they use other methods, such as key-value pairs, document stores, or graph databases. This means that data in NoSQL databases can be organized in a more flexible way, making them suitable for use cases where data is highly variable or dynamic, like for storing JSON files for example. [16]

NoSQL databases are also designed to be highly scalable and distributed, meaning that they can handle large amounts of data and traffic across multiple servers or nodes by scaling horizontally. This makes them a popular choice for applications that require high levels of performance and scalability.

However, NoSQL databases also have some drawbacks. They do not provide the same level of data consistency and integrity as traditional SQL databases, which can lead to data inconsistencies or data loss in some cases. NoSQL databases also lack a standardized query language, which makes it difficult to perform complex queries across multiple tables or collections.

They also don't have transaction support and often do not offer the same level of security features as traditional relational databases. For example, access control and encryption may be less mature or more difficult to implement. [17]

2.9 Docker

Docker is a prominent containerization technology that allows developers to package and encapsulate applications and their dependencies into lightweight, portable containers. In contrast to virtual machines (VMs), which require a separate operating system to be installed, Docker containers share the operating system kernel of the host machine, making them more lightweight, efficient, and faster to start up and shut down. [18, 19] Docker containers can be easily deployed on different machines or cloud platforms, providing a consistent environment for the application to run in.

Docker containers are created from Docker images, which are essentially snapshots of a particular configuration of an application and its dependencies. Docker images are built using a Dockerfile, which specifies the base image, the application code, and any necessary dependencies or configuration. Docker also provides a range of tools and services for managing containers, including Docker Compose, which allows multiple containers to be managed as a single application, and Docker

Swarm, which provides orchestration and scaling capabilities for containerized applications.

2.10 Security

Security is a critical consideration in any web application project. Without proper security measures, your application can be vulnerable to attacks that compromise user data, cause service disruptions, and harm to the reputation of the application and its developers.

2.10.1 Authentication

Authentication is the process of verifying the identity of a user. This can be done using various methods such as username/password authentication, passwordless email authentication, multi-factor authentication (MFA), and single sign-on (SSO). Proper authentication mechanisms can prevent unauthorized access to your web application and protect user data. [20]

2.10.2 Authorization

Authorization is to process of determining whether a user has permission to access certain resources or perform certain actions. Role-based access control (RBAC) and attribute-based access control (ABAC) are commonly used authorization mechanisms. Proper authorization mechanisms ensure that only authorized users can access sensitive data or perform critical actions within the application. [21]

2.10.3 JWT

JWT tokens are used in web applications because they provide a secure and efficient way of handling user authentication and authorization. The server generates a token that contains user-specific information, which can be verified without having to verify the users credential with each request. JWTs are also signed, so they cannot be modified or changed without the server being able to detect the tampering. This provides a secure way of exchanging information between server and client. [22]

2.10.4 Tokens

Access tokens play a crucial role in modern authentication and authorization systems. They are based on the concept of token-based authentication, where a token is used as a proof of identity and authorization. [23] Tokens are typically issued by an authentication server after a successful authentication process, such as username and password validation or through a third-party identity provider.

To ensure the security of access tokens, they are often digitally signed or encrypted. This prevents tampering and unauthorized modifications to the token during transmission. The server can verify the integrity and authenticity of the

access token by validating the signature or decrypting the token using the appropriate secret key.

2.11 Agile methods

Agile development is a popular approach to software development that emphasizes iterative and incremental development. It involves working in short cycles or sprints, where each cycle typically lasts a few weeks. The aim is to deliver working software in each cycle, allowing for early and continuous feedback from stakeholders. The process is highly collaborative, with close interaction between the development team, product owners, and users. Agile development values responding to change over following a plan, and emphasizes the importance of adaptability, teamwork, and customer satisfaction. [24] This approach is widely used in programming due to its flexibility, ability to adapt to changing requirements, and emphasis on user feedback, leading to the development of high-quality software products.

2.11.1 Scrum

Scrum is an Agile framework for managing and completing complex projects in software development. It is a lightweight and flexible process that focuses on iterative development, continuous improvement, and team collaboration. [25] The framework involves a series of short sprints, typically lasting two to four weeks, where the team works to deliver a potentially shippable product increment. Scrum is centered around the Scrum team, which includes the Product Owner, Scrum Master, and Development Team, who work together to achieve the sprint goal. The framework emphasizes transparency, inspection, and adaptation, and encourages regular meetings, such as daily stand-ups and sprint reviews, to ensure that the team is on track to meet their goals. Scrum is widely used in programming due to its effectiveness in improving productivity, communication, and teamwork, leading to the development of high-quality software products. [26]

2.11.2 Code review

Code review is a software engineering practice that involves a systematic and critical examination of source code to identify and correct defects, improve code quality, and ensure compliance with coding standards and design guidelines. [27] Code review can be conducted in different forms, such as formal or informal, manual or automated, and individual or collaborative. The primary purpose of code review is to prevent defects from being introduced into the codebase and to promote continuous learning and improvement among developers. Code review is an essential component of the software development life cycle, as it can reduce the cost of defects and enhance the maintainability, reliability, and scalability of software systems. Moreover, code review can also facilitate knowledge sharing, team building, and communication within the development team. [28] Therefore, code review is a valuable technique that can help software developing teams achieve their goals of delivering high-quality, reliable, and maintainable software products.

CHAPTER

THREE

METHODS

3.1 Planning and design process

The development process involved several stages, including planning, design, implementation, testing and deployment. We started out with an introduction meeting with FiiZK.

3.1.1 First meeting

In the first meeting, we got to know each other. We went over all the necessary details of what the project should include. We discussed and cleared up various uncertainties the group members had about the project. We also got a better understanding of the purpose of the project, as they explained their dissatisfaction with their current methods for configuration file management.

3.1.2 Pre-project plan

The pre-project plan was an essential component of our project management approach, providing a roadmap for the successful execution of the project. It served as a guiding document that outlined the project's objectives, scope, timeline, resources, team roles and key milestones.

Furthermore, the plan served as a communication tool, facilitating discussions and negotiations with the client, and stakeholders. It ensured that everyone involved in the project had a shared understanding of the project's scope, objectives, and deliverables, and helped us manage expectations effectively.

3.1.3 Design

Creating wireframes with the feedback of the client before coding was a deliberate decision made to ensure effective communication and alignment with the client's requirements and vision. By creating wireframes, we wanted to visually represent the proposed design and functionality to allow the client to provide feedback and suggestions early in the development process, allowing for any necessary modifications or adjustments to be made before proceeding with the actual coding.

3.1.3.1 Target audience

An important element in the design decision-making process was the identification of the user group. Specifically, the intended users are individuals who predominantly operate within a computer-based environment and are accustomed to user interfaces that prioritize sleek, futuristic aesthetics and efficient functionality. In light of this, our design sought inspiration from similar interfaces.

3.1.3.2 Wireframes

Wireframes played a critical role in the initial stages of our project, where we worked closely with the client to agree on the design and functionality of the web application. These sketches have also helped us to quickly iterate through different design ideas and gather feedback from the client. This ensured that the client was fully aware of what to expect from the finished product, and it also helped our development team to work more efficiently by having a clear understanding of the requirements from the start. The wireframes can be seen in Figure C.1.

3.1.4 Client meetings

We held regular meetings with the client to plan the next sprint and discuss important decisions. These meetings were vital in ensuring effective communication and collaboration between our team and the client.

During the sprint planning meetings, we reviewed the progress of the previous sprint, identified any challenges or roadblocks, and set clear objectives and goals for the upcoming sprint with the client and the supervisor. This also included clarifying project requirements, design choices, and technical decisions.

3.1.5 Research

The client granted us the autonomy to determine the most appropriate technologies to utilize for the web application, server, and database. Our initial meetings were focused on this topic.

The decision on the tech stack was made after careful consideration and evaluation of various factors during the early stages of the project. Our team conducted thorough research and analysis to identify the most suitable technologies for the project's requirements, taking into account factors such as scalability, performance, ease of use, maintainability, industry trends, and compatibility with the project's goals and objectives. We also considered the team's expertise and familiarity with different technologies to ensure smooth development and implementation.

3.1.6 Current state-of-the-art

During the initial phases of our bachelor's project, we conducted a comprehensive review of state-of-the-art projects that were similar to our own. The purpose of this review was to gain insights into existing solutions, identify potential gaps or

opportunities for improvement, and ensure that our project would be innovative and relevant within the current landscape.

Despite a thorough search, we were unable to identify any other products that fully met our specific needs. As a result, we decided to explore NoSQL databases for design inspiration, as they offer a structure that closely resembles a JSON file. Although we did not find any existing products that fully met our project's requirements, we did come across a few products that addressed certain aspects of the problem. One such example is a web application designed specifically for JSON schema validation. This tool allows users to input a JSON schema and configuration, which will be validated against the schema to ensure compliance. While the web application mentioned provides an important feature for validating JSON schemas, our project aims to offer a comprehensive tool for managing JSON configurations. In contrast to the existing tool, where JSON configurations have to be created manually or with a separate tool before being validated, our product provides a centralized approach where the entire process of configuration management can be carried out seamlessly with a single tool. This gives our product a significant advantage in terms of efficiency and simplicity, offering a more streamlined and effective solution for JSON configuration management.

3.1.6.1 Firebase

As part of our research, we investigated Firebase for design inspiration. We were interested to see how one of the most popular and widely used NoSQL databases handles traversal through a recursive or hierarchical JSON-like structure. Figure 3.1.1

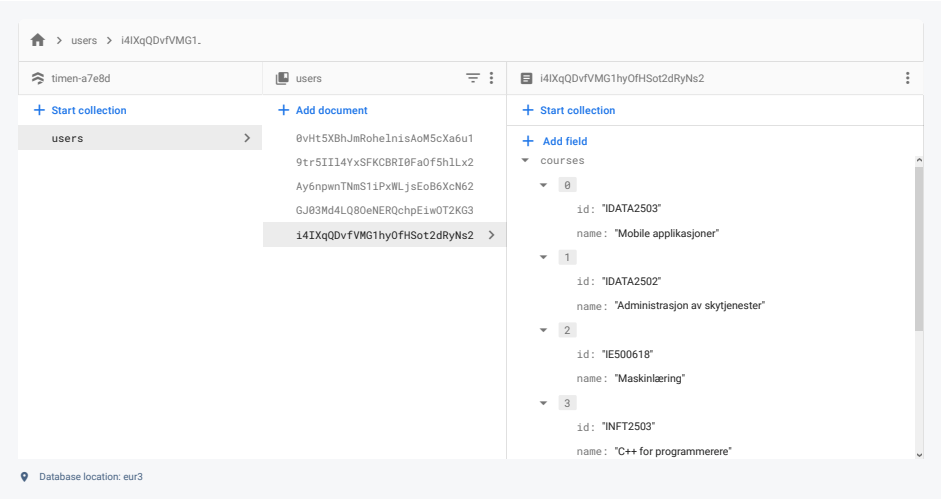


Figure 3.1.1: Firebase Cloud Firestore UI

We conducted a detailed analysis of Firebase's methods for displaying, traversing, and updating data within a recursive or hierarchical structure. We then adapted and applied these concepts to our own design, drawing inspiration from Firebase's proven methods to efficiently display and manage data with infinite nesting levels.

3.2 Tools and technology stack

3.2.1 Next.js

One of the requirements for our web application was that it needed to be performant, scalable, and compatible across multiple devices and platforms. Next.js enabled our team to achieve these goals while streamlining the development process. The project had a strict deadline, and the client wanted the application to be available as quickly as possible. With Next.js, we could leverage its hybrid rendering approach and built-in features to expedite the development process, ensuring that the project would be completed on time.

Next is a modern web development framework built on top of React developed by Vercel. In Next, user interfaces are constructed using components, which are self-contained, reusable pieces of code which makes it easy to build complex views while maintaining a clean and modular codebase. The development experience with Next.js is further enhanced by features such as hot module replacement and automatic code-splitting. [29]

3.2.2 NextAuth

We've chosen NextAuth as our authentication solution. One of the primary factors behind choosing NextAuth was its seamless integration with Next and its ability to efficiently implement an authentication system without facing compatibility issues or the need for extensive customizations.

NextAuth offers built-in support for a wide range of popular authentication providers, such as Facebook and Microsoft, in addition to those specifically requested by the client, such as Google, GitHub and a password-less email sign in. [30]

3.2.3 Chakra UI

The selection of Chakra UI as the UI library was based on its vast set of customizable components that adhere to modern UI/UX best practices. These components facilitate the creation of a visually appealing and user-friendly interfaces for the application. [31] Additionally, Chakra UI offers seamless integration with Next.js and React frameworks.

3.2.4 tRPC

The decision to utilize tRPC as the chosen tool for building the API was based on its modern and robust features that allow for the implementation of type-safe APIs across both the front-end and back-end components of the application. [32] With the use of TypeScript with tRPC, the API development process benefits from enhanced typing and code completion capabilities, ensuring a higher degree of code reliability and maintainability.

3.2.5 TanStack Query

We leveraged TanStack Query, a powerful data fetching and state management library, in combination with tRPC to handle data operations in our web application. tRPC utilizes TanStack Query to provide convenient query hooks for each endpoint, streamlining our server communication.

With tRPC's integration of TanStack Query, we gained access to query hooks that corresponded to our application's endpoints. These hooks facilitated type-safe and declarative data fetching from the server, ensuring efficient and error-free communication. [33]

Furthermore, TanStack Query's caching mechanisms were seamlessly integrated into the query hooks provided by tRPC. The query hooks automatically managed server response caching and updated the cache when data changes occurred. This optimized caching approach minimized redundant network requests and significantly improved our application's performance.

3.2.6 Prisma and the DB solution

Prisma is an open-source ORM library that simplifies database access and management. It provides an abstraction layer between the application code and the database. It supports popular databases like PostgreSQL, MySQL, SQLite, and more.

Prisma also has a type-safe query builder that reduces the risk of SQL injection attacks and completely removes the need to write raw SQL queries that could be error-prone and time-consuming. It's schema definition language can sync the current schema with the database schema which greatly streamlines migrations, simplifying the process of evolving and maintaining the database over time. [34]

In addition to selecting Prisma as our ORM solution, the team has decided to host the database on a free Render.com PostgreSQL instance. This decision meets our project needs and budget, delivering a quality and secure solution for our client cost-effectively. [35]

3.2.7 Why store JSON in a relational DB

Using a relational database to store JSON as a string provides several benefits over using a NoSQL database for this purpose. [36] Although NoSQL databases are well-equipped to handle unstructured data such as JSON documents, they lack the rigidity of SQL, which can result in data inconsistencies and compromised data integrity. In contrast, a relational database with SQL provides features such as automatic database delete cascading, constraints, and data types that help ensure data consistency and integrity. Additionally, SQL's rich querying capabilities make it easier to retrieve data in complex ways, which can be particularly useful for complex views or operations. The theory behind this was discussed in section 2.7 and section 2.8.

3.2.8 Ajv

Ajv (Another JSON Schema Validator) is a widely used and popular JavaScript library that provides comprehensive support for JSON schema validation. It is known for its efficiency, performance, and ease of use, and was therefore deemed an appropriate choice for our project. [37]

One of the primary factors that influenced our decision to select Ajv was its capacity to accurately validate JSON files against a JSON schema while generating comprehensible and user-friendly error messages. Additionally, it includes a path that enables us to easily navigate to the location of any errors.

3.2.9 Vercel

We have chosen to deploy our application on Vercel, a cloud platform that offers both static frontends and serverless functions and offers seamless integration with Next.js. [38] With Vercel, we can run and scale our code on-demand without the need to manage our own infrastructure or provision servers, and it automatically scales the application based on demand, which gives high availability and performance without any manual configuration. More theory behind this can be found in subsection 2.1.3 on serverless.

Vercel's tight integration with Next.js provides numerous benefits for our deployment process. Firstly, it simplifies the deployment workflow by offering built-in support for Next.js applications, allowing us to deploy our code with ease. Vercel's automatic code-splitting and server-side rendering capabilities optimize the performance and loading speed of our application. Additionally, Vercel leverages edge computing technology, running our code on the Edge, which brings the application closer to the end user, resulting in reduced latency and improved performance. [39]

Vercel's CDN (Content Delivery Network) capabilities further enhance the loading speed of static content, ensuring a smooth user experience. By caching and serving static assets from edge locations across the globe, Vercel minimizes the time required to fetch resources, providing faster page load times for users regardless of their geographical location.

One notable feature that Vercel offers is the ability to preview changes before merging them into the production environment. Each branch in our code repository is assigned a unique preview link by Vercel, allowing us to test and review the branch in a production-like environment without affecting the main application. [40] This preview functionality enables us to validate the changes and ensure they meet our requirements before being merged into the main branch.

3.2.10 Docker

Docker is a popular choice for deploying solutions because it provides a consistent and reproducible environment for applications to run in. This makes it easy to deploy, scale and manage applications across different environments. [41] The

team's familiarity with the platform and the client's requirement to run our solution in their Kubernetes cluster were also important factors in choosing Docker to containerize the app.

3.2.11 GitHub, Git-tags and other collaboration tools

3.2.11.1 GitHub

The team has selected GitHub as the designated code collaboration tool for our project. Leveraging the functionalities offered by GitHub has facilitated seamless collaboration among team members working on the same code-base. The team also has extensive experience with the tool, it's pull requests system and the GitHub Actions ecosystem.

3.2.11.2 Git Tags

We have utilized Git tags to capture and document progress. They offer a lightweight and efficient way to mark specific points in the version history of our code repository. By employing git tags, we can effectively track and reference the state of our code-base at the end of each sprint, making it easier to identify and review the changes made during that particular iteration upon the project's completion.

3.2.11.3 Figma

In the initial stages of the project, Figma was employed as the design tool. It is a collaborative and cloud-based design tool used to design iterations and get feedback from the client.

3.2.11.4 Jira

The decision to use Jira as our project management tool was driven mainly by it's comprehensive features set. It is a tools that allows teams to monitor both completed and outstanding tasks, track issues and their progression as well as log working hours. [42] Jira's issue board, a visual representation of the project's tasks and their status, makes it easier to track progress and prioritize tasks.

3.2.11.5 Confluence

Confluence was used to create, organize, and share information within the team. It has been primarily utilized to facilitate the sharing of meeting notes, tracking of minimum viable product (MVP) specifications, and documentation of sprint meetings.

3.2.12 Code style

3.2.12.1 Prettier

We chose to use Prettier as a code formatting tool for our project due to its ability to enforce a consistent code style across the team. Prettier automatically formats code according to predefined rules, ensuring that all team members follow the

same code formatting conventions. This helps to eliminate inconsistencies when committing to the repository, improves the overall code readability, and makes it easier for team members to understand and collaborate on each other's code.

3.2.12.2 ESLint

We've decided to utilize ESLint along with the eslint-config-next configuration because it enforces the code style and consistency of the codebase, which makes it easier to read and maintain. The eslint-config-next configuration also includes best practices and rules specific to Next.js development, ensuring that our code adheres to the framework's conventions. [43]

3.2.13 Algorithms and techniques used

In developing our project, we tried and incorporated advanced algorithms to enhance its functionality and performance. These algorithms play a crucial role in several key areas like improving the overall user experience.

3.2.13.1 Search

In the project, we investigated the performance of several advanced search algorithms. The idea was to allow the user to miss spell the name of a configuration and still get the desired search result.

We explored the applicability of advanced string matching algorithms such as Jaro-Winkler and Levenshtein distance in our application. These algorithms are commonly used for comparing and measuring the similarity between strings, which can be useful in various scenarios, including data de-duplication and fuzzy matching. [44] However, after thorough testing and evaluation, we determined that these algorithms did not yield satisfactory results for our specific use case. The performance, in terms of result quality, of the search algorithm was found to be no better or even worse than that of a naive algorithm.

We therefore made the decision to opt for a simpler and more straightforward approach, normal string comparison. By using direct string comparison methods, we were able to achieve the desired level of precision and accuracy in determining string equality. This approach proved to be more effective on the test data we used.

3.2.13.2 Color generation

For generating colors programmatically in our application, we developed a custom algorithm inspired by the principles outlined in an article by Martin Ankerl titled "How to create random colors programmatically." The article presents various techniques for generating visually appealing and diverse colors. [45] Our algorithm took inspiration from these techniques to create a color generation process. The algorithm was implemented using the seedrandom JavaScript library, which provided a reliable source of pseudo-randomness.

The development of the custom color generation algorithm was driven by the client's requirement to ensure visual distinctiveness between the different templates and configurations within our application. The client emphasized the importance of having visually unique and easily distinguishable color schemes for each template and configuration.

3.2.14 Communication

We employed a range of communication tools to facilitate effective collaboration. For internal communication, we utilized Meta's Messenger and Discord, which served as essential channels for planning work sessions, brainstorming ideas, and sharing various files such as concepts and images among team members.

Externally, we employed email as a formal means of communication with the client and our supervisor. Email was utilized for various purposes, such as scheduling and confirming meetings, sharing meeting agendas, and contacting the external parties for any clarifications or inquiries between the scheduled meetings.

3.3 Agile development and work allocation

Following consultation with both the project supervisor and client, the decision was made to utilize the Agile methodology for the project's development. This approach emphasizes an iterative and incremental process, allowing for flexibility and adaptability throughout the project's life-cycle.

3.3.1 Work allocation

Our project aimed to provide each team member with opportunities to work on various aspects of the project, from front-end to back-end and from design to implementation. It was believed that this approach would promote continuous learning, skill development, and equality among team members. We used Jira to organize the project into different tasks, so the members of the group could assign tasks to themselves and track their progress.

3.3.2 Sprints

We had set up regular biweekly meetings with the client at the end of each sprint to clarify the goals for the next sprint and to ensure that everyone was aligned with the project's direction. On these meetings we also gathered the client's feedback and set the goals for the next sprint, and made sure that everyone was aware which goals they had to complete until next meeting.

This approach allowed us to continuously refine and improve the project based on the client's feedback and requirements, while also ensuring that we were meeting their expectations at each stage of development.

3.4 Development process

3.4.1 User Experience

As previously mentioned, we conducted research on existing tools that had efficient methods of navigating recursive data. Our goal was to create a professional and polished tool that would provide users with an enhanced user experience. To achieve this, we opted to develop our own browser despite being aware that it would require significant additional time and effort.

3.4.2 Quality assurance

Continuous Integration (CI) played a crucial role in ensuring the quality of our project. We utilized CI to automatically run a build job on every commit, which helped to catch errors and notify the team of any build issues. We ran our CI pipeline on Vercel, a popular tool for CI/CD tool for Next. Additionally, Vercel also offers a preview deployment link from any branch, allowing us to preview our changes before merging them into the main branch.

We also utilized ESLint, a popular TypeScript linter tool, to catch potential bugs or errors before they can cause issues. To ensure that our code meets the highest possible quality standards, we customized our linting preferences to be particularly stringent.

3.4.3 Code review

Our team used a code review process to review each other's code and catch any potential issues or misunderstandings before merging into the main branch.
Experienced team members, are granted the authority to approve their own pull requests, however, for less experienced team members, an additional step is implemented wherein another team member is tasked with reviewing the code, and subsequently approving or disapproving the merge request. This process has been instrumental in ensuring that the produced code meets high-quality standards, contributing to the overall robustness of the project.

CHAPTER

FOUR

RESULTS

4.1 Scientific results

4.1.1 Easy and efficient to use

One crucial aspect of the product's design was to ensure that no action felt tedious. Given that the target audience consisted of individuals who were technologically adept, it was important to assume that they were accustomed to using efficient applications. With this consideration in mind, it was necessary to ensure that every action was perceived as a necessary step in the process. Minimizing the number of clicks required for task completion was essential in preventing user frustration, particularly for repetitive tasks.

To address this issue, certain dialogs were condensed. For instance, when creating a new template or configuration, the dialogs were modified from having a "Next" button that would take users to the subsequent step, to having all inputs displayed within the same window.

Another issue that was discussed in relation to ease of use, was the navigation within the configuration browser. As this is the primary operation of the application, it is important for navigation to feel intuitive and efficient. Any transition between mouse and keyboard usage could potentially impede efficient users. As such, we added the option to navigate and modify configurations using only the keyboard as a stretch goal. See Figure B.2 in the appendix for MVP requirements and stretch goals.

4.1.2 JSON, upload, and validation

We developed a unique tool that enables the storage, editing, and verification of JSON files. This tool serves as a valuable asset for users who need to work with JSON configurations, providing them with an intuitive interface to manage their data effectively. As far as our research indicates, this tool is one of the first of its kind, offering a comprehensive solution for JSON management.

One of the key features of our tool is its ability to validate JSON files against predefined schemas. We can ensure that JSON data adheres to the specified structure, rules, and constraints. This validation process significantly reduces the chances of encountering errors or inconsistencies within the JSON configurations.

The dramatic impact of our tool is evident in the subsequent chapter, subsection 5.1.1, where we demonstrate its effectiveness in minimizing the time required to fix errors in JSON files. By identifying and highlighting validation errors, users can quickly locate and rectify issues, thereby streamlining the correction process and increasing overall productivity. This tool not only simplifies JSON management but also contributes to improving the overall quality and reliability of JSON configurations.

4.2 Engineering results

This section can be found in appendix E, system documentation.

4.3 Engineering results - User Facing

This section provides a detailed analysis of the engineering results obtained from the web application.

4.3.1 Logging in and staying logged in

We're using NextAuth with a JSON Web Token, which provides a secure and efficient way for users to log in and staying logged in. The tokens are safely stored in the cookies, which allows the user to remain logged in even after they close and reopen the application. These tokens also help us when fetching data, to verify if the user has access to the given data. The theory behind this was discussed in subsection 2.10.4.

4.3.2 Adding a template (JSON schema)

An important part of the app is to add a schema to create a template. These templates are like different projects, where each project has its own set of rules that must be followed to be considered valid. The steps to create a template are shown in Figure 4.3.1. To make a template, the user simply has click the "Add Template" button, upload a JSON file, choose a name for the template, and submit it. It is worth noting that currently, there are no constraints in place to prevent the uploading of a standard JSON file instead of a JSON schema. In such cases, the configurations within the project will not be considered valid.

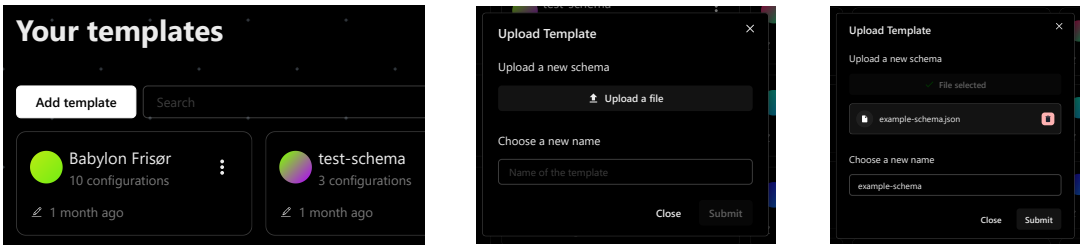


Figure 4.3.1: This figure illustrates the process of uploading a template (JSON schema).

4.3.3 Adding a configuration (JSON file)

When the user opens a template, they are presented with different configurations contained in that template. These are the different JSON files that the user has uploaded and want to verify the content of. The process of uploading a new configuration is similar to that of adding a template. However, in this case, the user is initially presented with three options to choose from. See Figure 4.3.2 and Figure 4.3.3 for visual representation of the steps. The first option is to create a new configuration from scratch. This will create an empty JSON file, where the user can add all the necessary fields. The second option is to copy JSON content from an already existing configuration, while the last option is to upload a JSON file.

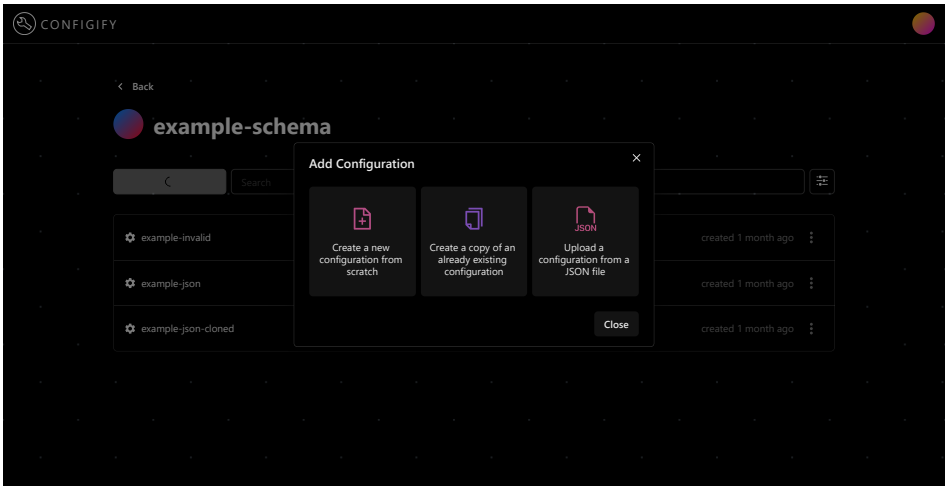


Figure 4.3.2: Figure showing the dialog for adding a new configuration

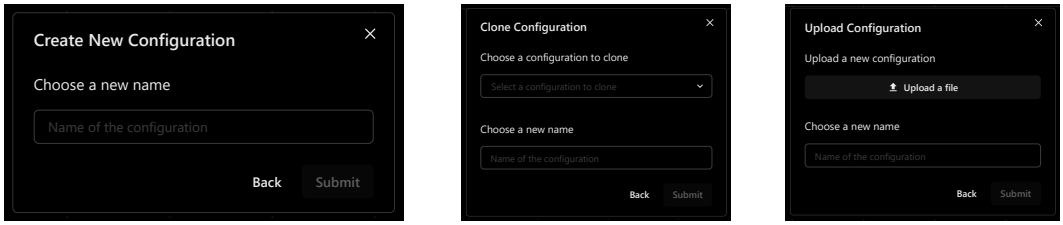


Figure 4.3.3: A figure showing the 3 ways to add a configuration. The different dialogs are displayed upon selecting the corresponding option from Figure 4.3.2

4.3.4 Browsing configurations

When the user clicks on a template they are directed to the page of that template. One will then be able to browse through all the configurations of that template. The configurations are sorted by last modified, the most recent modified will appear on the top.

4.3.4.1 Search

Due to the uncertainty related to the number of configurations to be added, it could potentially be hard to locate the specific configuration the user is looking for. To address this concern, we implemented a search functionality within to allow users to find specific configurations. Our approach utilized a simple search algorithm that involved checking if the search term appeared within the title of the template. The search algorithm was also case-insensitive to improve usability. The same is also used for the templates page. See Figure 4.3.4 for a simple demonstration of the search functionality.

Figure 4.3.4: Search functionality demonstration. A query for the term 'cloned' yielded a single result.

4.3.4.2 Filter

An important part of the application is the ability to update a schema. In relations to this, when a schema is updated, all the existing configurations for that template will be re-validated. This means that a lot of the configurations might be changed to being invalid. To allow the user to quickly find invalid configurations, we created a simple filter, so that the user can switch between displaying all, valid or invalid configurations. The filter button also displays a tiny icon on the button to show which filter is currently selected, see Figure 4.3.5 and Figure 4.3.6 for a demonstration.

4.3.4.3 Download

One of the most important aspects of the application is the ability to download the configurations. When the user modifes a configuration, they need to be able to download it. In the drop-down menu for the different configurations, we added a button that allows the user to download the JSON file.

4.3.4.4 Other

In the drop-down menu we also added the option to edit the configuration. This allows the user to change the name, or to upload a new configuration. If the user

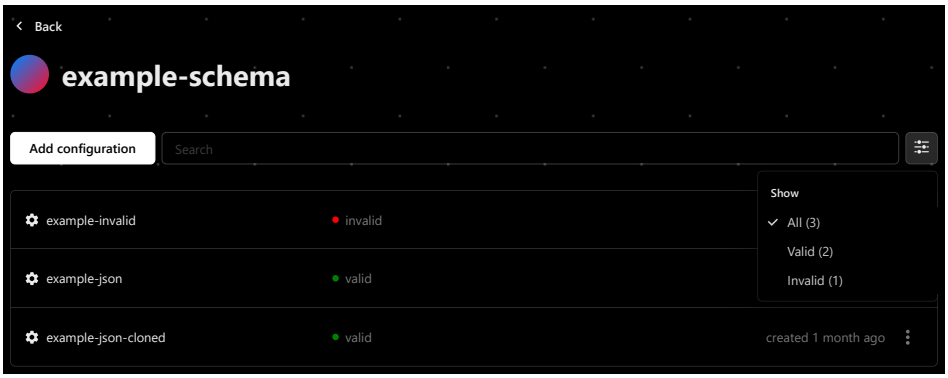


Figure 4.3.5: A figure showing the filter dialog

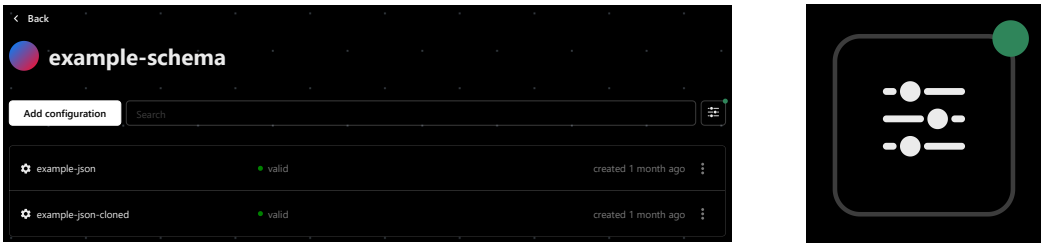


Figure 4.3.6: A figure demonstrating the filter functionality. On the left, only valid configurations are displayed, while the right figure shows a close-up of the filter button with an added dot to indicate that a filter has been applied

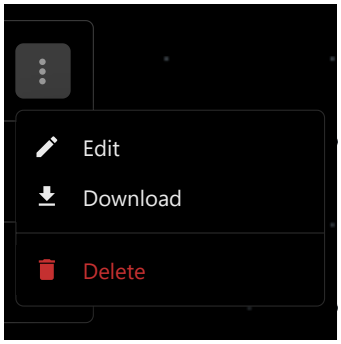


Figure 4.3.7: Configuration menu close-up. The menu displays options to rename, download, and delete the configuration

chooses to upload a new configuration, it will overwrite the existing one. The final option in the menu is the delete button. This triggers a dialog prompting the user to confirm their intention to delete the configuration. This serves as a safety measure to prevent accidental deletion.

4.3.5 Browsing a single configuration

In the configuration browser, the user will be met with all the inner most objects of the JSON file, see Figure 4.3.8.
The user can switch between the configuration browser and the Tree view. Its also possible to view the schema. In the left pane, one can see a list of all the other configurations for the template. It's possible to click any one of these to quickly switch configuration. The title of these are also highlighted as red if the

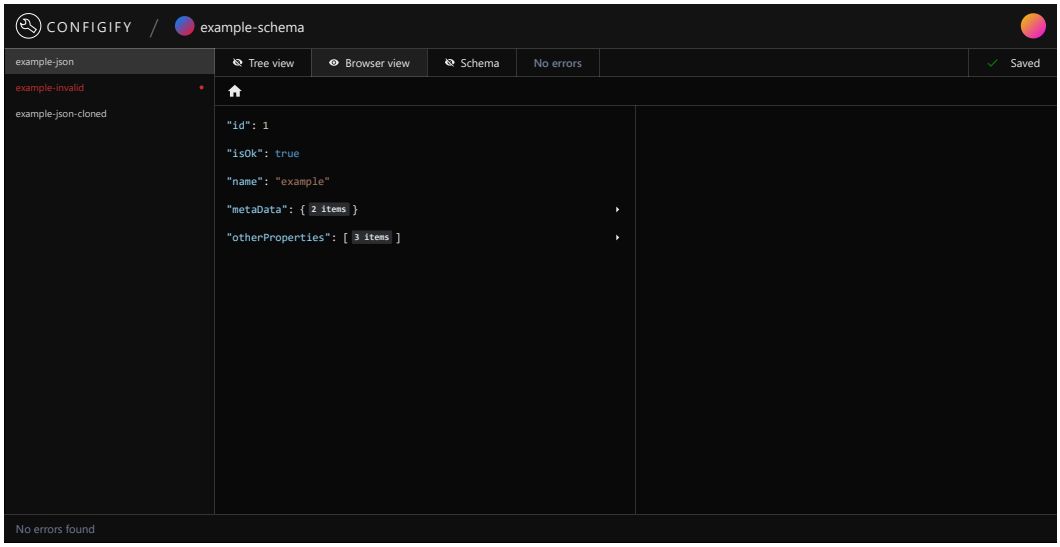


Figure 4.3.8: Illustration of the Configuration Browser Interface

configuration is marked as invalid. This allows the user to quickly navigate through all the configurations to fix any error.

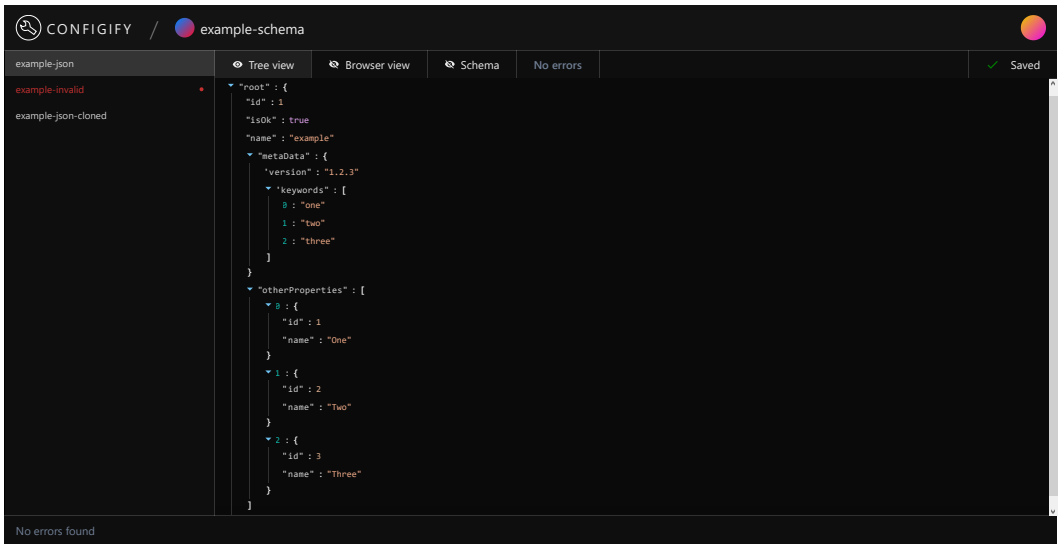


Figure 4.3.9: Illustration of the Configuration Browser's Tree view

4.3.5.1 Descending the object tree

The user can explore the object tree of the JSON file by clicking on any of the fields that contain objects. This action will display the content of that object in the right pane. As JSON objects can contain unlimited nested objects, clicking an object in the right pane that contains more objects will hide the original main pane, and the right pane will be shifted to the left to display the new content.

4.3.5.2 Editing values

In order to modify or insert values, the user must switch to the Tree view. To make changes to a value, the user will need to hover over the object that they wish

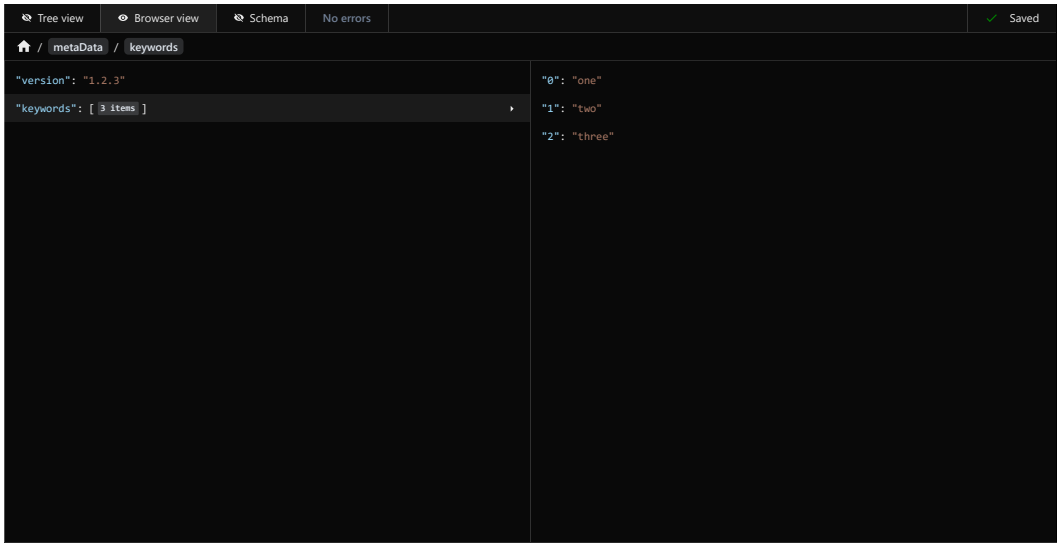


Figure 4.3.10: Screenshot of the Configuration Browser navigating through a JSON file

to modify. Two buttons will appear, one for editing the field and one for deleting it. Clicking the edit button will display a box containing the existing data. As the user enters new data, they can choose to save it as a string or, if the system detects that it can be inferred as a different data type, a different save button will appear underneath.

Example 1:
If the user types '12' in the box, one can either save it as a string, or infer it as a number.

Example 2:
If ones writes '{"id": 1, "name": "table"}', the system is able to infer that this text represents an object. It then offers a button to convert the text into an object. See Figure 4.3.11 and Figure 4.3.12 for a demonstration.

To add values, one needs hover the mouse over the desired object. A plus-sign will then appear, clicking this allows the user to choose a name for the value. Once it's created, one is able to edit the data as mentioned above.

4.3.5.3 Jumping to errors

In the case of configuration errors, a line at the bottom of the screen will display the total number of errors. This line is interactive and can be expanded to show each error individually. Moreover, if the user is in the browser view, one can simply click on each error to navigate to the location of the error automatically.

4.3.6 Account management

Account management is a crucial aspect of our project. We have developed a comprehensive account management system that enables users to perform essential

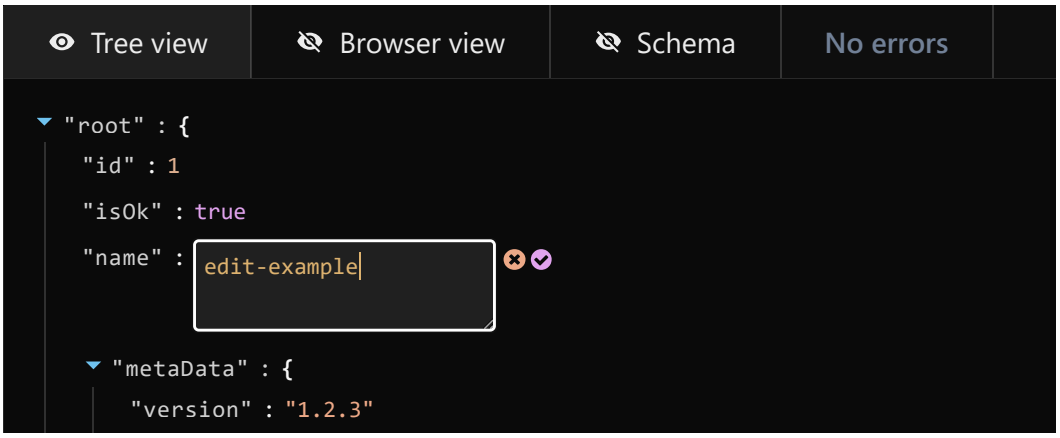


Figure 4.3.11: A figure showing the Configuration Browser's editing functionality in the Tree view

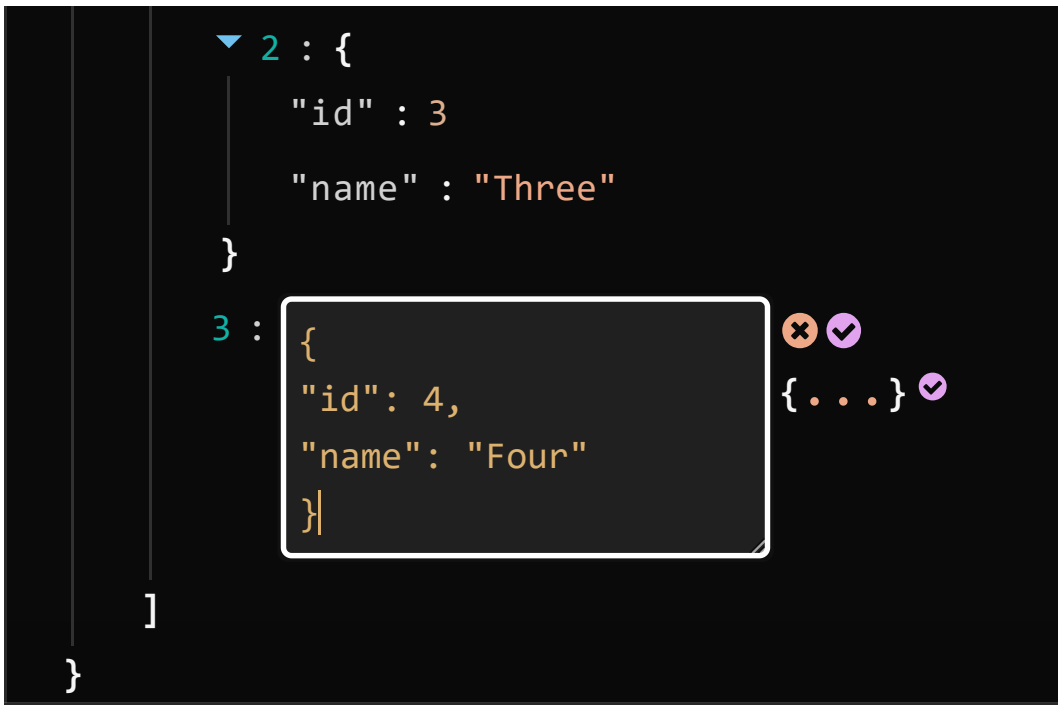


Figure 4.3.12: A figure showing how a user can add a complex object to the configuration

functions such as creating, modifying, and deleting their accounts as required.

4.3.6.1 Deletion

To delete their account, the user must navigate to the account page and click the delete button. A confirmation dialog appears, that once confirmed, makes the server removes all user data associated with the account from the database, and the user is logged out. Figure 4.3.14

We have taken appropriate measures to ensure that the deletion process is irreversible. This means that once a user confirms their request to delete their account, all associated data is permanently removed from our system. This ap-

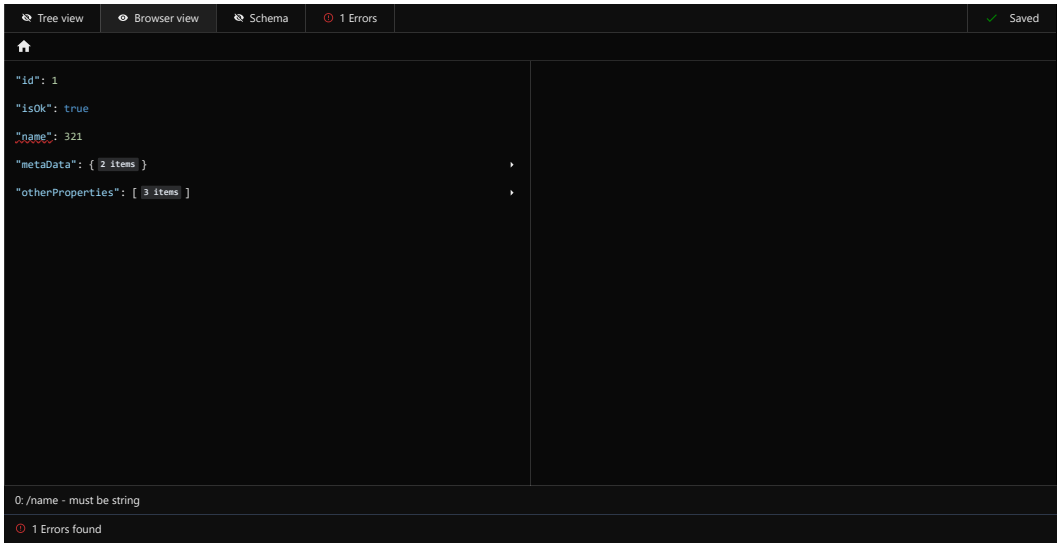


Figure 4.3.13: A figure demonstrating the Configuration Browser's error handling functionality, with clickable errors leading to their location in the JSON configuration

proach serves to safeguard user privacy and prevent the retention of user data by our application.

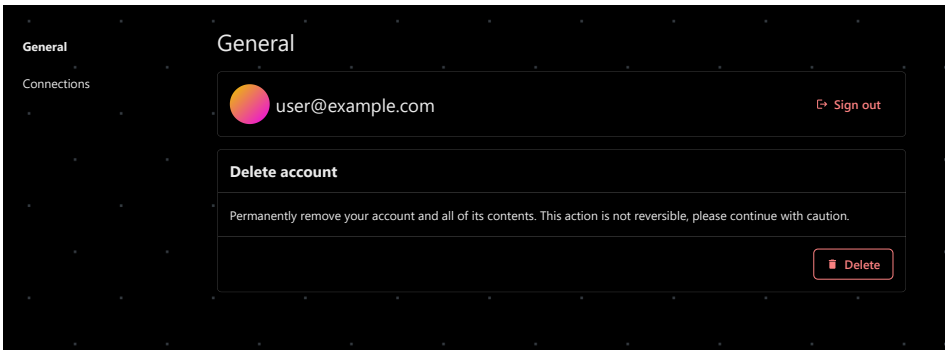


Figure 4.3.14: The account page showing the delete functionality

4.3.6.2 Connection

As one of the main requirements was the ability to link accounts, we have developed and integrated functionality that enables users to register or log in using their preferred identity provider and subsequently link their accounts that share the same email address.

Users can manage their account connections via the dedicated connections page, where they are able to view their current connections, establish new connections with additional identity providers, and remove existing connections as desired. The connections page showing an account that is linked to Google and email but not GitHub can be seen in Figure 4.3.15

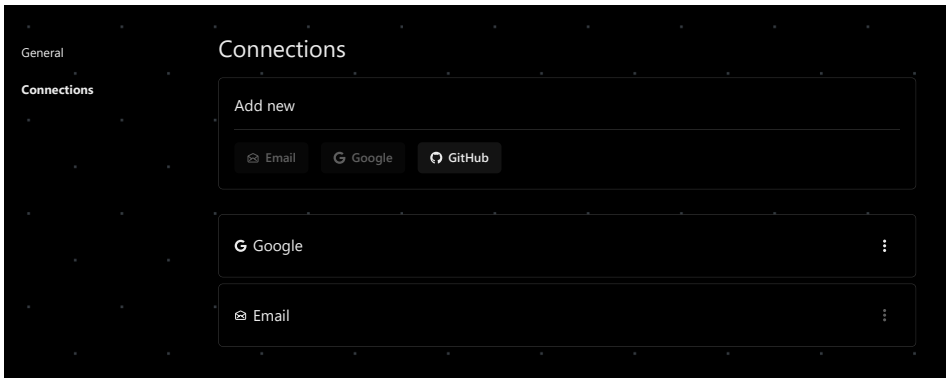


Figure 4.3.15: The connections page

4.3.7 Endpoints

4.3.7.1 Validate

This public endpoint takes in a JSON schema and a JSON as a string, then validates the JSON against the given schema as requested by the client. The validation process determines whether the JSON is valid or invalid based on the given schema. If the JSON is invalid, the endpoint will return an array of errors and paths to the errors.

4.3.8 Database

Our database schema was designed using the Prisma ORM. It's written as a Prisma schema and consists of five tables: Account, Session, User, Template, and Configuration. The schema was optimized for NextAuth and allowed for easy authentication and authorization. The configuration model was designed to store JSON templates and configurations, which could be edited by the user. The solution was effective and allowed for efficient database management and usage.

The complete database schema is presented in a figure in the appendix, as shown in Figure E.4. Additionally, a detailed description of the database design can be found in the database section of appendix E, System documentation.

4.4 Administrative results

4.4.1 Work in Jira

Jira was successfully utilized to keep track of the progress and backlog of the project. Jira provided a good system for splitting the project into smaller tasks, making it easy to get started working as you knew exactly what you were going to work on.

4.4.2 Sprints and planning

During the course of our project, we encountered some challenges with our sprints and planning. One issue was that our team did not estimate the time needed for

Jira issues or do planning poker like we planned. This made it difficult to accurately plan and allocate resources for each sprint.

Additionally, while the team members who were focused on coding were able to make progress, the member assigned to the scrum master duty rarely showed up to our daily work meetings and did not perform his assigned duty. This lack of participation from a key team member hindered our ability to effectively plan and execute our sprints. Although the rest of the team should have recognized and resolved this issue, it went unattended as their focus was consumed by tasks aligned with their individual responsibilities.

As a result of these challenges, we were unable to generate burn-down charts to track our progress. This made it difficult to assess our progress and adjust our plans accordingly.

4.4.3 MVP

During the development of our app, we found it important to prioritize the MVP functionality over the stretch goals. This allowed us to prioritize and deliver the most important features first, while ensuring that the product met the needs of our client.

CHAPTER

FIVE

DISCUSSION

5.1 Scientific discussion

5.1.1 Expectations VS Results

In order to evaluate the effectiveness of our solution and its impact on time efficiency, we conducted an experiment following the completion of the last sprint. The experiment involved three members of our team, each assigned specific roles. One member designed and set up the experiment, planting three deliberate errors within two JSON configuration files. The other two members were tasked with identifying and rectifying these errors, first using a basic text editor and then utilizing our web application.

Throughout the experiment, we carefully tracked the time taken by each team member to fix the configuration errors, as well as their overall success rate in resolving all issues. The results revealed a notable improvement in time efficiency when using our web application. The average time required to rectify a configuration error decreased from 156.50 seconds to 94.75 seconds, representing a significant $\approx$ 39% reduction in time spent. This can be seen in graph in Figure 5.1.1 and in Table 5.1.3.

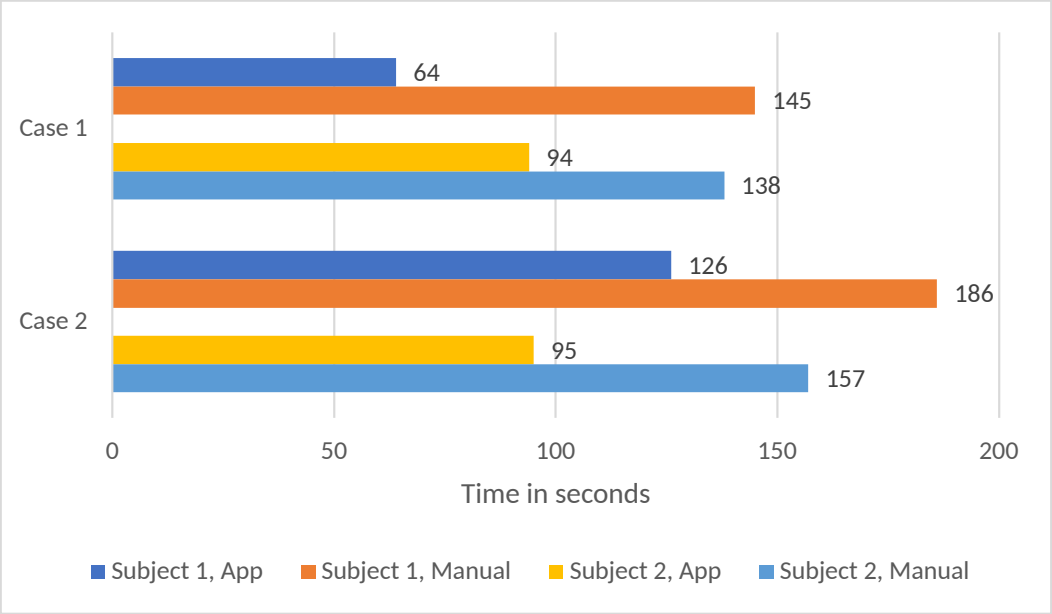


Figure 5.1.1: Comparison of results obtained through manual configuration fixes versus using our web application

Case 1

Subject 1	
Method	app
Time (s)	64
Errors	0
Method	manual
Time (s)	145
Errors	0

Subject 2	
Method	app
Time (s)	94
Errors	0
Method	manual
Time (s)	138
Errors	2

Case 2

Subject 1	
Method	app
Time (s)	126
Errors	0
Method	manual
Time (s)	186
Errors	1

Subject 2	
Method	app
Time (s)	95
Errors	0
Method	manual
Time (s)	157
Errors	0

Table 5.1.1: The raw data we collected during the experiment

Moreover, the participants provided valuable feedback regarding their experience with the experiment. They reported that manually identifying errors in the JSON schema and subsequently correcting the configurations using a text editor was challenging and confusing. Additionally, the absence of a validation check within the text editor made it difficult to ascertain whether the configuration was correctly fixed or not.

	Subject 1, App	Subject 1, Manual	Subject 2, App	Subject 2, Manual
Case 1	64	145	94	138
Case 2	126	186	95	157

Table 5.1.2: The data from the experiment formatted in a table

	Average time (s)
App	94.75
Manual	156.50
% difference	39.46

Table 5.1.3: The average time it took a test subject to correct a JSON file in each tool, time in seconds

These findings, although based on a small-scale experiment, demonstrate the potential for improvement in JSON configuration file management tools. The results underscore the importance of our web application's ability to streamline the error identification and resolution process, leading to enhanced time efficiency for users. Nonetheless, further research and exploration are necessary to fully explore and validate these initial findings.

5.2 Limitations and challenges

5.2.1 NextAuth challenges

We found that NextAuth is a powerful authentication library for Next.js that provides a seamless and flexible way to handle authentication and authorization in web applications. However, during our implementation of the library, we faced some issues that caused difficulties in achieving certain functionalities that were required by the client. One such issue was the inability to link accounts with different emails and the lack of automatic linking for accounts with the same email. We found that in order to enable account linking to the same email, we needed to use the allowDangerousEmailAccountLinking flag, which can be a security risk if not used properly. An example of this can be seen in Figure 5.2.1. There was however no solution for linking accounts with a different email addresses.

```
GoogleProvider({
    clientId: env.GOOGLE_CLIENT_ID,
    clientSecret: env.GOOGLE_CLIENT_SECRET,
    allowDangerousEmailAccountLinking: true,
}),
```

Figure 5.2.1: A code snippet demonstrating the configuration of the Google provider with the allowDangerousEmailAccountLinking property set to true

Another issue we faced was the lack of built-in support for unlinking accounts and refreshing the session token. While NextAuth does indeed provide a simple and easy to use API for handling authentication, we found that managing the state of linked accounts and refreshing the session after modifying an account required additional development and implementation effort. This might lead to potential security vulnerabilities in the future if not properly handled.

In light of these challenges and in consultation with the client, we ultimately decided to forgo these requirements.

5.2.2 Prisma problems

We found Prisma to be a great tool for our project, as it allowed us to focus on application logic rather than database operations. It simplifies database access and eliminates the need for raw SQL queries. One of the most significant advantages of using Prisma is its ability to perform database migrations easily. This makes it easy to keep the database schema in sync with the application codebase as it evolves over time and more fields and tables are added.

However, there have been some limitations with Prisma that we encountered during development. One of the biggest issues we faced while working with Prisma was its inability to use more than one database provider in a single project. In our project, we wanted to use SQLite for local development and testing, but we also needed to use PostgreSQL in production like the client requested. Unfortunately,

Prisma only supports a single database provider per project, which meant that we had to choose between using SQLite or PostgreSQL for our entire project.

This issue was particularly frustrating because we wanted to take advantage of the benefits of using SQLite for local development and testing, such as its lightweight nature and ease of use. However, we were forced to use PostgreSQL for the entire project, which was less than ideal. While we enjoyed working with Prisma and appreciated its many benefits, as Prisma continues to evolve and improve, we hope to see these issues addressed in future releases.

5.2.3 Deployment

In our project, we chose to use Vercel as our continuous integration and deployment solution. We connected our GitHub repository and set up a build and deployment process. Whenever we pushed new code to the main branch of the repository, Vercel would automatically build and deploy the changes to the live version of the application.

An advantage of using Vercel is that it provides a live preview of the deployed application for any branch. This meant that our client had access to the live version of the application during the entire duration of the project and this allowed them to provide feedback and make suggestions based on the actual application instead of just mockups or prototypes.

5.2.4 Requirements and Limitations

The requirements for this project were outlined in the project description, which is provided in appendix B, project description.
During the development of the app, we encountered several requirements and limitations that we had to address. These requirements were mainly based on the project description, but some of them had to be adjusted or dropped along the way due to technical limitations or time constraints.

5.2.4.1 Email and password authentication

One of the first requirements that we had to drop was the email and password login. We initially planned to have users log in with their email and password like the client requested, but due to the limitations of NextAuth, [46] we had to drop this feature in favor of a passwordless email login. While this may have caused some inconvenience for users, we believe that it was the best solution given the circumstances and it helps improve our security since no passwords are stored.

5.2.4.2 Download from a URL

Another requirement that we had to drop was the ability to download with a URL. Originally, we planned to allow users to download a JSON file by inputting a URL. However, we discovered that tRPC, the library we used for server-client communication, did not support binary file transfer, making it impossible to download a

file in this way. Support for this isn't planned in tRPC, and as a result, we had to drop this requirement. [47]

5.2.4.3 Users can't run the validation with a button

A third requirement that we had to drop was the ability for users to manually run the validation with a button. Initially, we planned to allow users to run the validation process manually with a button. However, the client requested us to remove this feature due to concerns about wasting compute time. Instead, we made the decision to only run the validation process when the configuration was updated, ensuring that the validation process was only run when necessary.

5.2.4.4 Editing fields inside the editor

We were unable to implement the ability for users to edit fields inside the configuration editor. While this was a feature that we initially wanted to include, we found that the way we had designed the editor prevented us from implementing this in a good way. Due to time constraints, we had to limit editing to the tree view, which is provided by an external library.

5.3 Communication

5.3.1 Client

Communication with the client was a crucial part of our development process. We held meetings with the client at the end of every sprint to discuss the progress made, receive feedback and to plan for the next sprint. Prior to each meeting we were expected to send a meeting notice along with a sprint report. However, there were instances were the report was sent out a little late due to us either forgetting, or us wanting finish more of the tasks before sending the status. Aside from this, the communication itself went really well. The feedback was a great tool in order to keep the focus on the correct aspects of the project.

5.3.2 Internal

During the course of the project, it became clear that the communication within the group was not as effective as it could have been. In the beginning, rather than discussing tasks and plans as a team, some members tended to work on tasks independently without adequate discussion with the rest of the group. This led to a lack of clarity and understanding of what other members were working on and what the overall progress of the project was. But we had some discussions within the group and made sure that everything would be documented, so this problem was solved pretty early.

Additionally, there were several instances where the group planned to meet up and work together at school, but it did not go as smoothly as they could have. There were times when some members did not respond or were late, which caused frustration and delays. But a lot of these problems occurred due to bad planning,

as we mostly decided on which days to work the day before. If we had put down a proper plan for the different weeks, this is a problem that could have been easily mitigated.

Another area of concern was that when disagreements arose, the conversations sometimes became heated and unprofessional. This led to tension within the group and negatively impacted the morale of the team.

5.3.3 How we worked

Creating tasks in Jira was a crucial aspect of our work process as it provided a centralized platform for task management. The intended purpose was to allow each team member to choose tasks from a backlog whenever they completed their current assignments, ensuring a continuous workflow. However, this method did not work as effectively as planned as not all team members were equally proactive in picking up new tasks. As a result, some team members ended up with additional workloads, leading to an uneven distribution of work within the group.

To address this issue, we re-evaluated the workload and tried to delegate tasks more evenly among group members, based on their individual strengths and areas of expertise. The resulting workload was still a bit unbalanced, but it helped to uneven some of the problem towards the end of the project.

To ensure effective communication with the client and supervisor, we made plans to hold meetings at the end of each sprint, which required all team members' presence. We also scheduled meetings with our supervisor to discuss various aspects of the report.

As mentioned earlier, we found that a lot of tasks were being completed without proper planning or discussion within the group. One notable example was the implementation of the database structure, which was not properly discussed in the group before being implemented. However, we quickly recognized this issue and took steps to improve our internal work processes.

5.3.4 The plans we made

Our planning process lacked structure and specificity. Although we had a preparation plan that outlined certain activities, it did not include a schedule or clearly designate who was responsible for each task. This lack of specificity may have contributed to a lack of motivation and productivity experienced by some team members in the initial stages of the project.

We decided to meet up on campus almost every day to work together as a team. Being in the same physical space meant that we could easily communicate and collaborate on tasks, ask each other for help, and have in-depth discussions about project details. Despite some members regularly not showing up for these ses-

sions, it still helped the rest of us work more effectively, as we were able to quickly address issues or concerns that arose during the development process. This also allowed us to give each other suggestions on how to solve various problems, if anyone got stuck with a task.

We also planned bi-weekly meetings with the client. Every member of the team were supposed to be present for these meetings, but unfortunately not all members showed up. Unfortunately, there was one member who failed to show up for these meetings on multiple occasions.

5.3.5 Agile methodology

In our project, we adopted the agile work methodology, which involved dividing the development process into smaller sprints that typically lasted around 2 weeks. After each sprint, we held a meeting with the client to gather feedback and prepare for the next sprint.

To streamline the planning process, we worked collaboratively with the client to establish a list of essential requirements and various stretch goals. We then prioritized these items based on their importance, which allowed us to determine what to include in each sprint and what to focus on. The team really felt that this approach helped us plan our work more effectively and ensured that we were making progress towards our goals.

5.3.6 Collaborations with Jira and Confluence

We utilized Jira and Confluence for collaboration purposes. Jira was used to plan our sprints, delegate tasks and report our hours. Although this system worked well, there were some challenges with the time-logging process, as hours needed to be properly logged in order to start the next sprint. Unfortunately, this process was not always done in a timely manner. Also, we had never used Jira before, which meant that there were some functions that we didn't utilize to their fullest potential.

In addition to Jira, we also made use of Confluence to store our product requirements, meeting notes, and decision logs. However, in retrospect, we probably could have used Confluence more proactively than we did. It is a very powerful tool, but we didn't fully leverage its capabilities, which may have hindered our collaborative efforts.

5.3.7 Pull requests

To improve our internal work processes, we also utilized branching and pull requests when needed. This helped us review and evaluate code changes before merging them into the main branch, enabling us to identify and fix any mistakes or errors before they were pushed to production.

CHAPTER

SIX

CONCLUSION AND FURTHER WORK

6.1 Conclusion

The goal of this project was to create a full-stack web application for managing the configuration of distributed software environments. We aimed to provide a user-friendly interface that minimizes potential misconfiguration and human errors.

To meet the project requirements, we designed the application as a containerized solution that can be deployed on Kubernetes. We used React with Next.js as the framework for the front-end, Prisma as the ORM solution, and Chakra UI as the component library. Our back-end was built with tRPC, providing end-to-end type-safe API code. Docker was used for containerization.

Despite successfully meeting our minimum requirements, we had to make certain compromises due to time constraints. As a result, we were unable to fully develop some of our envisioned features for the configuration browser. However, we are overall satisfied with the results achieved in this project. Given that we were unable to find any existing solutions for a problem like this, we are pleased with what we have accomplished.

6.2 Limitations and future work

6.2.1 Prisma cold start time

Serverless computing platforms, such as Vercel, have the benefit of scaling automatically based on traffic and usage, but they also have the limitation of experiencing a cold start when a new request comes in. This means that the server needs to initialize and allocate resources before it can start processing the request, which can add a delay to the response time. It's worth noting that once the server is initialized, it can process subsequent requests more quickly without incurring the same initialization delay. This being a serverless platform, the server has a set lifetime before it gets despawned, which allows it to handle multiple requests

within that time frame without incurring the cold start delay again.

In our project, we use Prisma as our ORM to interact with the database. Prisma currently uses a protocol based on GraphQL, which adds a considerable CPU and memory overhead that can further slow down the cold start time. To improve this, the Prisma team has recently created a new JSON-based wire protocol that eliminates GraphQL and can improve startup times by up to 9 times on large schemas.

This new protocol is more efficient mainly because it no longer requires the DMMF (Data Model Meta Format) to serialize messages, the Rust runtime that was used for running GraphQL was dropped, among other optimizations. It is worth noting that the protocol itself is not just JSON but rather a JSON-based wire protocol. [48]

This feature is still in preview and won't be ready for production until after the completion of this thesis. However, we were able to test the performance benefits of this new protocol. We took five measurements of the cold start time, plotted them and calculated the average start time. We found that this new protocol improves the start times by around $\approx 62\%$ in our case, reducing the cold start wait time from around 8 seconds to roughly 3. The results that are included in Table 6.2.1 and the mean time in Figure 6.2.1, show a significant improvement in cold start performance after the introduction of the new protocol.

	Before		After	
	8.84	s	2.58	s
	7.95	s	3.02	s
	7.82	s	2.30	s
	8.31	s	3.94	s
	8.41	s	3.92	s
Average	8.27	s	3.15	s

Table 6.2.1: Comparison of Prisma serverless cold start times before and after the introduction of the new JSON-based wire protocol. The start times of the five measurements are shown in seconds.

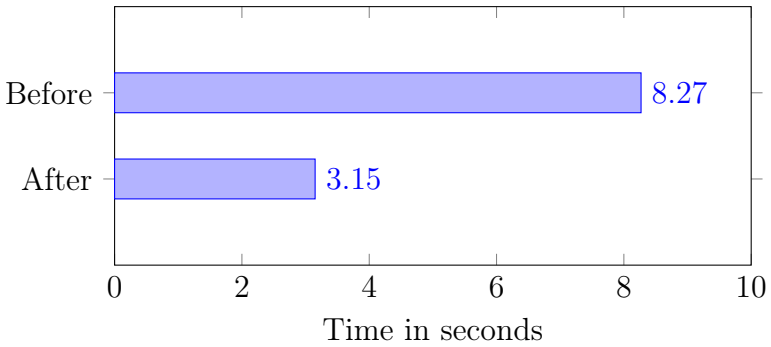


Figure 6.2.1: The comparison of mean start times before and after the introduction of the JSON protocol

6.2.2 Upgrading Next.js

Upgrading to a newer Next.js version would bring several improvements to our application. This latest version brings a range of enhancements that can significantly benefit our project. One notable improvement is the update to React 18+, which introduces advancements in concurrent rendering, streaming from server components, and automatic batching of state changes, among other out-of-the-box enhancements. [49] This upgrade would allow us to leverage the latest features and optimizations provided by React, leading to improved performance and a smoother user experience.

Next.js 13.4 also stabilizes the App Directory feature, which brings substantial improvements to the file system-based router and facilitates the creation of complex layouts with advanced routing patterns. [50] This feature provides a more streamlined and efficient approach to managing routes, enhancing the overall architecture of our application.

Additionally, Next.js 13 introduces Turbopack, a new Rust-based Webpack replacement. According to the creators of Vercel, Turbopack can deliver a performance boost of up to 700x compared to traditional Webpack. [51] This upgrade would result in improved image optimization, enhanced font management, and a more efficient link component, further optimizing the loading and rendering speed of our application.

CHAPTER
SEVEN

SOCIETAL IMPACT

7.1 Economic effects

The implementation of our web app has the potential to yield significant economic benefits for organizations. The application allows employees to manage JSON configurations with way more efficiency compared to doing it manually with existing tools, which means that they can get more work done in less time. In addition to that, misconfigurations and human errors can lead to product failures and customer dissatisfaction which can be costly for the company. With our solution for validating the configuration files before they are deployed, the risk of such mistakes is greatly reduced.

7.1.1 Open-source

The project will be open source and available to everyone. This allows for the use, distribution, and development of commercial software utilizing the project's solutions. By making the project freely available, it will be a contribution to the open-source community by providing example code and documentation that can be utilized and learned from by other developers. Since the solution is free to use, it gives small and medium sized businesses who don't have the same amount of resources as a large company a better chance to keep up with the industry without having to worry about resource limitations.

7.2 Relation to UN's sustainable development goals

Our solution can contribute to SDG 8: Decent Work and Economic Growth, by enabling organizations to increase the efficiency of their employees. When it comes to the topic of decent work, our solution can help improve morale and satisfaction for employees by making a task that is boring and frustrating easier and more bearable, leaving the employees with more energy and motivation. [52]

REFERENCES

[1] *Monolithic Architectures in Software Development*. Agile Academy. URL: https://www.agile-academy.com/en/agile-dictionary/monolithic-architecture/ (visited on 04/26/2023).

[2] *Microservices vs. monolithic architecture*. Atlassian. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (visited on 04/26/2023).

[3] Gustavo André Setti Cassel et al. "Serverless computing for Internet of Things: A systematic literature review". In: *Future Generation Computer Systems* 128 (2022), pp. 299–316. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2021.10.020. URL: https://www.sciencedirect.com/science/article/pii/S0167739X21004167 (visited on 05/03/2023).

[4] *10 Best Front end Frameworks for Web Development in 2023*. Monocubed. URL: https://www.monocubed.com/blog/best-front-end-frameworks/ (visited on 05/15/2023).

[5] Eugene Xiong. *Cloud-Based APIs And The Future Of Business Applications*. URL: https://www.forbes.com/sites/forbestechcouncil/2022/01/11/cloud-based-apis-and-the-future-of-business-applications/ (visited on 05/12/2023).

[6] *What is JSON*. W3Schools. 2022. URL: https://www.w3schools.com/whatis/whatis_json.asp (visited on 05/11/2023).

[7] *The JSON Data Interchange Format*. ECMAScript. 2013. URL: https://www.ecma-international.org/wp-content/uploads/ECMA-404_1st_edition_october_2013.pdf (visited on 05/11/2023).

[8] Karisma Kunwar. *The Reason why JSON is so Popular*. June 2021. URL: https://www.prokurainnovations.com/json/ (visited on 05/18/2023).

[9] *Understanding JSON Schema*. JSON Schema. 2021. URL: https://json-schema.org/understanding-json-schema/index.html (visited on 04/03/2023).

[10] Priya Pedamkar. *Advantages of OOP*. URL: https://www.educba.com/advantages-of-oop/ (visited on 05/18/2023).

[11] *Intro to Functional Programming Basics*. freeCodeCamp. 2020. URL: https://www.freecodecamp.org/news/intro-to-functional-programming-basics/ (visited on 05/16/2023).

[12] *Built-in React Hooks*. React. 2023. URL: https://react.dev/reference/react (visited on 05/20/2022).

[13] *Reusing Logic with Custom Hooks*. React. 2023. URL: https://react.dev/learn/reusing-logic-with-custom-hooks (visited on 05/20/2022).

[14] *Relational database*. Wikipedia. 2022. URL: https://en.wikipedia.org/wiki/Relational_database (visited on 05/02/2023).

[15] *Relational databases*. IBM. URL: https://www.ibm.com/topics/relational-databases (visited on 04/10/2023).

[16] *Postgres vs. MongoDB for Storing JSON Data: Which Should You Choose?* Sisense. URL: https://community.sisense.com/t5/knowledge/postgres-vs-mongodb-for-storing-json-data-which-should-you/ta-p/111 (visited on 04/12/2023).

[17] *What Is NoSQL? NoSQL Databases Explained*. MongoDB. URL: https://www.mongodb.com/nosql-explained (visited on 04/12/2023).

[18] *What is a Container?* Docker. 2023. URL: https://www.docker.com/resources/what-container/ (visited on 04/18/2023).

[19] *Docker architecture overview*. Docker. 2023. URL: https://docs.docker.com/get-started/overview/#docker-architecture (visited on 04/18/2023).

[20] *JWT Authentication and Authorization: A Detailed Introduction*. DZone. URL: https://dzone.com/articles/jwt-authentication-and-authorization-a-detailed-introduction (visited on 04/14/2023).

[21] *JWT Authentication And Authorization*. C# Corner. URL: https://www.c-sharpcorner.com/blogs/introduction-and-detail-about-the-jwt-authentication-and-authorization (visited on 05/18/2023).

[22] Flavio Copes. *JWT authentication: Best practices and when to use it*. URL: https://blog.logrocket.com/jwt-authentication-best-practices/ (visited on 04/10/2023).

[23] *What Is Token-Based Authentication?* Okta. 2023. URL: https://www.okta.com/identity-101/what-is-token-based-authentication/ (visited on 05/08/2023).

[24] *Agile 101*. Agile Alliance. URL: https://www.agilealliance.org/agile101/ (visited on 04/10/2023).

[25] *Scrum Overview*. Mountain Goat Software. URL: https://www.mountaingoatsoftware.com/agile/scrum/resources/overview (visited on 04/10/2023).

[26] *What is Scrum Software Development?* Stackify. URL: https://stackify.com/what-is-scrum/ (visited on 05/18/2023).

[27] Dan Radigan. *What is a Code Review & How It Can Save Time*. URL: https://www.atlassian.com/agile/software-development/code-reviews (visited on 05/04/2023).

[28] Simona Galant. *Code Review Checklist: A Guide to Your First Peer Code Review*. Feb. 2021. URL: https://www.springboard.com/blog/software-engineering/code-review-checklist-for-2021/ (visited on 05/04/2023).

[29] *Next.js Docs*. Vercel. URL: https://nextjs.org/docs (visited on 05/18/2023).

[30] Ejiro Asiuwhu. *NextAuth.js for client-side authentication in Next.js*. Mar. 2022. URL: https://blog.logrocket.com/nextauth-js-for-next-js-client-side-authentication/ (visited on 04/11/2023).

[31] *Chakra UI - A simple, modular and accessible component library that gives you the building blocks you need to build your React applications.* ChakraUI. URL: https://chakra-ui.com/ (visited on 04/27/2023).

[32] Michal Kuncio. *Type safe APIs with Nuxt 3 and tRPC*. 2022. URL: https://dev.to/michalkuncio/type-safe-apis-with-nuxt-3-and-trpc-202g (visited on 04/12/2023).

[33] Chidiebere Onyejekwe. *TANStack Query: How It Changes the Way You Query APIs*. URL: https://dev.to/codewithonye/tanstack-query-how-it-changes-the-way-you-query-apis-5fog (visited on 04/07/2023).

[34] *Hassle-free Database Migrations with Prisma Migrate*. Prisma. 2022. URL: https://www.prisma.io/migrate (visited on 04/14/2023).

[35] *Cloud Hosting for Developers*. Render. 2022. URL: https://render.com/pricing#postgresql (visited on 04/16/2023).

[36] *Store JSON documents in SQL Server or SQL Database*. Microsoft. 2023. URL: https://learn.microsoft.com/en-us/sql/relational-databases/json/store-json-documents-in-sql-tables?view=sql-server-ver16 (visited on 04/27/2023).

[37] *Ajv JSON schema validator*. Ajv. 2022. URL: https://ajv.js.org/ (visited on 04/17/2023).

[38] *Serverless Functions Overview*. Vercel. URL: https://vercel.com/docs/concepts/functions/serverless-functions (visited on 04/19/2023).

[39] *Edge Network Overview*. Vercel. URL: https://vercel.com/docs/concepts/edge-network/overview (visited on 05/12/2023).

[40] *Preview Deployments Overview*. Vercel. URL: https://vercel.com/docs/concepts/deployments/preview-deployments (visited on 05/11/2023).

[41] madhur912. *Why should you use Docker? 7 Major Reasons*. GeeksforGeeks. URL: https://www.geeksforgeeks.org/why-should-you-use-docker-7-major-reasons/ (visited on 04/16/2023).

[42] *Project Management Made Easy with 8 Features of Jira*. Ricksoft Inc. 2023. URL: https://www.ricksoft-inc.com/post/project-management-easy-8-features-of-jira/ (visited on 05/06/2023).

[43] *Next Features: ESLint*. Vercel. 2022. URL: https://nextjs.org/docs/basic-features/eslint (visited on 04/11/2023).

[44] Srinivas Kulkarni. *Jaro Winkler vs Levenshtein Distance*. Mar. 2021. URL: https://srinivas-kulkarni.medium.com/jaro-winkler-vs-levenshtein-distance-2eab21832fd6 (visited on 05/09/2023).

[45] Martin Ankerl. *How to Create Random Colors Programmatically*. 2009. URL: https://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/ (visited on 02/07/2023).

[46] *NextAuth FAQ*. Vercel. URL: https://next-auth.js.org/faq#compatibility (visited on 04/03/2023).

[47] lkj4. *v10 tRPC client API Design*. July 2022. URL: https://github.com/trpc/trpc/discussions/2270#discussioncomment-3217169 (visited on 04/13/2023).

[48] Sabin Adams, Jan Piotrowski, and Nikolas Burk. *How We Sped Up Serverless Cold Starts with Prisma by 9x*. Apr. 2023. URL: https://www.prisma.io/blog/prisma-and-serverless-73hbgKnZ6t (visited on 05/01/2023).

[49] *React v18.0*. React. URL: https://react.dev/blog/2022/03/29/react-v18 (visited on 05/21/2023).

[50] *Blog - Next.js 13*. Vercel. URL: https://nextjs.org/blog/next-13 (visited on 05/21/2023).

[51] *Advanced Features: Turbopack*. Vercel. URL: https://nextjs.org/docs/advanced-features/turbopack (visited on 05/21/2023).

[52] *Economic Growth - United Nations Sustainable Development*. United Nations. 2023. URL: https://www.un.org/sustainabledevelopment/economic-growth/ (visited on 04/19/2023).

[53] *tRPC - Move Fast and Break Nothing*. tRPC. URL: https://trpc.io/ (visited on 05/04/2023).

[54] *Subscriptions | tRPC*. tRPC. URL: https://trpc.io/docs/subscriptions (visited on 05/04/2023).

[55] Christopher Ehrlich. *tRPC & Next Data Flow Diagram*. URL: https://www.c-ehrlich.dev/img/youtube/trpc-data-flows.png (visited on 04/15/2023).

APPENDICES

A - TRPC

tRPC is a client-server framework for building scalable web applications. It allows developers to define procedures on the server side and call them from the client side using a strongly-typed API. [53] tRPC uses HTTP requests and responses to send and receive data between the client and server. It uses a three-tier architecture, where the client, server, and data flow are separated into these three layers.

Front-end
Here, tRPC works by defining procedures that the client can call. These procedures are defined in a schema file, which describes the input and output types of each procedure. The client then uses a generated API to call these procedures.

Middleware
The middleware part of tRPC refers to the code that sits between the frontend and the backend. tRPC provides an Express middleware that intercepts requests and responses and can perform various operations such as authentication, logging, error handling, rate limiting, and caching. The middleware is designed to be modular, so multiple middlewares can be chained together to form a pipeline.

Back-end
The back-end of tRPC is the actual server that receives and processes the requests from the frontend. It typically consists of a collection of microservices or functions that perform specific tasks and return the necessary data to the frontend.

tRPC is designed to be flexible and can be used with any full-stack technology. It also includes built-in support for TypeScript and GraphQL. There's also support for subscriptions via Websockets. [54]

A1 - tRPC data flow

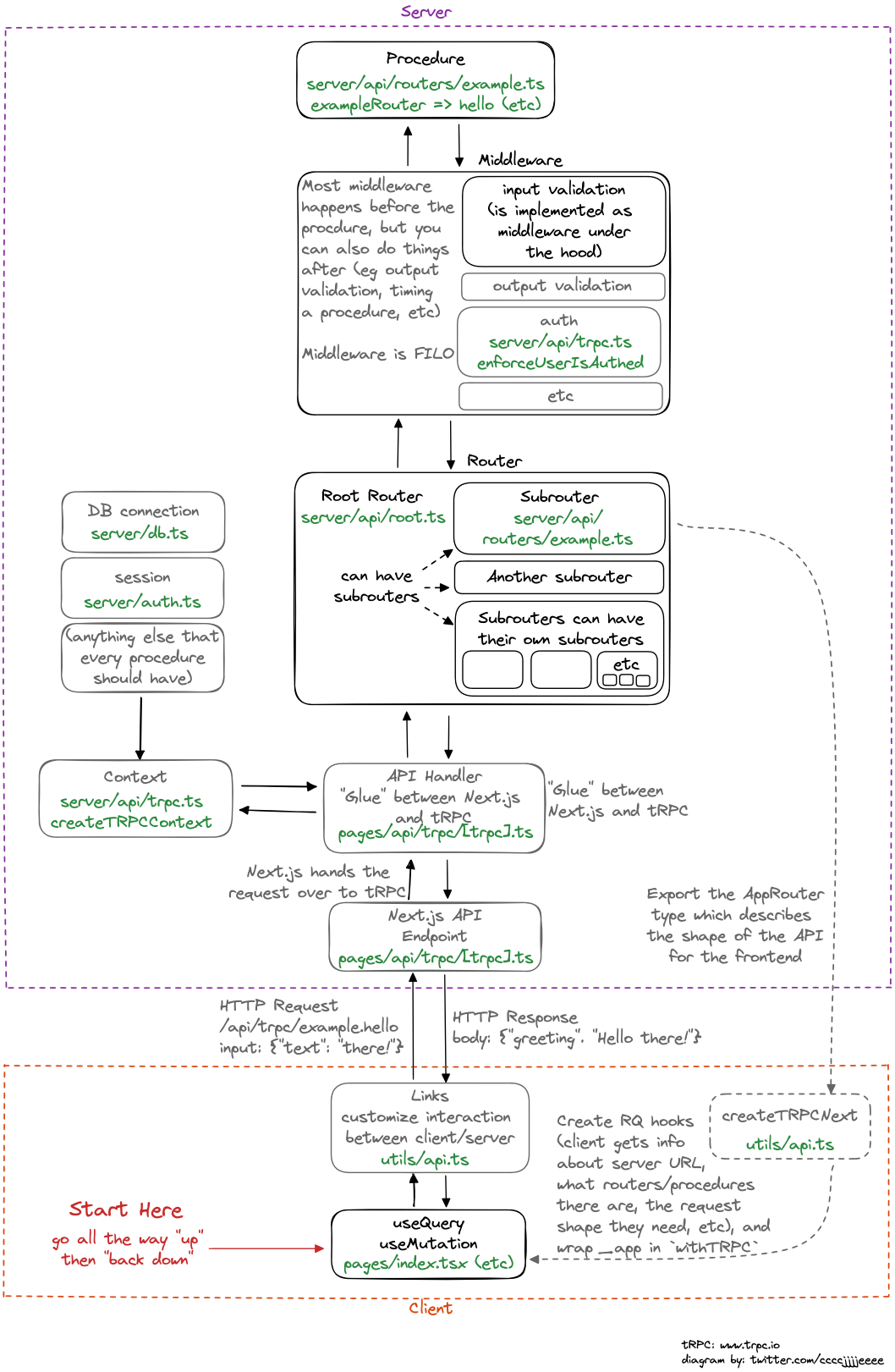


Figure A.1: Figure showing the data flow in tRPC [55]

B - PROJECT DESCRIPTION

This appendix includes the original project description, as presented in Figure B.1, which we received during the application process for various Bachelor projects. Additionally, it contains the revised project requirements, depicted in Figure B.2, which were developed following our first meeting with FiiZK where we clarified the goals.

B1 - Project description

Managing Complex Cloud Operations in a Distributed Software Environment

In a land of microservices and multi-tenant applications developers move more of day-to-day operations into the cloud to take advantage of scalability, on demand cloud operations, and security to mention a few. This brings some flexibility when it comes to selecting the resources to hand out jobs, since different customers and processes have different needs, some may have on premise hardware to respect internal security concerns, others may hire on-demand cloud resources to scale with workload. Either way, to comply with customer demands and needs we must be able to secure the data foundation for services we provide for the customer.

Create a fullstack web application for creating, maintaining, and validating the configuration used in our distributed data pipelines. The application should provide a user-friendly interface to minimize potential misconfiguration and human errors.
The students will get a first person view of how the Aqua Culture industry innovates with digital tools.

The solution should be built as a containerized application that can be deployed to our existing architecture in Kubernetes. The students can decide how to design the application architecture as long as the technology is compliant with containerized hosting. We can provide the best guidance if the students align with our technology that are .NET 6, Angular, MSSQL, Docker, and Kubernetes, but no prior knowledge of the technologies is required.

For more information about FiiZK and FiiZK Digital, visit https://fiizk.com/

Marius Lundbø

FiiZK Digital AS

marius.lundbo@fiizk.com

Figure B.1: The original project description

MVP and stretch goals

Main goals

- Log in/sign up
    1. The user should be able to sign in with at least Google
    2. The user should be able to sign in with email and password
        a. Verify email
- Import schema
    1. The user should be able to upload a JSON schema
    2. Schema will create a valid template
- Configuration
    1. The user should be able to upload JSON configuration files
    2. The user should be able to validate an existing JSON file
    3. The user should be able to download a configuration in JSON format
    4. The user should be able to modify an existing config
- Configuration editor
    1. The user should be able to browse a config
    2. The user should be able to edit fields
    3. The user should be able to run the validation
    4. The user should be able to save the config
    5. The user should be able to add new fields
- The app should be able to run inside Docker

Stretch goals

- Other Providers are optional (GitHub, Microsoft for ex.)
- The accounts from the social Providers with the same email should be merged
- Error verification for importing schema, show what's wrong
- Error messages when validating configuration, maybe details on what's wrong
- Add a search bar
- Keyboard navigation for power users
- Create a publicly accessible main page with a list of all features
- Transition animation when navigating the configurations
- Maybe Cypress testing on the configurator
- Create a landing page

Figure B.2: The MVP and stretch goals

C - WIREFRAMES

These wireframes are a visual representations of the user interface design for our web application. They show the layout of each page, including the placement of buttons, forms, and other interactive elements.

C1 - Wireframes of the web application

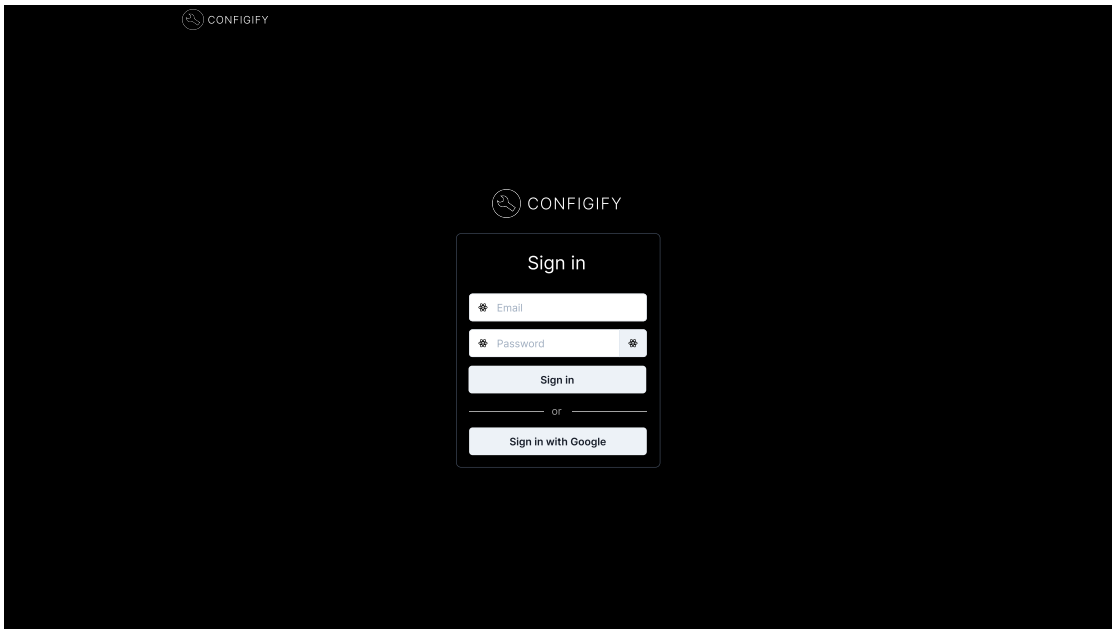


Figure C.1: A figure showing the sign in page

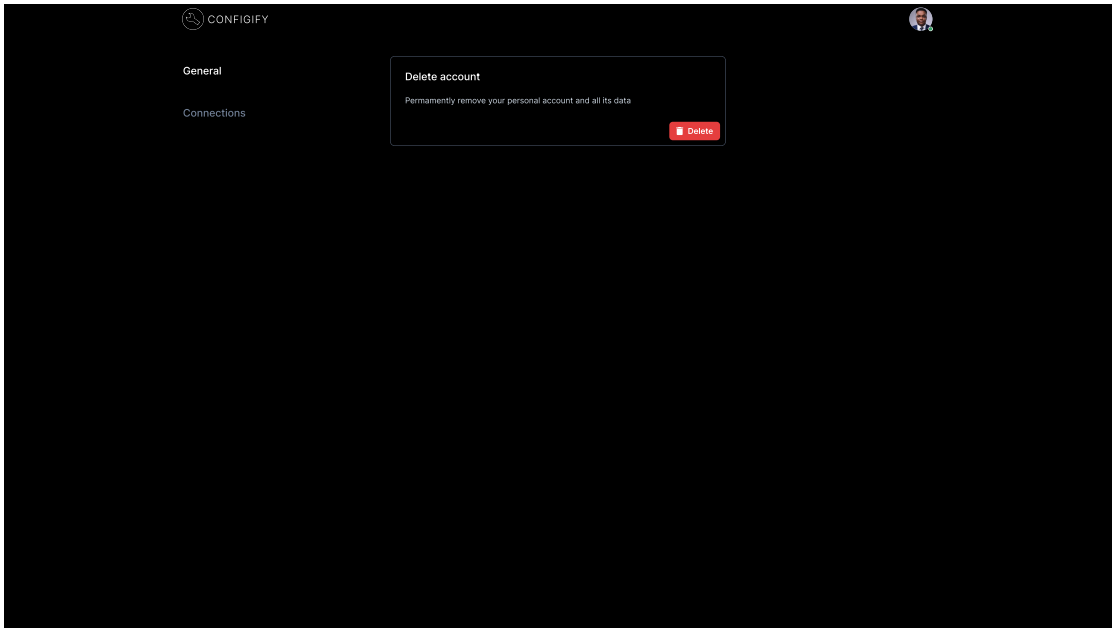


Figure C.2: A figure showing account page

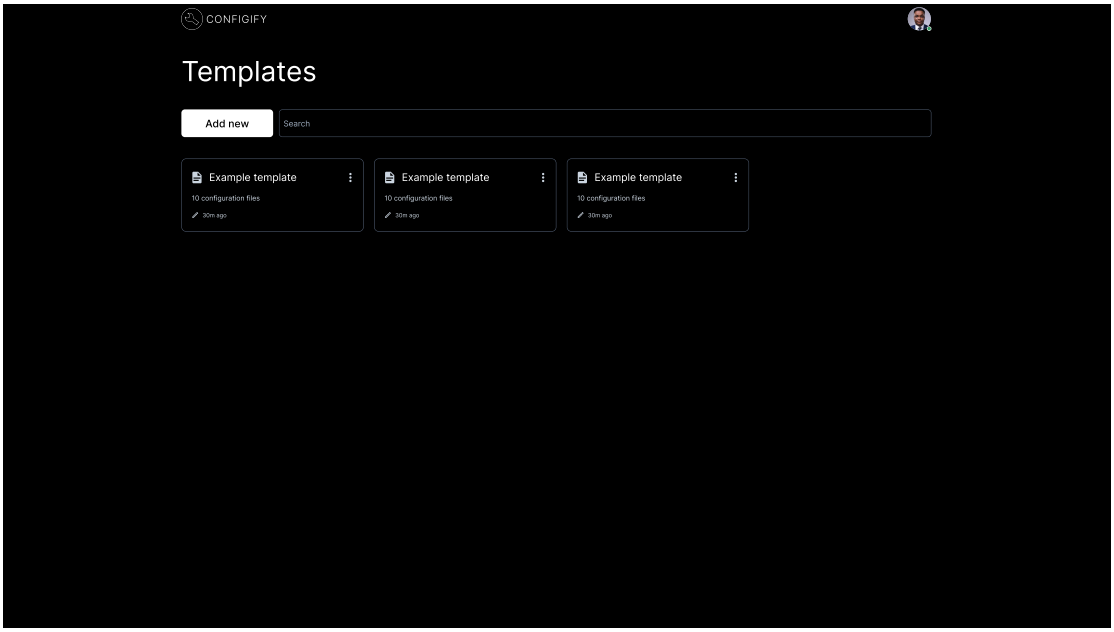


Figure C.3: A figure showing the templates page

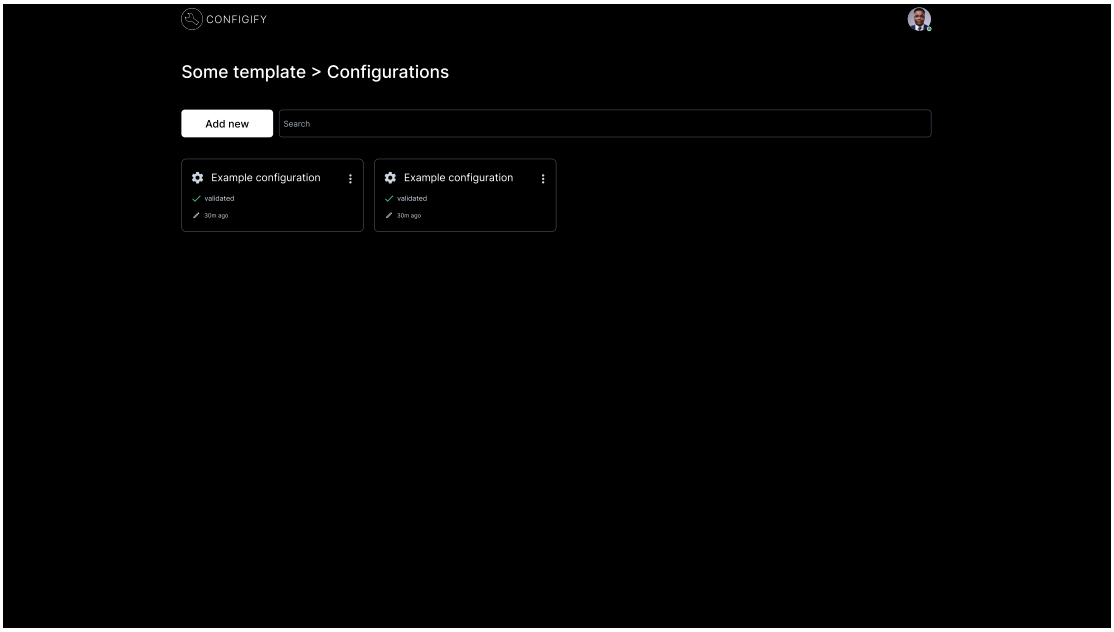


Figure C.4: A figure showing the configurations for a specific template

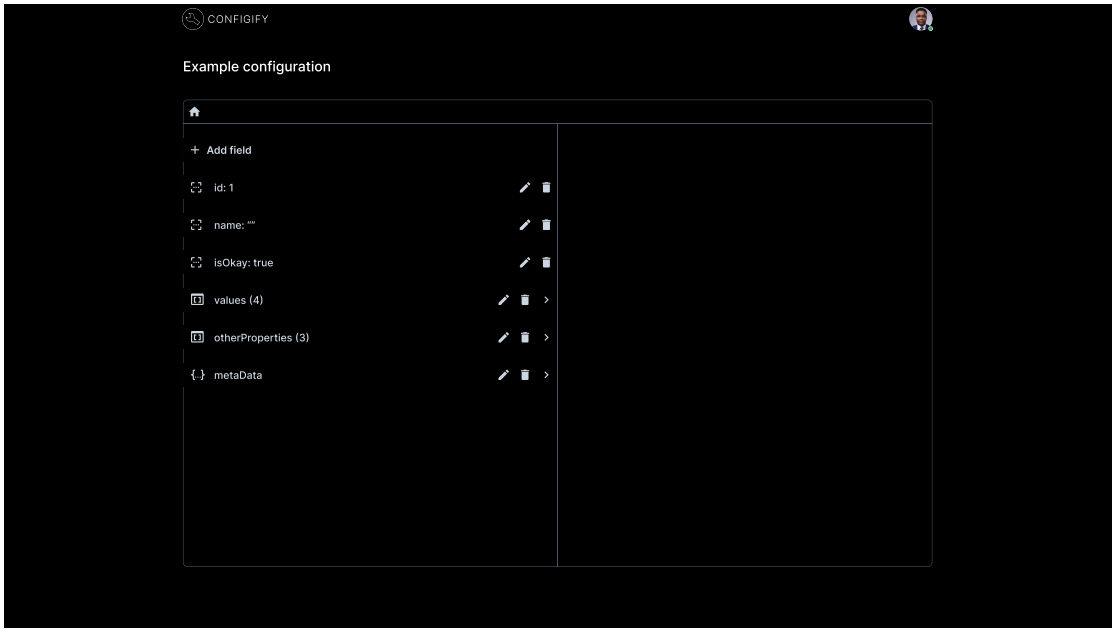


Figure C.5: A figure showing the configuration browser

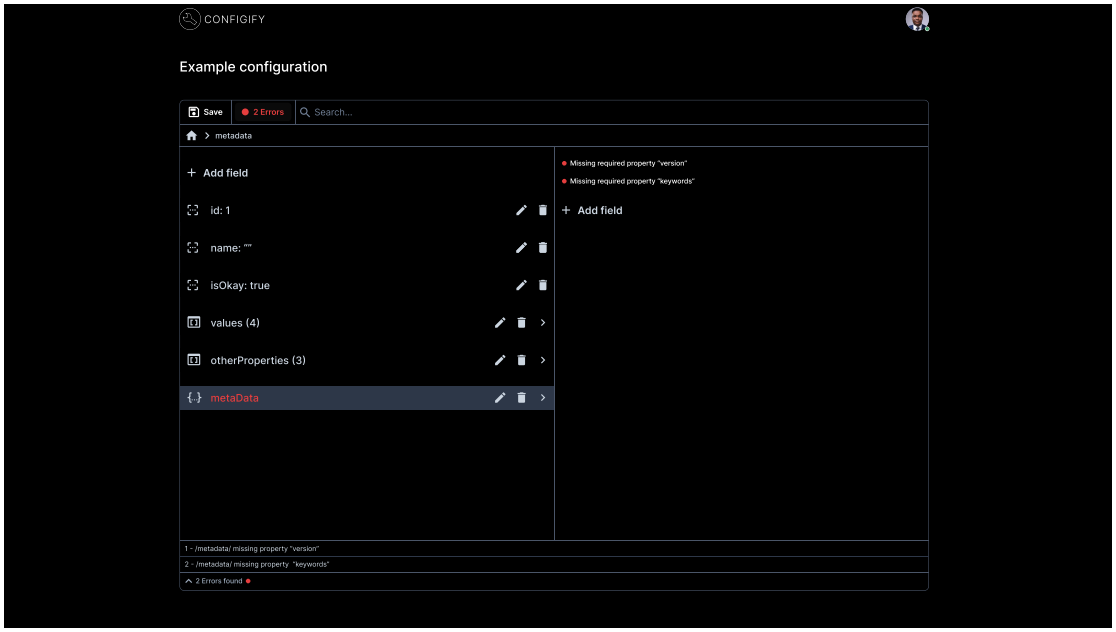


Figure C.6: A figure displaying the error messages

D - PROJECT MANUAL

Please refer to the separate attachment for additional information.

E - SYSTEM DOCUMENTATION

This appendix provides detailed information about the project's structure and code architecture. It includes an overview of the project's organization and a description of how the code is designed and implemented.

Contents

1 Architecture

The architecture of our full-stack web application consists of three main components: the frontend, backend, and database. These components work together to create a secure, scalable, and reliable system for managing configurations. In this section, we will provide an overview of the key features of each component and how they interact with one another.

1.1 Frontend

The frontend of our application was built using the Next.js framework and the ChakraUI component library. Next.js is a framework for building server-rendered React applications, and ChakraUI provides a set of pre-built React components that we used to create a customizable, accessible, and performant interface.

To integrate the frontend with the backend, we used tRPC, a TypeScript-based Remote Procedure Call (RPC) framework. tRPC provided us with a set of React Query hooks for querying and mutating data, as well as middleware for validating authorization for protected endpoints. We also utilized React Query's caching system to automatically refresh data based on various conditions.

Authentication in the frontend is handled using the NextAuth library, which provides a set of utilities for handling authentication actions such as sign ins and sign outs with different providers, account linking, and more.

1.2 Backend

The backend of our application was designed to provide seamless development, security, stability, and scalability. It uses Next.js' built-in API Routes, tRPC, Prisma, and AJV.

The RESTful API is built using tRPC, which allows us to create endpoints using procedures that define the route, input, and output of each endpoint. tRPC middleware is used to validate authorization for protected endpoints. Prisma is used to interact with the database, which is a PostgreSQL database.

Authentication in the backend is integrated with Prisma using a Prisma adapter for NextAuth, which allows us to store authorization data for each user, such as session data, information about different providers the user has authenticated with, scopes, and refresh tokens. tRPC can directly access this data to authenticate the user for protected endpoints.

AJV, a JSON schema validator, is used to validate all JSON configurations and schemas. It is contained within the validation tRPC endpoint.

Overall, the architecture of our application provides a reliable and scalable solution for managing configurations. The frontend and backend components work together

seamlessly to provide a user-friendly interface and a secure and efficient backend for managing data.

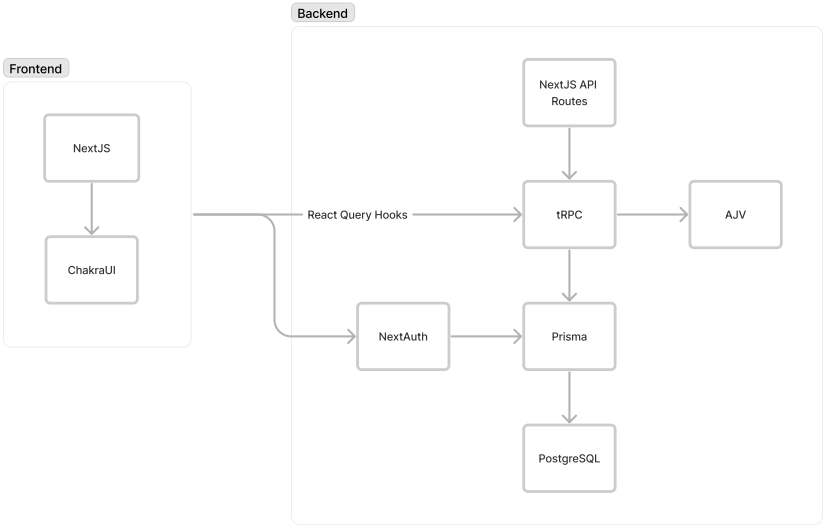


Figure E.1: A figure showing an overview of the software architecture

2 Requirements Documentation

2.1 Use-case diagram

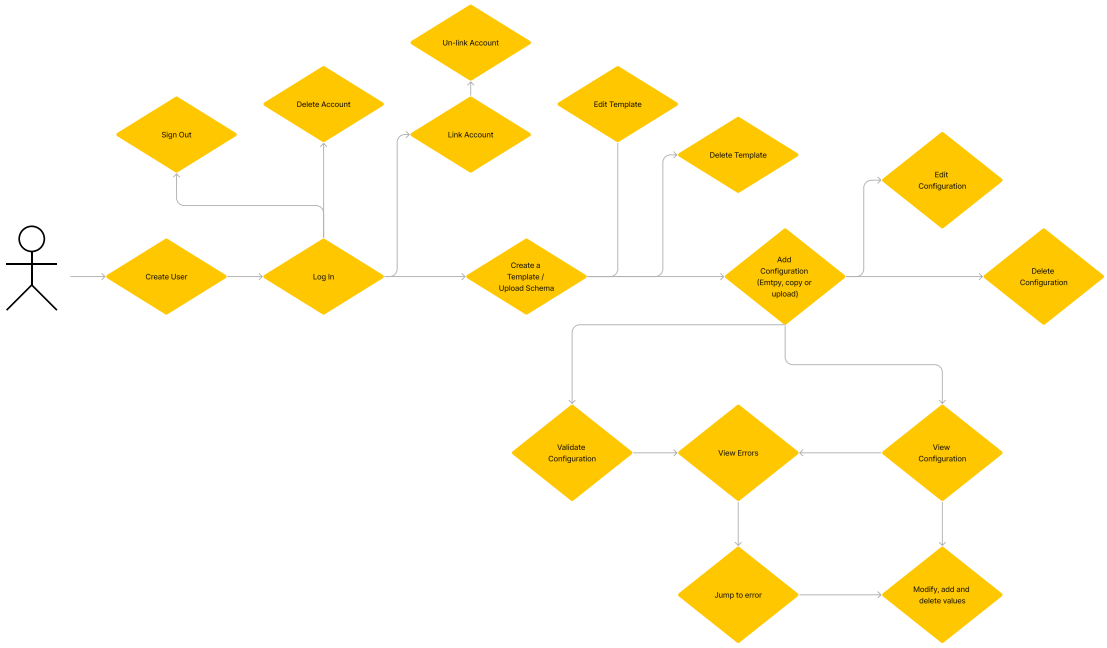


Figure E.2: Use-case Diagram

2.2 Project description

Head to the Appendix B, Project description to see the original task description and the MVP/stretch goals.

3 Project structure

Next.js provides a comprehensive solution for developing and deploying web applications with features for both front-end and back-end. Consequently, we have organized our code in the following manner. For the front-end, Next.js utilizes a file-system-based router, where each file in the "pages" folder represents a unique page of the application. The "components" folder contains reusable React components that can be imported into different pages, while the "public" folder stores static assets, such as images, fonts, or other files, that can be served by the application.

For the back-end, we have created a "server" folder to house server-side code that interacts with databases or external APIs, including our tRPC code. Additionally, we have other folders, such as "theme", "utils", or "types", that contain various components, utilities, and type definitions utilized throughout the project.

For a better overview of the directory tree, see the Figure E.3

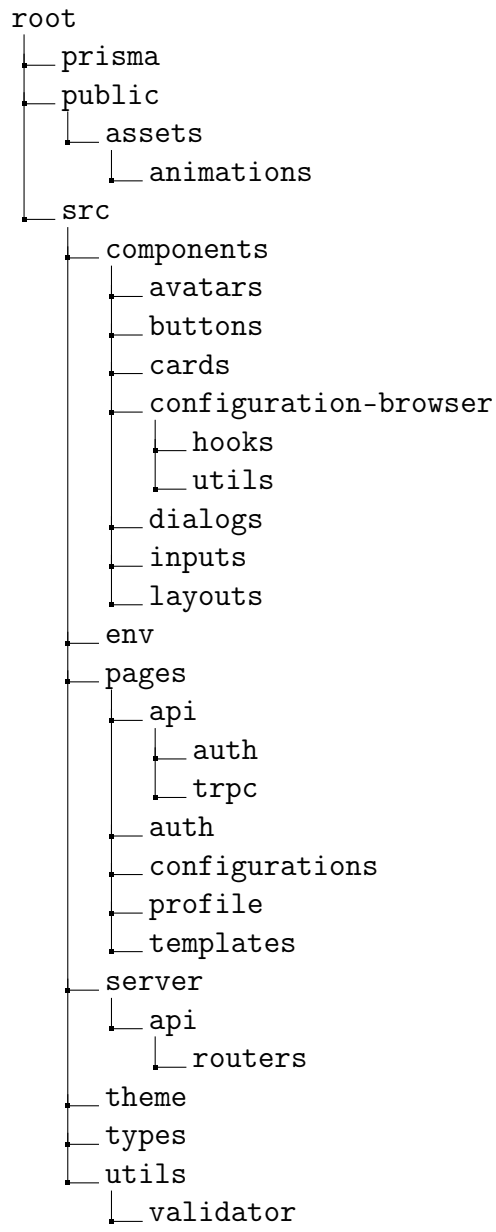


Figure E.3: The directory tree of the project

4 Functional programming

We employed functional programming paradigms in our project extensively. We utilized TypeScript, a super-set of the JavaScript language which provides support for both object-oriented and functional programming paradigms. Additionally, the Next.js framework, built on top of React, allows for the flexibility of choosing between functional and object-oriented programming paradigms with it's functional components and class components. The theory behind these two paradigms can be found in section 2.4 and section 2.5.

We opted to utilize the newer React functional components in our project, as they offer several key advantages. Functional components result in a more concise codebase that is easier to comprehend and maintain. This approach aligns with the functional programming principles of immutability and pure functions, enabling us to write cleaner and predictable code.

One of the key benefits of functional components is the ability to utilize React hooks. Hooks allow us to manage state and side effects in a more declarative and composable manner, enhancing code modularity and reusability. By leveraging hooks, we were able to effectively handle complex functionalities such as authentication and data fetching, including caching and refetching via TanStack Query hooks. The theory behind hooks has been covered in section 2.6 and more detailed info on TanStack Query can be found in subsection 3.2.5.

5 Database

The database for this project uses the Prisma ORM and is written as a Prisma schema. It is implemented using a PostgreSQL database. The schema itself is divided into several tables that represent the different entities in the system, including User, Account, Session, Template, Configuration, and VerificationToken.

The User model is used to store user information such as their name, email, and image URL. This model also has relationships with other models such as Account, Session, Template, and Configuration. The Account model is used to link user accounts from various providers, such as Google and GitHub. The Session model is used to store session information for authenticated users.

The Template model is used to store templates that can be used to validate configurations. The Configuration model is used to store user-generated configurations. This model includes a list of ConfigurationError entities that are associated with it and indicate any errors in the configuration. The VerificationToken model is used to store tokens that are generated when a user requests to verify their email address.
The complete database schema can be seen in Figure E.4.

6 Database schema

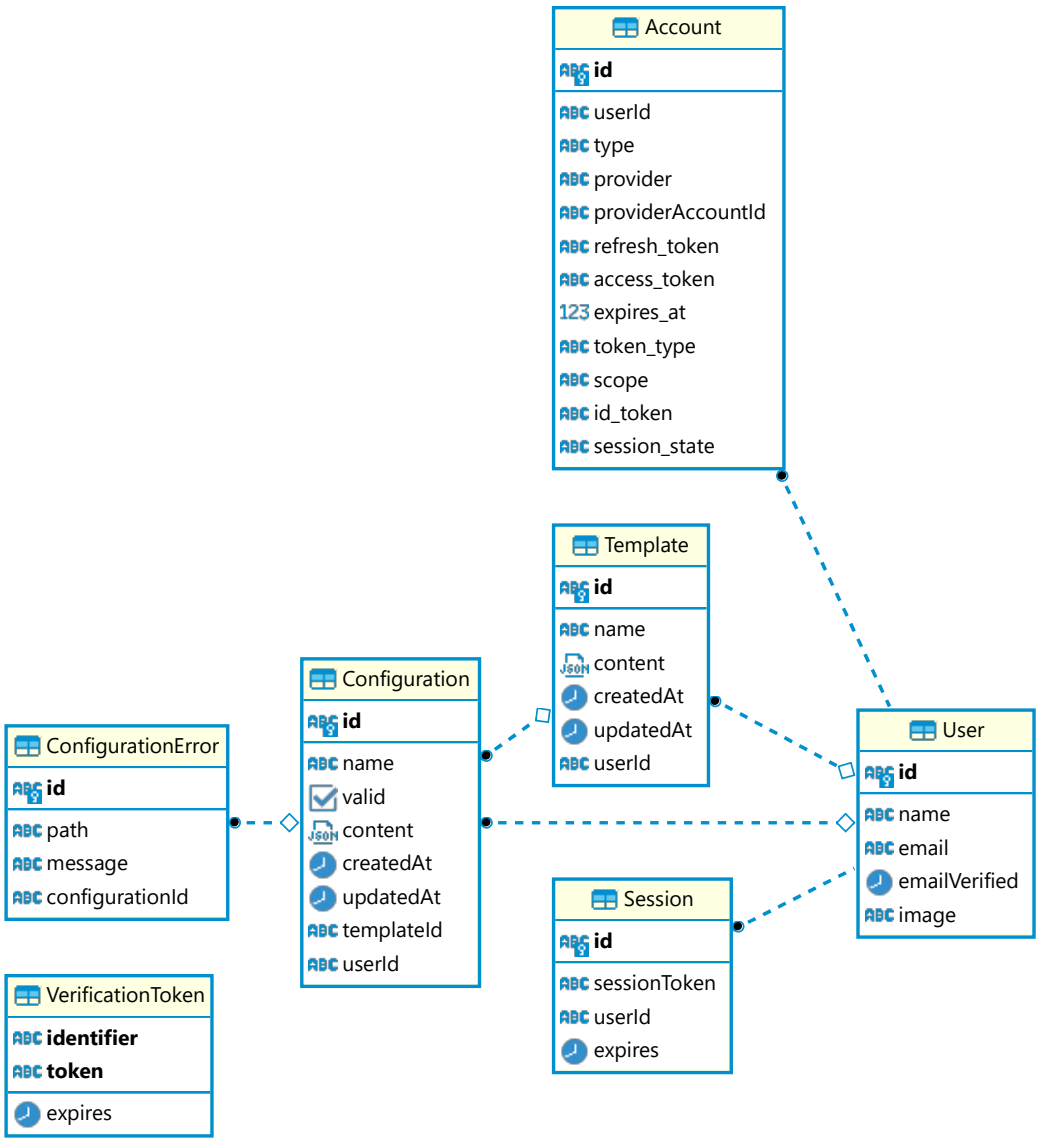


Figure E.4: A figure showing the database schema

7 Server endpoints

This subsection describes the various tRPC endpoints available on the server for different functionalities such as configurations, user management, templates, and validation. These endpoints allow clients to perform CRUD operations on various data entities on the server.

7.1 Configurations

These are the tRPC server CRUD operations for the configurations.

getAll
Retrieves all configurations for the authenticated user. It accepts an optional *templateId* query parameter that filters the configurations based on the parameter if given.

get
This method retrieves a single configuration by the *id* and the authenticated user id.

add
The add method creates a new configuration for the authenticated user. It accepts *templateId*, *name*, *valid*, and *content* inputs. The content input is validated against the JSON schema of the template associated with the configuration and the errors are generated if necessary.

delete
Deletes a configuration by *id* for the authenticated user.

clone
Creates a new configuration by cloning an existing configuration by *id* for the authenticated user.

download
This method retrieves a configuration by *id* for the authenticated user, encodes it as Base64, and then returns the configuration as a string.

update
Updates an existing configuration by *id* for the authenticated user. It accepts *name* and *content* inputs. The *content* input is validated against the JSON schema of the template associated with the configuration and the errors are generated as needed.

7.2 Me

These endpoints are used for account data fetching and management.

get
Retrieves the user information in a User object, including all associated accounts

based on the user *id* in the current session.

delete
Deletes the user account based on the user *id* in the current session.

unlink
Removes the association between the user account and the specified third-party account provider based on the user *id* and *provider* name given as the input.

7.3 Templates

REST endpoints for managing the templates.

getAll
Returns an array of all templates belonging to the currently authenticated user, along with the count of configurations associated with each template.

get
Returns a specific template belonging to the currently authenticated user, identified by the provided *id*.

add
This endpoint creates a new template belonging to the currently authenticated user, with the provided *name* and *content* (if any).

delete
Deletes a specific template belonging to the currently authenticated user, identified by the provided *id*.

update
The method updates a specific template belonging to the currently authenticated user, identified by the provided *id*. The name and/or content of the template can be modified, and if the content is modified, the associated configurations are checked for validity using a validation function before being updated.

7.4 Validate

Endpoint used for validation. This is also the only public endpoint on the server meaning it can be accessed without authentication.

validate
Validates a JSON configuration against a JSON schema and returns whether the configuration is valid and any errors encountered during validation. The configuration and schema are provided as strings in the input. Returns an object with a boolean value indicating whether the configuration is valid, and an array of error objects if any errors were encountered during validation. Each error object contains a path to the location of the error within the configuration and a message

describing the error.

8 Security

We have implemented several security measures to safeguard our data and server endpoints against unauthorized access.

In the context of tRPC, we have established both public and protected procedures. These procedures enable us to differentiate between endpoints that are accessible to the public and those that require authorization. For protected procedures, we employ a middleware method that receives the authentication session for each request, ensuring that only authenticated users can access our private endpoints. Additionally, this approach prevents users from querying data that does not belong to them by verifying the stored user authorization token.

Authentication is handled through the implementation of the NextAuth library. By leveraging NextAuth, we are able to eliminate the need to store or encrypt user passwords in our database. Instead, we utilize secure tokens provided by authentication providers to manage user sessions and access control. This approach significantly reduces the risk of password leaks and mitigates other common security vulnerabilities.

9 Deployment

We deployed our application to Vercel, a cloud platform for serverless deployment of web applications. Vercel was a great choice for our deployment needs as it provides seamless integration with Next.js and a continuous deployment feature which automatically deploys changes to the live site whenever changes are pushed to our GitHub repository.

In addition to deploying to Vercel, we also used a free database instance on Render.com for our production database.

9.1 Libraries

Here's a list of libraries and frameworks we used throughout the project.

- **@chakra-ui/react**: A library for building accessible and responsive user interfaces with React.
- **@emotion/react**: A library for writing CSS styles with JavaScript.
- **@emotion/styled**: A library for creating styled React components using CSS-in-JS.
- **@next-auth/prisma-adapter**: An adapter for using Prisma with NextAuth.js.

- **@prisma/client**: An auto-generated and type-safe database client.
- **@tanstack/react-query**: A library for fetching, caching and updating asynchronous data in React.
- **@trpc/client**: A library for making type-safe API calls to a tRPC server.
- **@trpc/next**: A library for integrating tRPC with Next.js.
- **@trpc/react-query**: A library for integrating tRPC with React Query.
- **@trpc/server**: A library for creating a tRPC server.
- **ajv**: A JSON schema validator.
- **chakra-react-select**: A Chakra UI styled version of the react-select component.
- **eslint**: A pluggable and configurable linter tool for identifying and reporting on patterns in JavaScript and TypeScript.
- **eslint-config-next**: An ESLint configuration for Next.js.
- **framer-motion**: A library for animating React components.
- **javascript-time-ago**: A library for formatting relative time strings.
- **lodash-es**: A modern JavaScript utility library delivering modularity, performance and extras.
- **next**: A React-based framework for building server-rendered or statically-exported React applications.
- **next-auth**: An authentication library for Next.js applications.
- **nodemailer**: A module for sending emails from Node.js applications.
- **prettier**: An opinionated code formatter that supports multiple programming languages.
- **react**: A JavaScript library for building user interfaces.
- **react-dom**: The entry point of the DOM-related rendering paths in React applications.
- **react-icons**: A collection of popular icon packs to use in React projects.
- **react-json-view**: A component for displaying and editing JSON data in React applications.
- **react-lottie**: A component for rendering Lottie animations in React applications.
- **react-time-ago**: A component for formatting relative time strings in React applications.

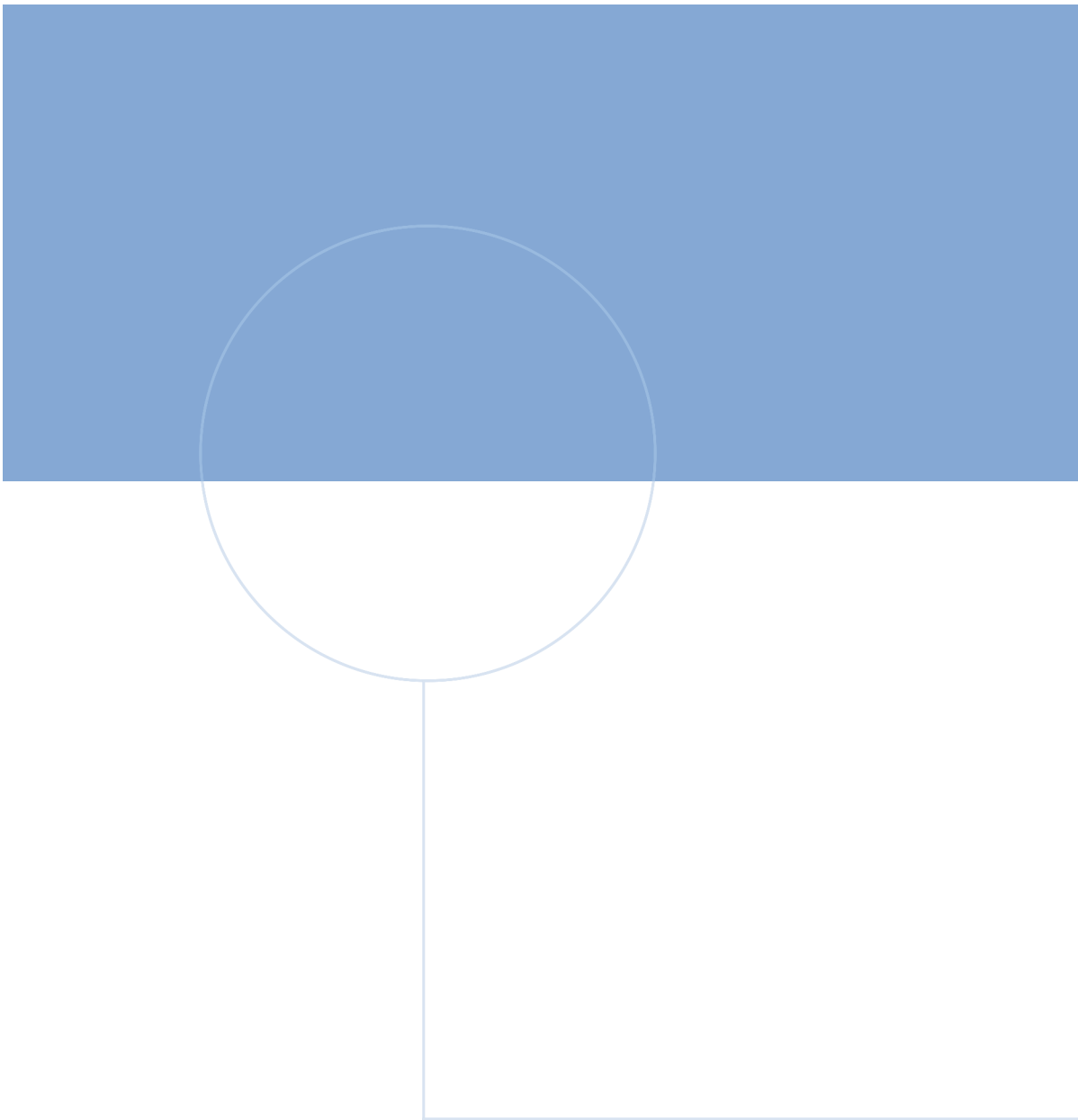
- **seedrandom**: A seeded random number generator for JavaScript, used to generate colors from strings in our project.
- **superjson**: An enhanced version of JSON that supports more data types and allows custom serialization logic.
- **trpc-openapi**: A tool for generating OpenAPI documentation from a tRPC server definition.
- **typescript**: A typed superset of JavaScript that compiles to plain JavaScript.
- **use-file-picker**: A hook for opening file picker dialogs in React applications.
- **zod**: A TypeScript-first schema validation library.

10 Documentation of source code

During the development of the project, we made sure to document the code appropriately. We used descriptive names for functions and variables to make the code more readable and understandable. For more complex or harder to understand pieces of code, we included comments using the // syntax to explain the logic behind them. Additionally, if we were inspired by or borrowed code from other sources, we included references in the comments.

F - REPOSITORY

The source code for our project can be found on GitHub at the following URL: `https://github.com/nilssen98/bachelor-project`. A .zip file containing the source code is also included as a separate attachment with this thesis.



NTNU

Norwegian University of
Science and Technology