Bachelor’s thesis (UAS)

Information and Communications Technology

2024

Yan Zborovskij

Design and Implementation of a Digital Competition and Collection Management System

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikan koulutus

2024 | 81 sivua

Yan Zborovskij

Digitaalisen kokoelman ja kilpailunhallintajärjestelmän suunnittelu ja toteutus

Tämä opinnäytetyö keskittyy dokumentoimaan prototyypin kehitystyötä web-pohjaiselle sovellukselle, jolla pyritään digitalisoimaan yliopiston isännöimien tapahtumiin osallistuminen ja niiden organisoinnin. Opinnäytetyöprojekti käsittelee haastetta hallita opiskelijoiden vuorovaikutusta yritysten osastoilla ja heidän osallistumistaan arvontaan paperipohjaisen passijärjestelmän kautta.

Opinnäytetyön päätavoitteena oli luoda digitaalinen järjestelmä, joka virtaviivaistaa tapahtumien hallintaa ja osallistumista.

Tavoite saavutettiin käyttämällä MongoDB:tä, Node.js:ää, Express.js:ää ja Vitea Tailwind CSS:llä. Kehitysprosessi sisälsi eri vaiheita, mukaan lukien projektidokumentaation, kehitysympäristön asennuksen, tausta- ja käyttöliittymäkomponenttien toteutuksen sekä vianmäärityksen.

Tuloksena oli toimiva web-sovelluksen prototyyppi, joka sisälsi yksityiskohtaisen projektisuunnitelman, tyylioppaan, kehitysympäristön, hyvin jäsennellyn tietokanta-arkkitehtuurin, malliskeemat, reitityslogiikan sekä toimivan sivun, joka suorittaa front-to-backend CRUD -operaatioita.

Tämä prototyyppi osoitti onnistuneesti tapahtumanhallintaprosessien digitalisoinnin toteutettavuuden ja hyödyt, mikä auttoi ymmärtämään web-sovelluskehityskäytäntöjä ja nykyaikaisten teknologioiden soveltamista todellisten haasteiden ratkaisemiseen.

Asiasanat:

verkkosovellus, ohjelmistokehitys, tietokanta, frontend, backend

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 81 pages

Yan Zborovskij

Design and implementation of a digital collection and competition management system

This thesis focuses on documenting the development of a prototype for web-based application aimed at digitalizing the process of event participation, stamp collection, and voting at university-hosted events.

This thesis project addresses the challenge of managing student interactions with company stands and their participation in sweepstakes through a paper-based pass system.

The main objective of thesis work was the creation of a digital system that streamlines event management and participation. Objective was achieved utilizing MongoDB, Node.js, Express.js, and Vite with Tailwind CSS. The development process encompassed various stages, including project documentation, setup of the development environment, implementation of backend and frontend components, and troubleshooting.

The result was a prototype which successfully demonstrated the feasibility and benefits of digitalizing event management processes, contributing to the understanding of web application development practices and the application of modern technologies in solving real-world challenges. This study contributes to the understanding of web application development practices and the utilization of modern technologies in addressing real-world challenges.

Keywords:

web-application, software development, database, frontend, backend

Contents

Glossary

Array	In software development, “array is a linear data structure where all elements are arranged sequentially” (GeeksForGeeks 2024). Each element is identified by key also known as array index.
Asynchronous communication	Asynchronous communication is any method of communication that does not require real-time back-and-forth (Cooks-Campbell 2023). In software development specifically allows making a request from client to backend without requiring an immediate answer from the server side.
Backend	The backend or server-side is the part of software that operates behind the scenes. Data management, processing requests, computations and logic are all performed in backend. It typically interacts with a database and frontend to deliver responses to users.
Development environment	A development environment is a collection of tools and recourses available for developers that allows debugging, coding, and testing software applications. Includes IDEs (integrated development environments), debuggers, code compilers, text editors, SDKs (software development kits) etc.

Frontend	Frontend is the part of software’s UI, design, and functionality that users interact with directly.
Function and function name	A function is a block of code that performs a specific operation. Typically, a function takes input parameters, performs computations, and returns a result. A function name is a way to identify the function and reference or call it.
JavaScript	High-level programming language that is widely used in software- and web development. Supported by most modern browsers, can also be used client-(in browsers) and server-side (with Node.js).
MongoDB	MongoDB is a document database used to build highly available and scalable internet applications (MongoDB 2024). Fairly popular NoSQL database program that uses a document-oriented data model.
Node.js	As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications (Node.JS 2024). Node.js executes JavaScript code outside of browser and allows asynchronous communication. Perfect tool for building networking and server-side applications.
NoSQL database	NoSQL databases are non-relational databases that store data in a manner other than the tabular relations used within SQL

	databases (Cousera 2024). NoSQL databases have flexible structures and use different query languages.
Object	An object is an instance of a class, containing attributes and methods. Objects allow modeling of real-world entities, where attributes and methods reflect properties and actions respectively.
Placeholders	In software development, temporary values, functionalities, or pages that are used to represent content or data that will be displayed at a later development stage. Widely used in UI to indicate dynamic content locations for instance.
Runtime environment	A runtime environment is either a framework or a platform in which programs are executed. It is an ecosystem where software can be run without the necessary need for explicit deployment. In a runtime environment, the program is provided with all the necessary system recourses and memory management to execute effectively.
Server and client	In client-server architecture, server is a computer system or a program that distributes recourses to other computer systems or programs. Recourse recipients are called clients and distribution happens over a network. Clients send a request to server, server makes necessary

	computations, performs tasks on their behalf, and returns data back to client.
Server-side processing	Server-side processing refers to data processing, execution of logic, generating responses, request handling, response generation. Essentially, background processes that allow server-client communication.
Variable	A variable is a named storage location that holds a value. Variables are stored in computer memory, have a data type, and can hold different value types such as strings, Booleans, integers, or objects.
Web-based	Refers to service or a software that is accessed and interacted with via web browser, instead of being installed and ran directly on the device.

1 Introduction

Currently Turku University of Applied Sciences (TUAS) hosts various competitions, recruitment fairs, and startup/company demonstrations at its premises.

Visitors can visit companies' stands, collect stamps and when a specified amount of stamps is collected, visitors can participate in a sweepstake. Visitors can also vote for any company in a specified category (e.g., the most interesting product/service). The company with the most votes in a specific category receives a prize from the event organizers.

The entire process, from visitors collecting stamps to company representatives receiving prizes, is currently relying on a manual paper-based system, which is inefficient, time-consuming, and organizationally challenging.

The goal of this thesis project is to document the development and implementation of a prototype for a web-based collection and voting management system that would digitalize aforementioned procedures, combining them in a robust and user-friendly environment. This web-based system will solve issues that TUAS is currently facing, will enable streamlining of organizational process and real-time event monitoring as well as will enhance visitor experience.

The system client architecture will be developed using Tailwind CSS for front-end design. It is a popular tool that enables simplicity of customization for design elements as well as has a coherent documentation. The back-end logic and functionality will be developed using a NoSQL database MongoDB, and JavaScript utilizing Node.js runtime environment. MongoDB is scalable if needed, which makes it an ideal choice for dynamic data storage. Node.js allows asynchronous communication between server and the client, as well as efficient server-side processing.

The thesis aims to document the development process, and will include all the steps, decision-making and justification, challenges faced, and solutions found during this process.

1.1 Overview of the thesis structure

This thesis is organized into several key chapters that document the process of developing a prototype for a web-based application aimed at digitalizing event participation, stamp collection, and voting at university-hosted events. Each chapter details specific stages of the project, from initial planning and documentation to the final implementation and testing of a functional prototype.

Chapter 1, titled “Introduction”, outlines the current state of affairs, namely the paper-based system in use at Turku University of Applied Sciences (TUAS) and highlights the inefficiencies and challenges associated with it. This chapter also presents the goals of the project, which include creating a digital platform to streamline these processes and enhance the overall experience for participants and organizers.

Chapter 2, titled “Project documentation”, covers the preliminary planning stages. This includes a breakdown of the interactive prototype, a comprehensive project plan, a style guide to ensure consistent design, and detailed flowcharts depicting the user journey through registration, login, and app navigation. This documentation provides a foundation for the development work in this project’s scope and scopes that will follow.

Chapter 3, titled “Development environment setup”, delves into peripheral tasks that need to be dealt with in order to create a smooth workspace. It starts with the database and backend setup and then it details steps such as software installation, securing connections, defining the database architecture, creating models, routing, and seeding the database with mock data. Following backend, the frontend setup is covered, focusing on the integration of ReactDOM and JSX. This chapter also addresses common troubleshooting issues and provides an overview of the initial Git commits, ensuring that the project’s codebase is well-organized and version-controlled from the get-go.

Chapter 4, titled the “Creation of the prototype’s first page”, is a detailed description of the development process that highlights the steps taken to set up both frontend and backend components, connect them, and implement features that utilize CRUD operations. This chapter concludes with testing the prototype page to ensure functionality and performance meet the project’s requirements.

Finally, Chapter 4, titled “Conclusion”, summarizes the achievements of the thesis, reflecting on the challenges encountered and the solutions implemented. It also discusses the broader implications of the project, such as its potential impact on event management practices at TUAS and similar institutions.

2 Project documentation

In this chapter project documentation is addressed in full capacity. An already existing documentation is analyzed, additional essential documentation, such as project plan and style guide are created.

This is an essential aspect in establishing an organized and modular development environment, both for the developer and the commissioner. Telegraphing intentions while having a scaffolding like guideline is important and useful.

2.1 Interactive prototype breakdown

Documentation and work previously carried on the project, namely an interactive prototype for the software under construction, will serve as a basis for flowchart and future database structure documentation. The full version of the interactive prototype was created by a student (Mahlamäki 2024).

The interactive prototype is also under construction, some of the views are non-existent and some are not yet functional, but basic understanding of application's purpose still can be derived from it.

Figure 1 illustrates web-application's front page when user is going to the web-address for the first time.

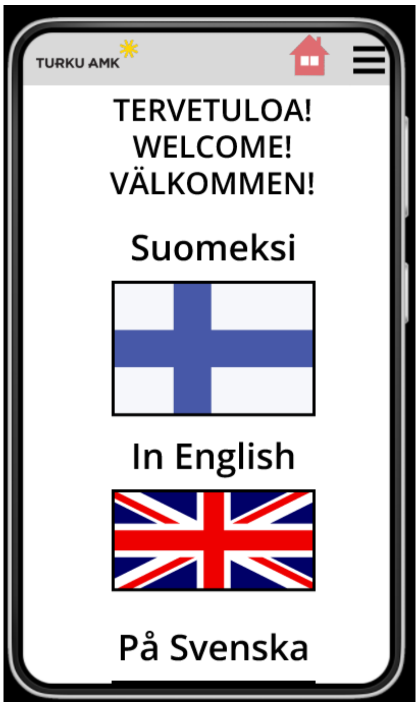


Figure 1. Screen capture of interactive prototype’s language selection page.

The user is prompted to choose a language, after language is chosen, user is prompted to GDPR text that can be either accepted or declined. After successfully choosing the language and accepting GDPR, the user is redirected to a home page illustrated in Figure 2.

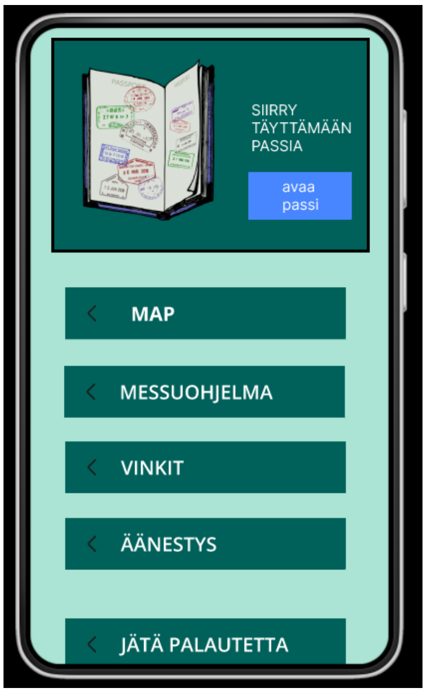
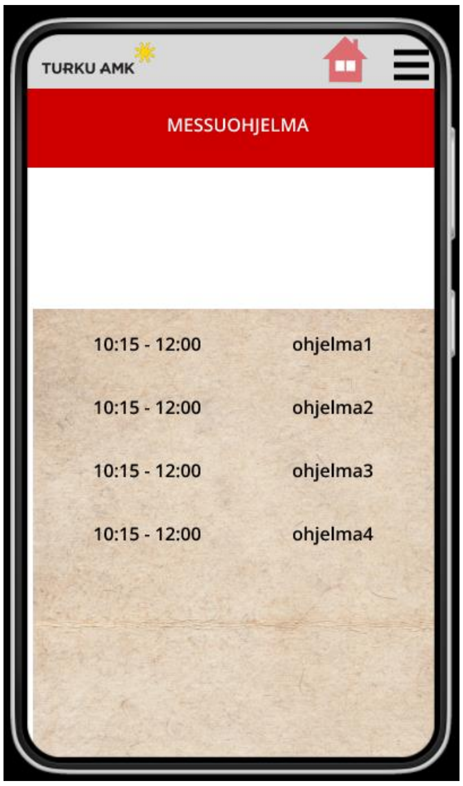


Figure 2. Screen capture of interactive prototype's home page view.

From the home page, users can access six different views. First is the stamp collection view, where the user enters and the company’s unique code that serves as a stamp. After completing the stamp pass, in this case filling all the required code fields, the user can press continue to return their pass and therefore becoming eligible for a sweepstake participation.

The second page is a map view, digitalized layout of TUAS campus buildings will be located on that page to help visitors navigate the premises.

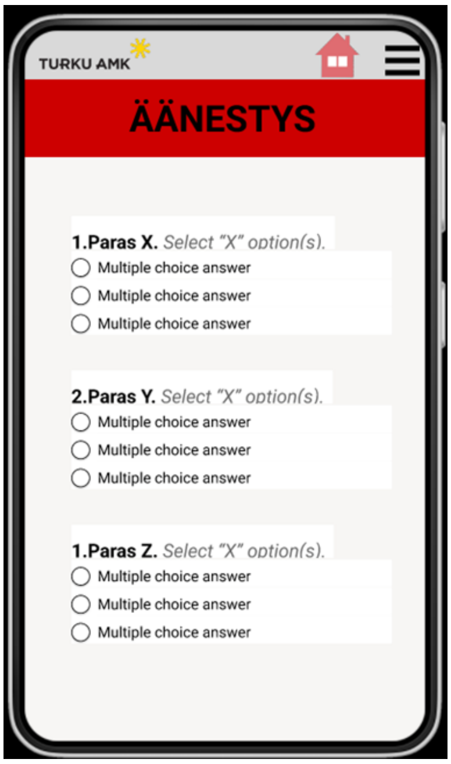
The third page is an event program page, time schedule and locations of key points of interest etc. The page will be filled out by an organizer via a form in admin view (Figure 3).



10:15 - 12:00 ohjelma1
10:15 - 12:00 ohjelma2
10:15 - 12:00 ohjelma3
10:15 - 12:00 ohjelma4

Figure 3. Screen capture of interactive prototype's event program view.

The fourth page is a voting view, where the user will be able to vote for a candidate that performed the best in a sub-category. Category will contain all the candidates, as a multi-choice or a dropdown menu depending on the number of the candidates. Users will be able to choose one candidate for each category (Figure 4).



1.Paras X. Select "X" option(s).
- Multiple choice answer
- Multiple choice answer
- Multiple choice answer

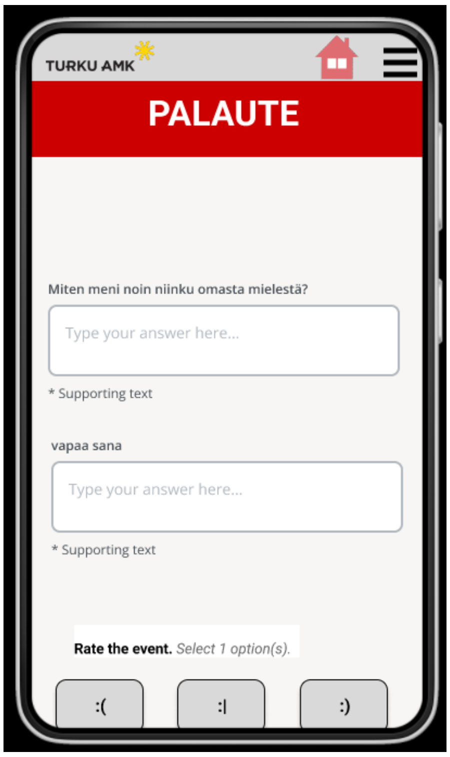
2.Paras Y. Select "X" option(s).
- Multiple choice answer
- Multiple choice answer
- Multiple choice answer

1.Paras Z. Select "X" option(s).
- Multiple choice answer
- Multiple choice answer
- Multiple choice answer

Figure 4. Screen capture of interactive prototype's voting view.

The fifth page is a user feedback form page, where users can leave their general event feedback as well as some recommendations for future events (Figure 5).

Lastly, the sixth page is called "Hints", and its function is quite elusive, and its purpose is unknown (Figure 6).



TURKU AMK

PALAUTE

Miten meni noin niinku omasta mielestä?

Type your answer here...

* Supporting text

vapaa sana

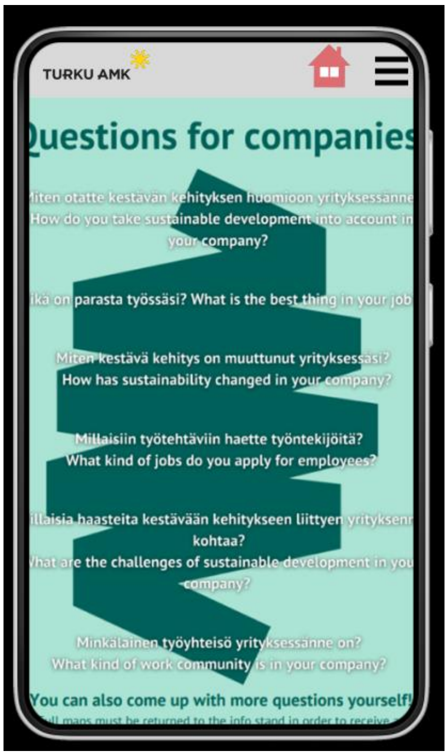
Type your answer here...

* Supporting text

Rate the event. *Select 1 option(s).*

:(:| :)

Figure 5. Screen capture of interactive prototype's user feedback view.



TURKU AMK

uestions for companies

iten otatte kestävän kehityksen huomioon yrityksessänne
How do you take sustainable development into account in
your company?

ikä on parasta työssäsi? What is the best thing in your job

Miten kestävä kehitys on muuttunut yrityksessäsi?
How has sustainability changed in your company?

Millaisiin työtehtäviin haette työntekijöitä?
What kind of jobs do you apply for employees?

illaisia haasteita kestävään kehitykseen liittyen yrityksen
kohtaa?
hat are the challenges of sustainable development in you
company?

Minkälainen työyhteisö yrityksessänne on?
What kind of work community is in your company?

You can also come up with more questions yourself!

ull maps must be returned to the info stand in order to receive

Figure 6. Screen capture of interactive prototype's hint view.

The documentation is somewhat lacking. There is no contextual documentation to accompany the interactive prototype. Upon initial review, it was decided that some features will be excluded from the upcoming project's scope. These features include the Swedish and Finnish language versions of the prototype, the “Hints” page, the “User Feedback” page, and the “Map” page. These features are deemed unnecessary for the current phase and can be implemented in later stages of development.

The next section details the project’s scope.

2.2 Project plan

To start a project, first coherent project documentation is needed and is missing. Most often after the initial planning phase, “the project manager creates a more detailed project plan. It serves as a roadmap for the project, defining the key project milestones and placing them on a timeline” (Nuclino 2022). An open-source project plan documentation template was used as a reference document for the current project.

The document consists of 4 parts that are broken down into sub-categories, some of which are relevant to the thesis and will be elaborated on.

1. Introduction

   1.1 Introduction and Project Goal, where overall project aim, and current situation is described

   1.2 Project Scope and Outcome, where scope of the project as well as expected project outcome is described

   1.3 Project Limitation, where project limitation is defined and specified

   As of now, the current status involves verbal description of the desired end product, accompanied by a wireframe and an interactive prototype.

The goal of the project is to design and implement a prototype of web-based application in accordance with provided specifications, as well as to lay groundwork for future development.

Prototype will include functioning architecture, core functionalities such as deployable front- and backend, and a database. Additionally, a prototype page that utilizes all the aforementioned components. Version control is going to be managed via GitHub online repository.

Verbal description briefly touched on QR-scanning: project's gold will not include QR-scanning feature.

WebRTC (Web Real-Time Communication), a technology that "is available on all modern browsers as well as on native clients for all major platforms" (WebRTC.org 2024), makes QR-scanning feature entirely possible, however granting device component permissions has the potential to expose users to unnecessary cybersecurity risk. For instance, "Threat actors clone an authentic QR code that redirects you to a malicious site or infects your device with malware to extract your personal data when you scan it" (Canadian Centre for Cyber Security 2024).

A simple 4-digit unique ID assigned to a company serves the same purpose and up 10 000 of unique IDs can be generated from 4 digits, which will be enough to cover years of competitions. 5-digit unique ID provides 100 000 combinations respectively. IDs are less risky and more robust.

2. Organization
   2.1 Project Group, where project members' roles and contact details are located
   2.2 Customer Information, where customers' roles and contact details are located

3. Project Implementation Plan
    3.1 Schedule, where project stages are defined and described
    3.2 Cost estimate
    3.3 Resource plan, where project members' estimated weekly hours put into the project are defined
    3.4 Software and Hardware, where software and hardware that is going to be used in the project is defined and described
    3.5 Outcome Delivery, where the way finalized project is going to be delivered to the customer

4. Project Management Plan
    4.1 Meetings and Communications, where preferred internal and external communication methods are defined
    4.2 Documentation Storage and Code Repository, where appropriate information is located
    4.3 Project Quality Goals

2.3 Style guide

Next step in creating project documentation has been decided to be the creation of a style guide.

A programming style guide is an opinionated guide of programming conventions, style, and best practices for a team or project (Angel 2018). It helps in unification and overall code consistency if a lot of people are working on the same program. Usually contains rules for naming, commenting, formatting etc. Content of the document varies from project to project.

The current project's style guide contains instructions on file names, function names, variables, comments, Git commits, branches and pull requests. For instance, "Variables must be named using camelCase (´const exampleVariable`). Name must relay the purpose the variable serves" is a typical instruction developer would find in a style guide document.

Some guidelines fall outside of readability and uniformity of code, for example one of the guidelines in this project’s style guide states: “Use either ´const` or ´let`. ´const` is a default, ´let` is used only if the variable is up for redefinition somewhere else. ´var` is not used anywhere period.”

Variables declared with ´const` or ´let` have block-level scope. Block-level scope means that variables are limited in scope to the block in which they were defined, while variables declared with ´var` have a function-level scope.

Block-level scope “restricts the variable that is declared inside a specific block, from access by the outside of the block” (Geeksforgeeks 2022). ´let` and ´const` keywords facilitate the variable that comes after them to be block-level scoped (Code 1).

Code 1. An example of block-level scope variable declaration.

```
function exampleFunction() {
  const localVariable = 'I am outside ´if` statement block';
  if (true) {
      const innerVariable = 'I am inside ´if` statement block';
      console.log(localVariable); // <- variable declared outside of
´if`statement block is available inside of it
  }
  console.log(innerVariable); // <- variable declared inside of
´if`statement block is NOT available outside of it
}
exampleFunction();
```

Using block-level scope variables is considered “best practice” amongst developers, reduces mutable state, helps to avoid accidental global variable declarations. In addition, modular code is easier to test and troubleshoot.

Ultimately, the style guide contains all the essential information to maintain integrity and uniformity of code within the project.

2.4 Flowchart

A flowchart is a diagram that depicts a process, system, or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. (Lucidchart 2024).

For this particular project flowchart is highly beneficial, as it helps in several aspects of developing, namely clarifying and understanding fundamental requirements, guiding testing, and implementation, communicating, and illustrating system logic, pinpointing key components and user interactions, as well as potential issues and bottlenecks.

Flowchart was created using a free online diagram creation tool called draw.io.

It was decided to separate the flowchart into two parts to maintain ease of reading and understanding, as flowcharts are generally prone to inflation with unnecessary details.

2.4.1 Registration & log in

Flowchart is created assuming the user’s behavior to be in user’s best interest i.e. user is not trying to jeopardize system’s performance nor to abuse functionality. It is assumed that the user is interacting with software in an intended manner.

Registration and log in sequence logic is illustrated below. Starting from the “Home page” node, potential paths and deviations can be tracked and analyzed with the help of flowchart. These are broad strokes, rudimentary but robust guidelines for a developer to base their work off of. Registration and log in flowchart are demonstrated in Figure 7.

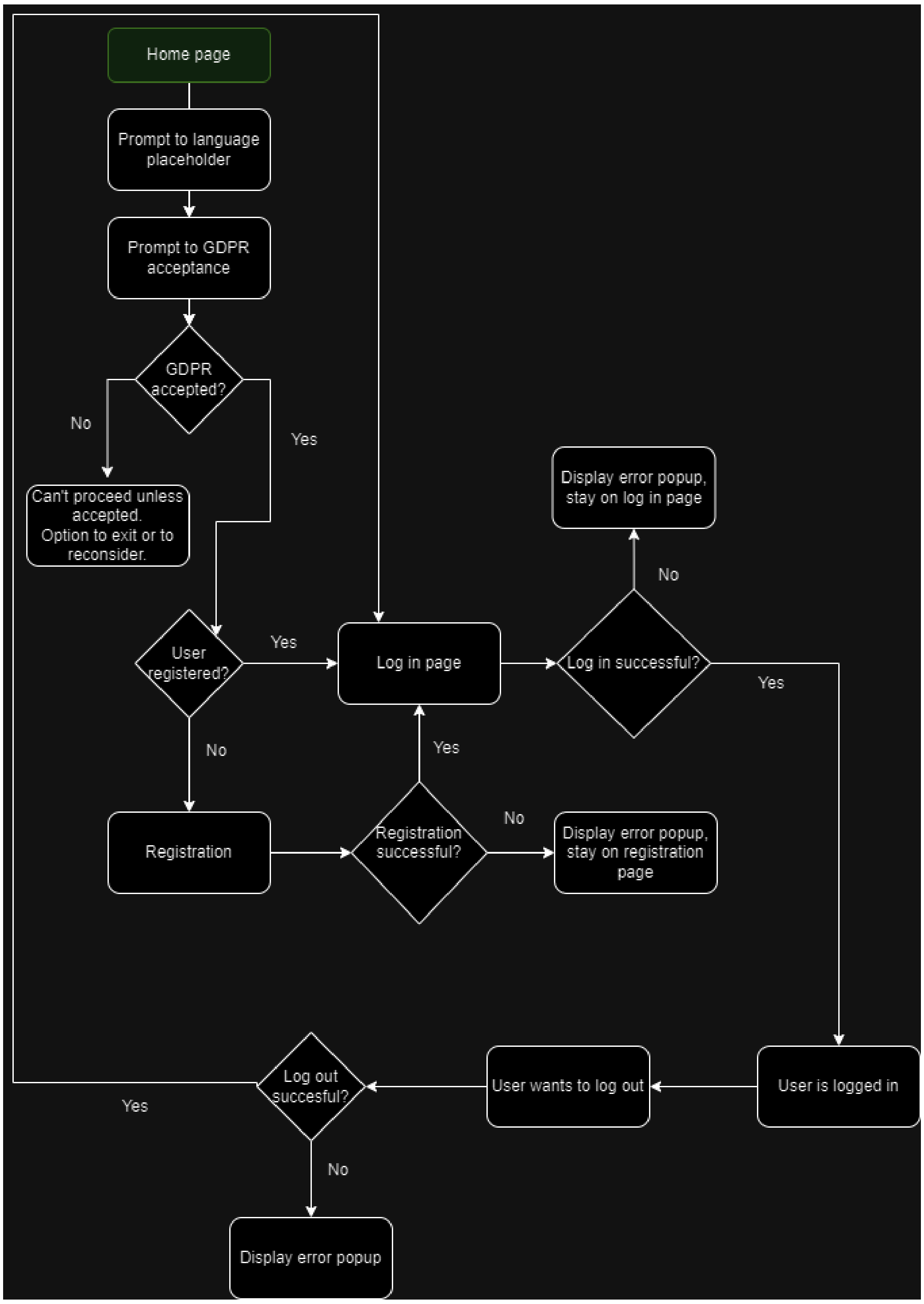

Home page
Prompt to language placeholder
Prompt to GDPR acceptance
GDPR accepted?
No
Yes
Can't proceed unless accepted. Option to exit or to reconsider.
User registered?
Yes
No
Log in page
Log in successful?
No
Yes
Display error popup, stay on log in page
Registration
Registration successful?
Yes
No
Display error popup, stay on registration page
User is logged in
User wants to log out
Log out succesful?
Yes
No
Display error popup

Figure 7. Registration flowchart.

2.4.2 Navigating the app

After successfully registering and authenticating, the user will be prompted to the main menu. From the main menu or home page (Figure 8), users will be able to navigate to stamp collection view, map, event program, voting view and their user profile.

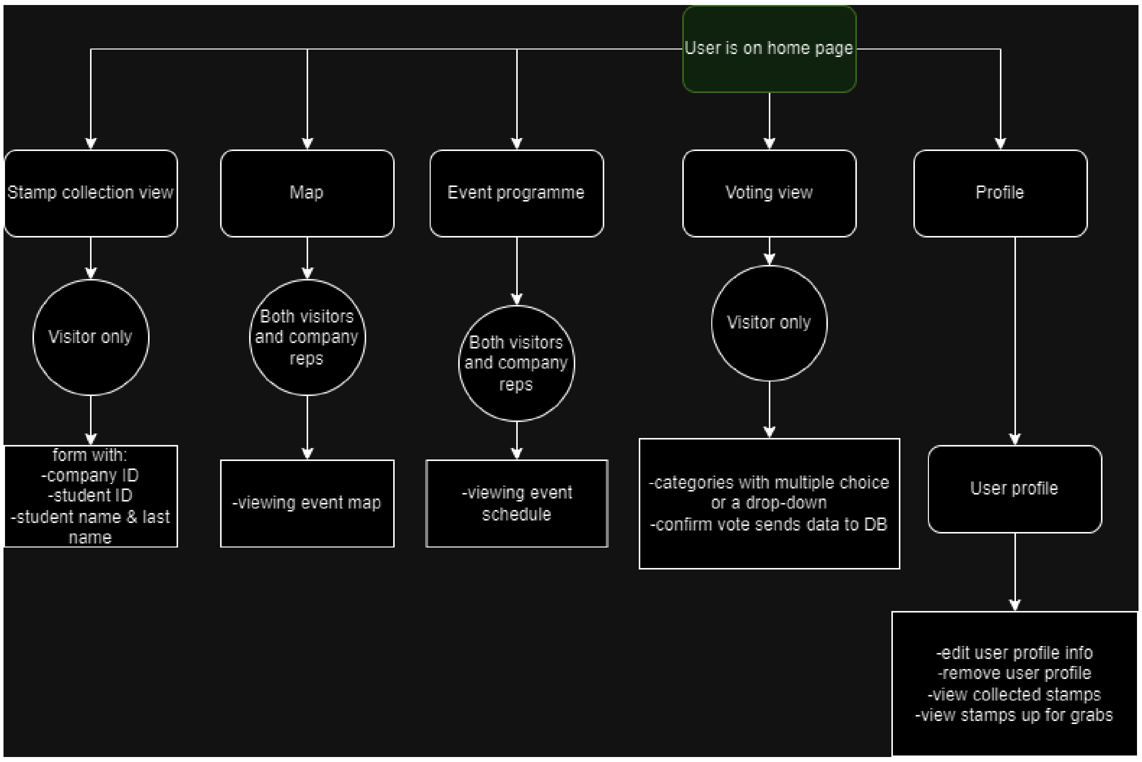

User is on home page
Stamp collection view
Map
Event programme
Voting view
Profile
Visitor only
Both visitors and company reps
Both visitors and company reps
Visitor only
form with: -company ID -student ID -student name & last name
-viewing event map
-viewing event schedule
-categories with multiple choice or a drop-down -confirm vote sends data to DB
User profile
-edit user profile info -remove user profile -view collected stamps -view stamps up for grabs

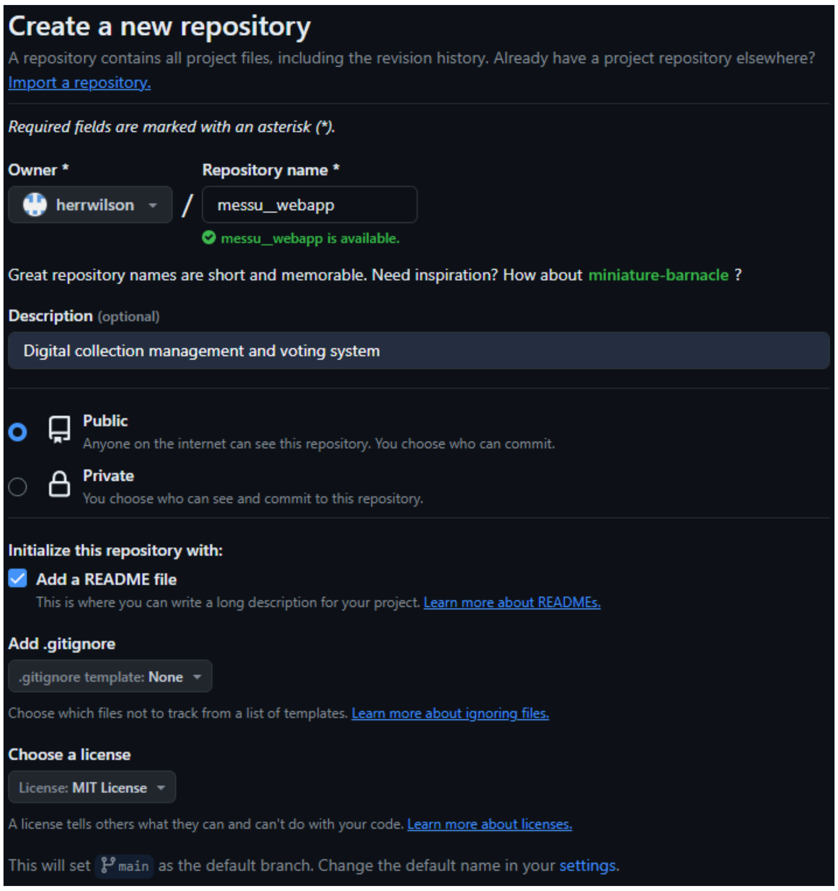
Figure 8. draw.io .png export of navigation flowchart.

3 Development environment setup

It has been decided to use GitHub for version control, that's where the first step of project initialization happens.

Git repository can be created via Git interface. It requires a unique name, must be either public or private with the only difference being permission to see. Public is for everyone with access to the internet to view and private for selected recipients only.

With prefabricated README file checked and MIT license chosen project repository is now public and ready to be cloned to a local machine. Repository creation view is depicted in Figure 9.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner *: herrwilson / Repository name *: messu__webapp

messu__webapp is available.

Great repository names are short and memorable. Need inspiration? How about miniature-barnacle ?

Description (optional): Digital collection management and voting system

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private: You choose who can see and commit to this repository.

Initialize this repository with:

[✓] Add a README file — This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore: .gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license: License: MIT License

A license tells others what they can and can't do with your code. Learn more about licenses.

This will set `main` as the default branch. Change the default name in your settings.

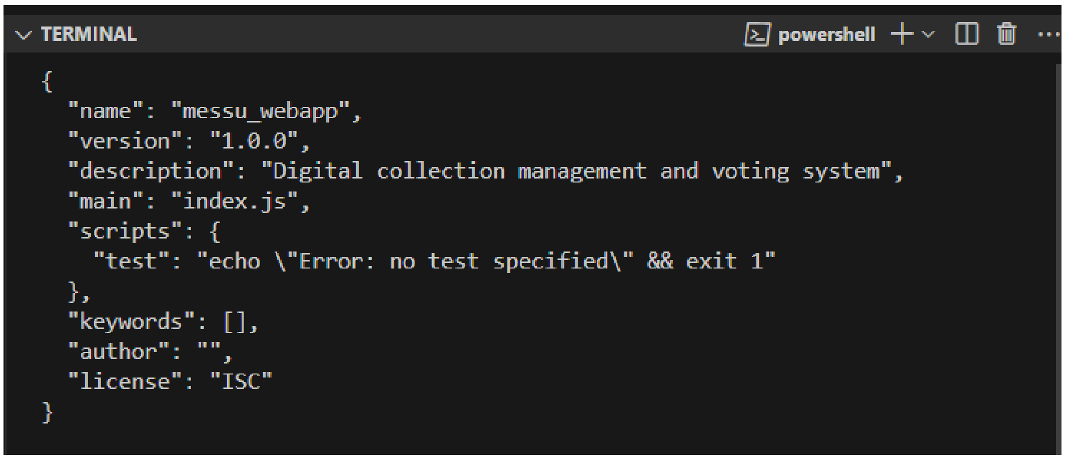
Figure 9. Screen capture of Git interface.

After successful cloning, the project is a blank slate. Main root directory ´messu_webapp` only contains LICENSE file and a README.md file. A software license, such as MIT License chosen for messu_webapp “is a permissive software license that places few restrictions of reuse. Users of software using an MIT License are permitted to use, copy, modify, merge publish, distribute, sublicense, and sell copies of the software” (University of Pittsburgh 2023).

For future initialization, two additional directories are created into root: ´frontend` and ´backend`.

It is paramount to separate the frontend development section from the backend development section. Having organized and modular directories is considered a healthy practice and is overall very helpful.

To initialize a new Node.js project, ´npm init -y` needs to be run in the terminal. This command initializes Node.js project and creates a ´package.json` file. The purpose of this file is to list runtime and development dependencies that project requires, contain scripts, versions, and metadata. Figure 10 shows appropriate terminal output after Node.js is initialized.



```
{
  "name": "messu_webapp",
  "version": "1.0.0",
  "description": "Digital collection management and voting system",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Figure 10. Screen capture of project initialization terminal output.

Last step is running ´npm install`, a command that parses through generated ´package.json` file, detects mentioned dependencies and installs necessary node modules.

3.1 Database and backend

Now that general development workflow documentation is created and empty project is cloned to a local machine, a database infrastructure can be established.

Most projects are powered by relational databases. A relational database is a collection of information that organizes data in predefined relationships where data is stored in one or more tables (or "relations") of columns and rows, making it easy to see and understand how different data structures relate to each other (Google Cloud 2024).

Typical database structure is divided into components such as tables, rows, columns, primary keys, foreign keys, indexes, constraints, and views. Essentially, it is the project’s digitalized memory bank and is paramount in the development of any web-based application.

Table is a primary database structure, which represents a collection of related data entities. For instance, in a car dealership’s database you would find tables for car brands, car models, customers etc.

Rows, also referred to as tuples or records, are individual table entries. Each row is a single instance of data. In a car dealership’s database, in a car model table each row can represent a specific car model with attributes such as year it was released, ratings, price etc.

Columns, also known as attribute or field, define the type of data that each row can contain.

The primary key serves as a unique identifier for each row in a table. Typically composed of one or more columns with unique values for each row.

Foreign key on the other hand is a c Column or columns that reference the primary key of another table. Essentially the foreign key is establishing a relationship between tables. For instance, in a database for a car dealership, the table that stores orders can contain a foreign key column referencing customer ID from the customers table.

Index is a data structure that enhances speed and efficiency of data retrieval operations. Indexes enable quick lookup based on a column or criteria.

Constraint can be viewed as a rule or condition imposed on the data in a database. Serves data consistency and integrity purposes. Typical constraints would include primary key constraints or foreign key constraints.

Finally, view is a virtual table, a way to present customized data without altering the contents of it.

MongoDB was chosen for this project as it has several compelling advantages. Firstly, MongoDB is a NoSQL database, which allows skipping some of the relational database pre-requisites such as creating a relation schema and other diagrams.

It allows storing heterogeneous data within one collection, which adds a flexibility layer for data structuring. This flexibility helps greatly in the process of development, especially if development requires adaptation on the spot, which is definitely the case for this project.

Lastly, solid querying and indexing performance of MongoDB is going to enhance responsiveness and efficiency of this web application.

3.1.1 Database software installation and setup

Firstly, MongoDB and peripheral software needs to be installed. For a prototype under construction no hosting services will be required, so Atlas, a MongoDB hosting service will not be used. Database will be stored and hosted locally on a developer’s machine.

For the project’s current needs, a MongoDB-shell, software that allows direct interaction with the database itself, is essential.

Interaction is encapsulated in CRUD operations. CRUD is an abbreviation derived from Create, Read, Update and Delete. These are four basic persistent storage functions.

“Create" is an operation that creates new documents or other data entries and typically inserts new data into a database.

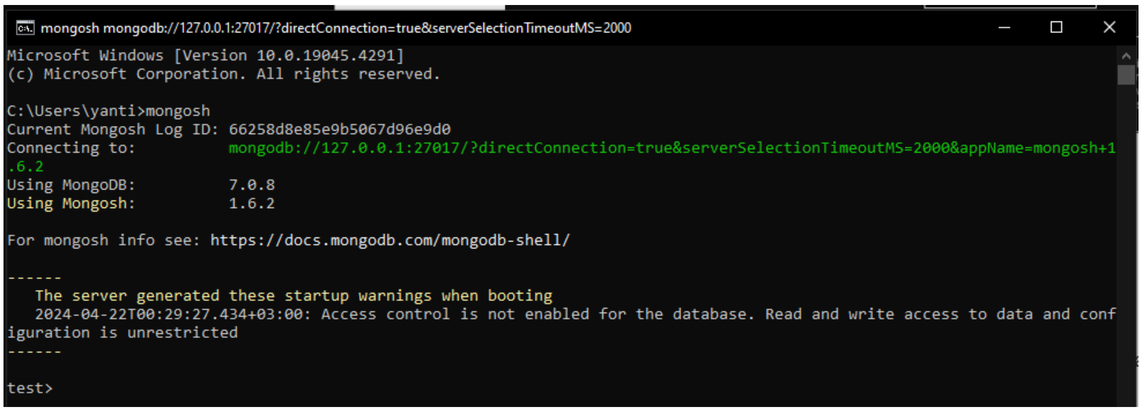
“Read” is related to retrieving or reading data that already exists inside of the database, fetching specific documents or other data entries from database based on a specific criterion.

“Update” involves modifying existing data and allows changing values of fields in the database.

“Delete” is an operation that removes records or documents from anywhere in the database.

MongoDB Compass, a graphical user interface, visual representation of said data will is not essential, but convenient. It helps to view and interact with contents of MongoDB without using shell commands.

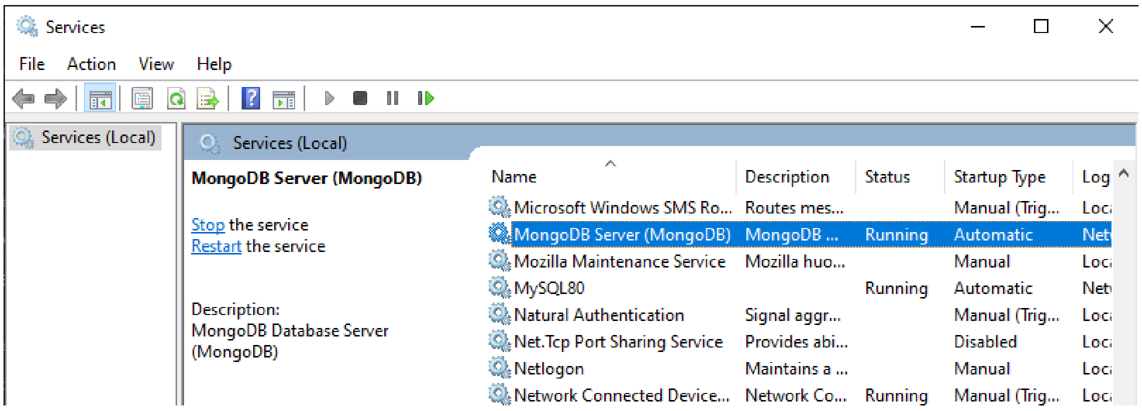
To ensure that MongoDB and MongoDB Shell have been successfully installed, a ´mongosh` command can be run. This command launches MongoDB Shell and allows immediate interaction with MongoDB. Expected output is shown in Figure 11.


mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

```
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\yanti>mongosh
Current Mongosh Log ID: 66258d8e85e9b5067d96e9d0
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.6.2
Using MongoDB:          7.0.8
Using Mongosh:          1.6.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-04-22T00:29:27.434+03:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test>
```

Figure 11. Screen capture of command prompt output.

Additionally, MongoDB Server service’s status needs to be “Running” on Windows system in order for the database to function properly. It can be confirmed in Windows’ local services menu (Figure 12). Having the service running on a background of Windows eliminates the need to launch it manually each time the project is booting. Different developers, however, use different OS configurations, so having a manual service start option is going to be useful and will be created soon.



MongoDB Server (MongoDB)

Stop the service
Restart the service

Description:
MongoDB Database Server (MongoDB)

Name	Description	Status	Startup Type	Log
Microsoft Windows SMS Ro...	Routes mes...		Manual (Trig...	Loc
MongoDB Server (MongoDB)	MongoDB ...	Running	Automatic	Net
Mozilla Maintenance Service	Mozilla huo...		Manual	Loc
MySQL80		Running	Automatic	Net
Natural Authentication	Signal aggr...		Manual (Trig...	Loc
Net.Tcp Port Sharing Service	Provides abi...		Disabled	Loc
Netlogon	Maintains a ...		Manual	Loc
Network Connected Device...	Network Co...	Running	Manual (Trig...	Loc

Figure 12. Screen capture of Windows OS local services window.

Lastly, to integrate MongoDB a ´mongoose` package need to be installed with ´npm install mongoose`, as Figure 13 demonstrates. Mongoose is a tool that integrates Node.js applications with MongoDB, which enables developers to interact with MongoDB using JavaScript syntax.

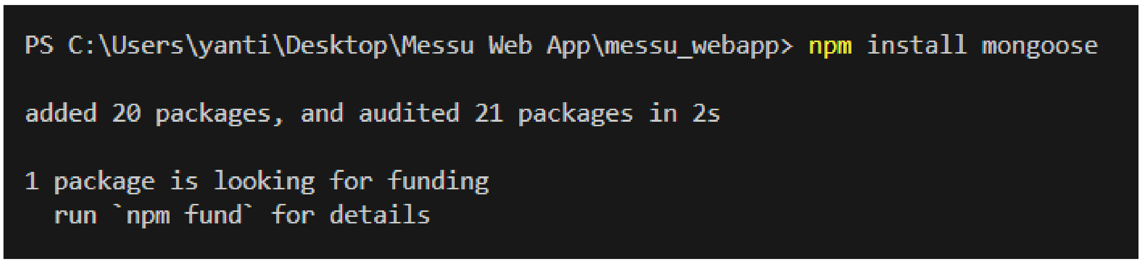

```
PS C:\Users\yanti\Desktop\Messu Web App\messu_webapp> npm install mongoose

added 20 packages, and audited 21 packages in 2s

1 package is looking for funding
  run `npm fund` for details
```

Figure 13. Screen capture of mongoose installation process.

3.1.2 Securing connection to a local database

To connect messu_webapp to a database locally hosted by MongoDB, a configuration file needs to be added to project’s backend directory. Inside of that file, a connection string needs to be defined. In this case, ´config.js` contents are shown by Code 2.

Code 2. Contents of ´config.js`.

```
module.exports = {
  mongoURI: "mongodb://127.0.0.1:27017/messu_webapp",
};
```

This connection string is going to be used in another new Express server file, which will serve as an entry point for Node.js application. It will be called ´server.js`. The file will contain necessary module imports such as Express.js and Mongoose, as well as Express application creation, and connection to MongoDB using connection string defined in ´config.js`. Essential scaffolding contents of ´server.js` are in Code 3.

Code 3. Scaffolding contents of ´server.js`.

```
const express = require('express');
const mongoose = require('mongoose');
const config = require('./config');

const app = express();

// connect to mongoDB using mongoose
mongoose.connect(config.mongoURI, { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log('MongoDB Connected'))
  .catch(err => console.log(err));

const PORT = process.env.PORT || 5000; // listen to port defaulted to
5000
app.listen(PORT, () => console.log(`Server running on port
${PORT}`));//start express server & console.log a message
```

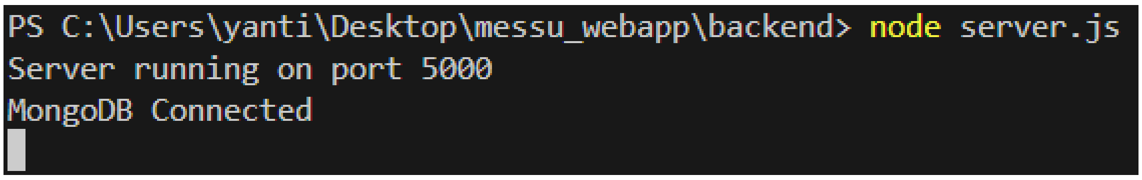
Since MongoDB service is run locally by Windows on a background, there is no need to launch the service manually, however, depending on developer’s OS configuration, there can be a need for a manual service start script. This can be done by preparing a script for Node.js to run that specifies configuration file’s location (Code 4).

Code 4. Creation of manual MongoDB service booting script

```
"scripts": {
  "start-mongodb": "mongod --config \"C:\\Program
Files\\MongoDB\\Server\\7.0\\bin\\mongod.cfg\""
},
```

This will allow user to launch service manually by calling for script in terminal with ´npm run start-mongodb`. Script will be added later on into a ´package.json` file that will get generated during project initialization and package installation.

Now connection to database can be established and backend booted with terminal command ´node server.js`, successful booting should result in terminal output shown in Figure 14.



```
PS C:\Users\yanti\Desktop\messu_webapp\backend> node server.js
Server running on port 5000
MongoDB Connected
```

Figure 14. Screen capture of successfully booted backend and database connection.

3.1.3 Database architecture

MongoDB is a document-oriented database, this means that it stores data in a format similar to JSON documents. Simple MongoDB databases typically consist of collections, documents, and fields. Each collection typically consists of multiple documents and each document of multiple fields. Each document within a collection can have a structure of its own.

Fields are instances of data within a document, which can be of various types such as numbers, arrays, or strings. For instance, collection “Cars” can have a document, which would contain fields “Model”, “Year” and “Color”. Within the same collection, another document could contain only “Year” and “Color” fields. This is a crude example of unstructured data that can be worked with using

MongoDB. Documents are not required to be of specific template to be held inside a collection, however if structure is needed, developers have the tools to enforce it via schemas or defining specific field data types.

For the messu_webapp project it was decided that starting with five collections will be sufficient.

Companies-collection is a repository for basic information about companies participating in the event. It will contain ´companyName` and ´companyID` fields.

Users-collection is going to contain essential user information such as ´firstName`, ´lastName`, ´emailAddress`, ´userPassword` and ´userID`.

Votes-collection is for managing votes, each document will record user's vote in a specific category. The ´userID`-field will establish a link to the user who cast the vote, while ´companyID` and ´category` are denoting vote recipient company and voting context, respectively.

Passes-collection is going to store pass information, where ´userID` will be linked to ´companyID`.

Lastly, Sweepstakes-collection will capture user's participation through ´userID` and ´emailAddress`.

This general architecture is going to form a foundation for messu_webapp, facilitate user-company interactions, sweepstakes participation as well as voting logic.

3.1.4 Models

"Models are higher-order constructors that take a schema and create an instance of a document equivalent to records in a relational database." (Sazib 2022) They are defining schema of said data and allow performing of CRUD operations with it.

Models include field constraints, data entity relationships as well as field data types. They serve as an additional layer between application and the database, so instead of directly interacting with MongoDB, the application interacts with the model itself and the model then handles CRUD operations internally.

Encapsulating logic related to data handling within these models enables reusability and modularity of codebase. Created models can be utilized across the whole application and even outside of it in different projects if they are built appropriately. An additional benefit of such modularity is ease of testing, since models can be tested in a vacuum, isolated from the rest of the application logic.

Firstly, to keep the project modular and organized, a new directory called ´models` will be created in backend root directory. ´models` will contain JavaScript files with schemas for each model. As an example, Code 5 is contained in ´messu_webapp/backend/models/user.js`.

Code 5. User model schema.

```
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true,
  },
  lastName: {
    type: String,
    required: true,
  },
  emailAddress: {
    type: String,
    required: true,
  },
  userPassword: {
    type: String,
    required: true,
  },
  userID: {
    type: String,
    required: true,
    unique: true,
  },
});

const User = mongoose.model("User", userSchema);

module.exports = User;
```

In similar fashion, schema for each model is created in corresponding JavaScript file. The schema will specify the structure of documents inside the related collection. After models are created, ´models` directory of backend looks like shown in Figure 15. Each JavaScript file contains a model schema with necessary data fields, types, and restrictions.

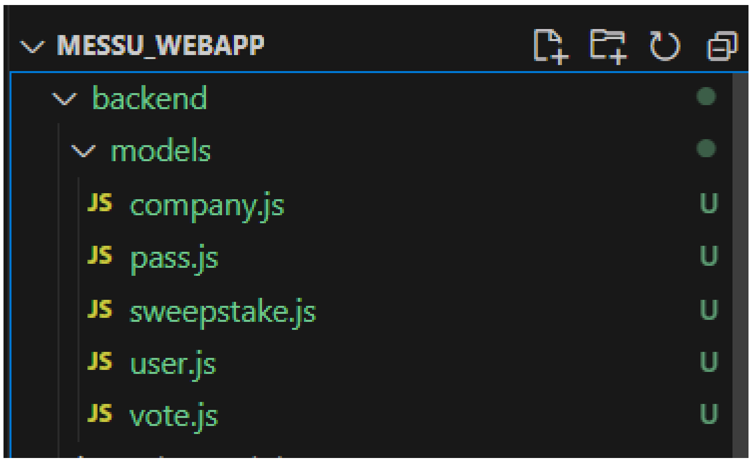


Figure 15. Screen capture of models directory in VSCode

3.1.5 Routing

In order to perform CRUD operations with MongoDB, API endpoints and routing need to be defined and created respectively.

API (Application Programming Interface) endpoint “is a digital location where an API receives requests about a specific resource on its server. In APIs, an endpoint is typically a uniform resource locator (URL) that provides the location of a resource on the server” (Juviler 2024). For instance, a user wants to take a closer look at Honda Civic on a car dealership’s website. By clicking on the picture of that car, the client makes a GET request to a server, let’s say ´GET /cars/honda_civic`. Server sends a response to the client displaying content that was requested. ´GET/cars/{car_name}` in this example is the API endpoint.

To maintain organization and modularity of the project, a new directory for routes should be created. ´messu_webapp/backend/routes` will contain JavaScript router files that are going to handle HTTP requests related to a specified model.

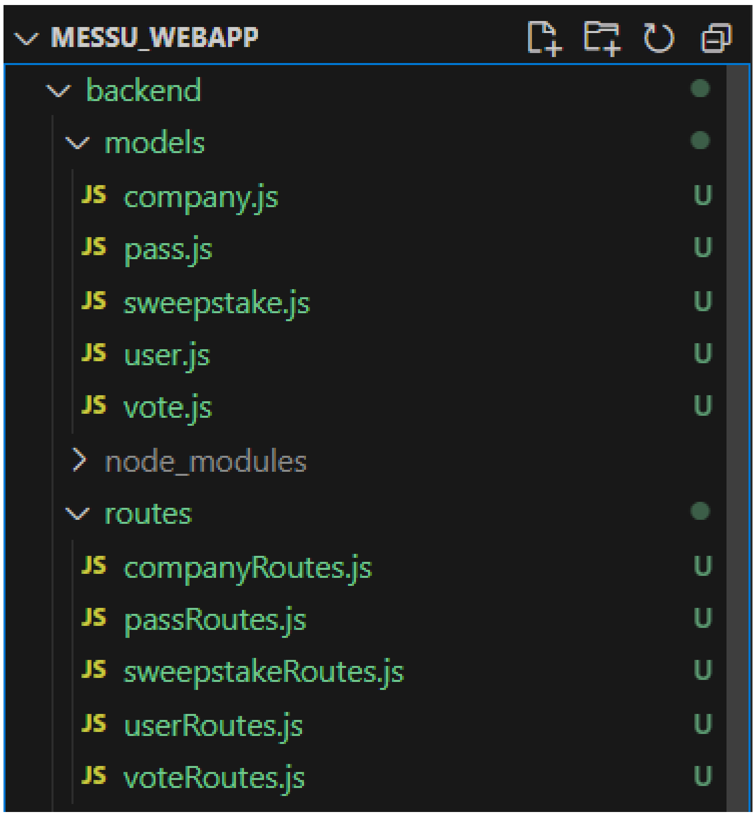
HTTP (Hypertext Transfer Protocol) request can be described as a message sent to a server from client, message contains a request for a specific action. HTTP requests are a foundation of client and server communication.

Router files are paramount for modularizing and organizing route-handling logic for any web application.

When handling logic is created and API endpoints are defined, router file can be exported and used by main application file, in this case ´server.js`.

It has been decided that currently the most useful routes would be ´router.get`, a route to get all documents in a collection, ´router.post` to create a new document and insert it into a collection, and ´router.delete` to delete a specific document from a collection. Document is specified by its unique ID, that MongoDB automatically assigns to each one of the documents inside any collection.

Code 6 can be viewed as an example of what a typical router file looks like. In a similar fashion, a router file for each existing model is created. After router files are created, backend structure starts to take shape of an actual project (Figure 16).



```
MESSU_WEBAPP
  backend
    models
      JS company.js              U
      JS pass.js                 U
      JS sweepstake.js           U
      JS user.js                 U
      JS vote.js                 U
    node_modules
    routes
      JS companyRoutes.js        U
      JS passRoutes.js           U
      JS sweepstakeRoutes.js     U
      JS userRoutes.js           U
      JS voteRoutes.js           U
```

Figure 16. Screen capture of backend root directory in VSCode.

Code 6. Contents of ´userRoutes.js`.

```
const express = require("express");
const router = express.Router();
const User = require("../models/user");

// route to get all users
router.get("/api/user", async (req, res) => {
  try {
    const users = await User.find();
    res.json(users);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

// route to create a new user
router.post("/api/user", async (req, res) => {
  const user = new User({
    userName: req.body.firstName,
    userLastName: req.body.lastName,
    emailAddress: req.body.emailAddress,
    userPassword: req.body.userPassword,
    userID: req.body.userID,
  });
  try {
    const newUser = await user.save();
    res.status(201).json(newUser);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});
// route to delete a user
router.delete("/api/user/:id", async (req, res) => {
  try {
    await User.findByIdAndDelete(req.params.id);
    res.json({ message: "User deleted" });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});
module.exports = router;
```

When routers are created and exported, they can be used by ´server.js` to perform CRUD operations. This is an essential step for manipulating data travelling between backend and client sides. Without routing, populating the database with temporary mock data would be unnecessarily convoluted and inefficient. Depending on request, ´server.js` uses one of the routes specified in a router file. Router files are imported into and required inside of ´server.js` (Code 7).

Code 7. Contents of ´server.js`.

```
const express = require("express");
const mongoose = require("mongoose");
const config = require("./config");
const app = express();
const companyRoutes = require("./routes/companyRoutes");
const passRoutes = require("./routes/passRoutes");
const sweepstakeRoutes = require("./routes/sweepstakeRoutes");
const userRoutes = require("./routes/userRoutes");
const voteRoutes = require("./routes/voteRoutes");

// middleware
app.use(express.json());

// company routes
app.use(companyRoutes);
// pass routes
app.use(passRoutes);
// sweepstake routes
app.use(sweepstakeRoutes);
// user routes
app.use(userRoutes);
// vote routes
app.use(voteRoutes);

// start the server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

// connect to mongoDB
mongoose
  .connect(config.mongoURI)
  .then(() => console.log("MongoDB Connected"))
  .catch((err) => console.log(err));
```

3.1.6 Seeding

Defining API endpoints and establishing routing infrastructure allows developers to manipulate the database.

In order to streamline creation of mock data for ease of future development, a seeding script can be created. Instead of manually creating a database, collections, documents, and fields, developer can use a single terminal command that will do it all automatically. Enabling this possibility is the next step for this project.

Mocking is the process of creating Mock objects that imitate the behavior of real objects in a controlled way. When Mocking, developers create a fake version of an external dependency, such as a database or API, and use it to test their code in isolation. (Raka 2023.) Mock-data is artificially generated, simulated, or manually entered data that exists for development and testing purposes. To develop software, developers often need to mimic real-world scenarios which are not possible to work with without mimicking production data. This is where mock data comes in handy.

To make a script, first ´seed.js` file needs to be added to the root directory of messu_webapp’s backend. This file will be called in the future with ´node seed.js`, which will command Node.js to compile the code inside of the file.

Firstly, required modules need to be imported. Mongoose, configuration file where connection URI and the name of database is specified, as well as previously created user schema. Mongoose then establishes a connection to MongoDB with ´mongoose.connect(config.mongoURI)` (Code 8).

Code 8. Declaring variables and establishing MongoDB connection.

```
const mongoose = require("mongoose");
const config = require("./config");
const User = require("./models/user");

// Connect to MongoDB
mongoose
  .connect(config.mongoURI)
  .then(async () => {
    console.log("Connected to MongoDB");
```

There is no need to run commands to create a database itself and fill it with collections.

Mongoose connects to MongoDB using the URI that was specified in ´config.js`, in this case to ´ mongodb://127.0.0.1:27017/messu_webapp`. If the database does not exist, MongoDB will create it.

Furthermore, after connection to the database is established, MongoDB will use variable defined in ´user.js` to insert documents into the collection corresponding to ´User` model. If the model does not exist, MongoDB will pluralize singular instance of that model and generate a collection automatically. This happens when ´User.insertMany(mockUsers);` is compiled. Code 9 is an example of one set of user documents that will be inserted into a database.

Code 9. Creation of user mock data.

```
const mockUsers = [
  {
    firstName: "Abby",
    lastName: "Abbot",
    emailAddress: "abby.abbot@email.com",
    userPassword: "strongpassword1",
    userID: "1",
  },
  {
    firstName: "Bobby",
    lastName: "Kotik",
    emailAddress: "bobby.kotik@email.com",
    userPassword: "strongpassword2",
    userID: "2",
  },
  {
    firstName: "Cobby",
    lastName: "Consty",
    emailAddress: "cobby.consty@email.com",
    userPassword: "strongpassword3",
    userID: "3",
  },
];
```

It is recommended to implement error handling mechanisms into seeding script, it can be handled with try/catch JavaScript structure (Code 10). Block would try inserting data into a database and catch possible incoming errors. In case of an error occurring, an error message is displayed in the terminal, connection to the database is closed and script is exited.

Code 10. Error handling via try/catch structure.

```
try {
  await User.insertMany(mockUsers);
  console.log("User data inserted successfully");
} catch (err) {
  console.error("Error inserting user data:", err);
  mongoose.connection.close();
  process.exit(1);
}
```

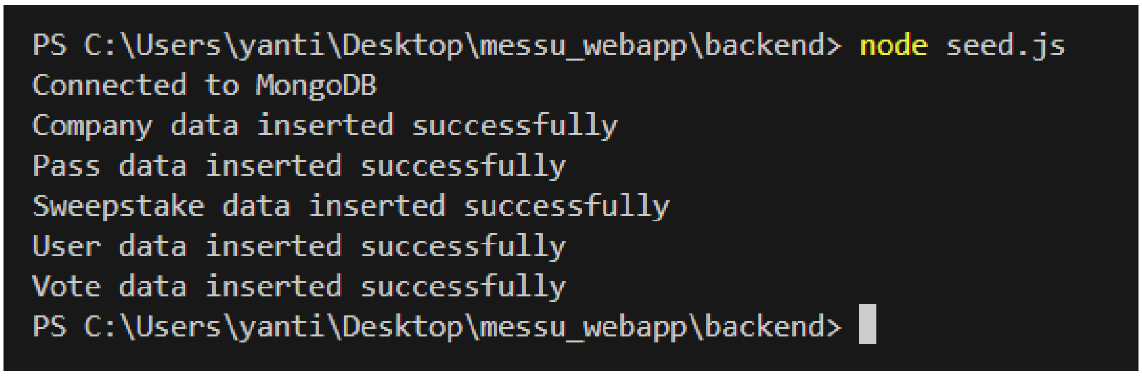
Similarly, all the collections are created and populated with mock data within ´seed.js`.

When database and collection are successfully created and filled with mock data, script closes connection to the database with ´mongoose.connection.close();` as shown in Code 11.

Code 11. Closing the connection to MongoDB after data is inserted.

```
    // Close the MongoDB connection
    mongoose.connection.close();
  })
  .catch((err) => {
    console.error("Error connecting to MongoDB:", err);
  });
```

After script file is complete, it can be called with ´node seed.js`. Successful seeding terminal output is shown in Figure 17. In case of an error occurring during the seeding process, previously created error handling would point at the exact spot the error took place.

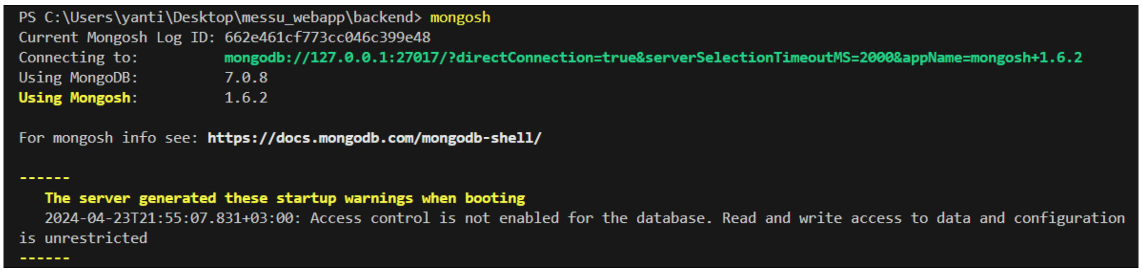


```
PS C:\Users\yanti\Desktop\messu_webapp\backend> node seed.js
Connected to MongoDB
Company data inserted successfully
Pass data inserted successfully
Sweepstake data inserted successfully
User data inserted successfully
Vote data inserted successfully
PS C:\Users\yanti\Desktop\messu_webapp\backend>
```

Figure 17. Screen capture of a successful seeding process.

To verify successful database creation either MongoDB shell or MongoDB Compass can be used. Verification via shell can be performed using appropriate terminal commands.

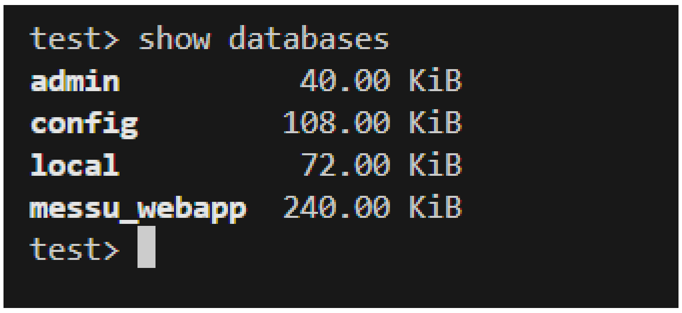
First a command ´mongosh` needs to be inputted in order to start the shell (Figure 18).



```
PS C:\Users\yanti\Desktop\messu_webapp\backend> mongosh
Current Mongosh Log ID: 662e461cf773cc046c399e48
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.6.2
Using MongoDB:          7.0.8
Using Mongosh:          1.6.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-04-23T21:55:07.831+03:00: Access control is not enabled for the database. Read and write access to data and configuration
is unrestricted
------
```

Figure 18. Screen capture of MongoDB Shell start.

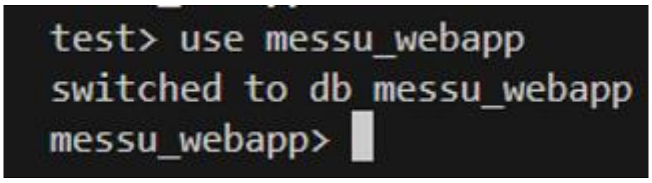
To view a full list of locally hosted databases, ´show databases` can be used in MongoDB Shell terminal. The resulting output is a list identical to the one that can be found in MongoDB Compass (Figure 19).



```
test> show databases
admin          40.00 KiB
config        108.00 KiB
local          72.00 KiB
messu_webapp  240.00 KiB
test>
```

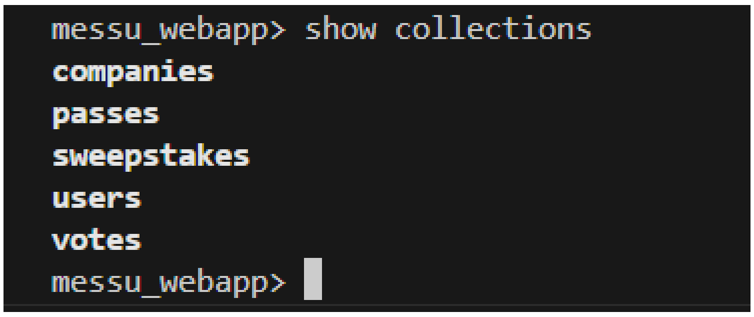
Figure 19. Screen capture of all local databases accessed via MongoDB Shell.

In order to switch to a specific database of interest, command ´use` is utilized in combination with ´messu_webapp`, which is the name of previously created database (Figure 20). Figure 21 demonstrates usage of command ´show collections` that outputs a list of all existing collections within the selected database. Lastly, to retrieve documents from a specific collection, command ´db.collection_name.find():` is used (Figure 22).



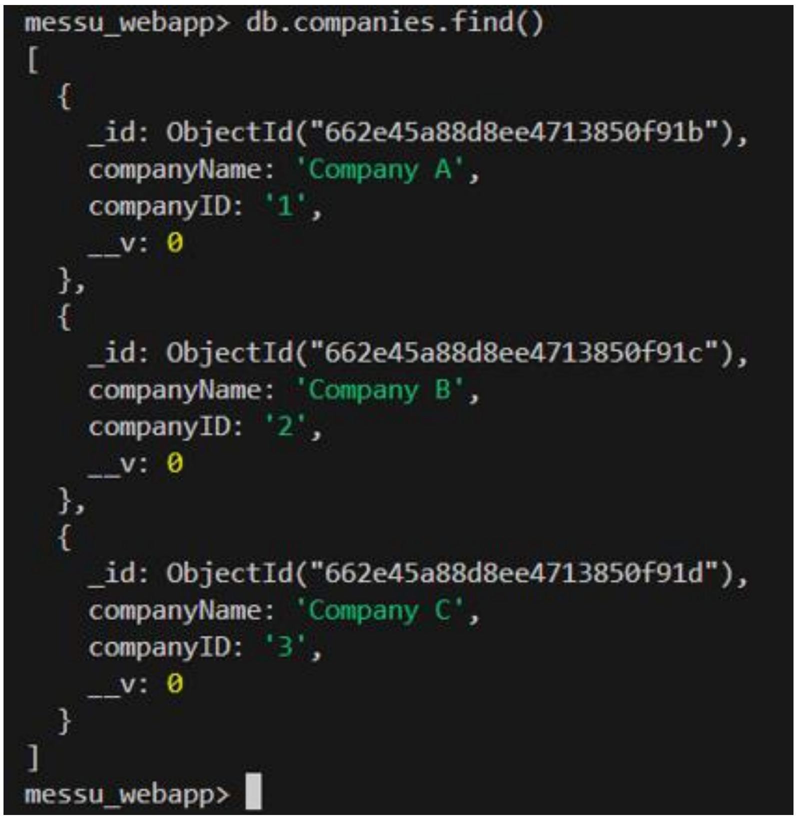
```
test> use messu_webapp
switched to db messu_webapp
messu_webapp>
```

Figure 20. Screen capture of successful switching to ´messu_webapp` database.



```
messu_webapp> show collections
companies
passes
sweepstakes
users
votes
messu_webapp>
```

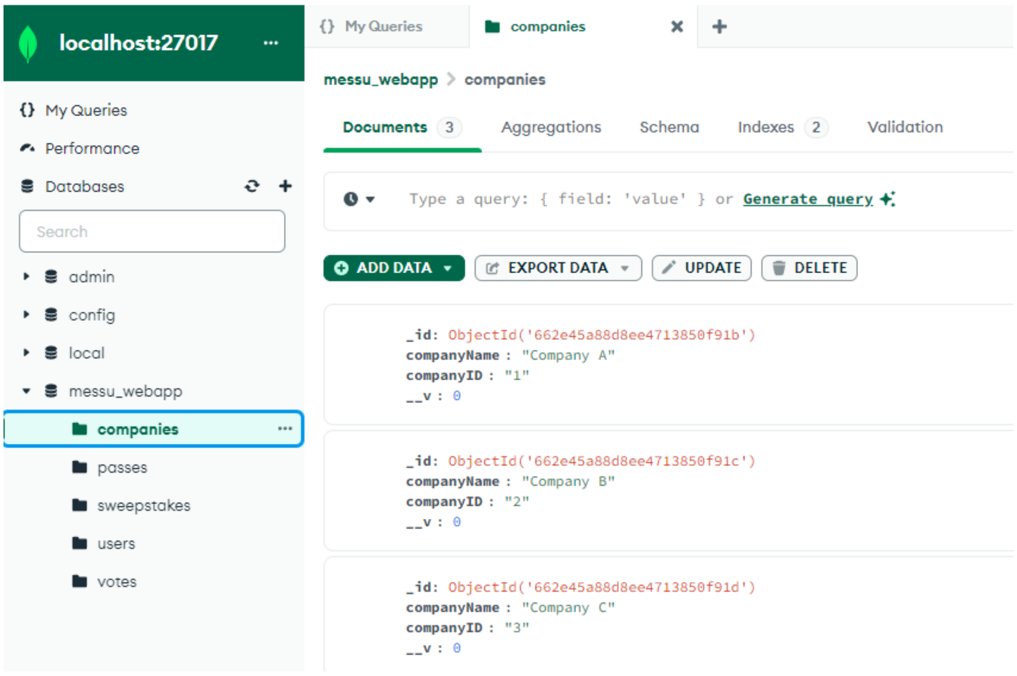
Figure 21. Screen capture of all the collections inside messu_webapp accessed via MongoDB Shell.



```
messu_webapp> db.companies.find()
[
  {
    _id: ObjectId("662e45a88d8ee4713850f91b"),
    companyName: 'Company A',
    companyID: '1',
    __v: 0
  },
  {
    _id: ObjectId("662e45a88d8ee4713850f91c"),
    companyName: 'Company B',
    companyID: '2',
    __v: 0
  },
  {
    _id: ObjectId("662e45a88d8ee4713850f91d"),
    companyName: 'Company C',
    companyID: '3',
    __v: 0
  }
]
messu_webapp>
```

Figure 22. Screen capture of ´companies` collection's contents.

Alternatively, contents of the newly created database can be inspected using MongoDB Compass GUI by simply navigating to database itself and desired collection respectively. Using MongoDB Compass is less convoluted and more user friendly compared to navigating via terminal (Figure 23).

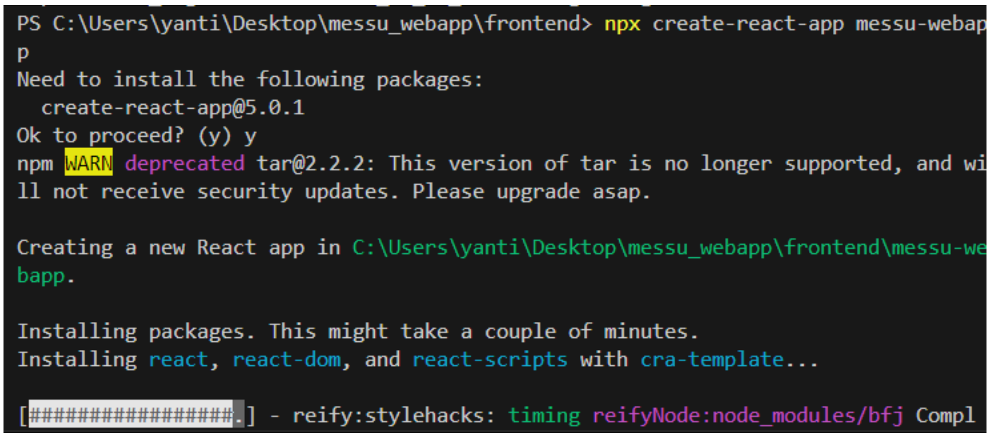


```
_id: ObjectId('662e45a88d8ee4713850f91b')
companyName : "Company A"
companyID : "1"
__v : 0

_id: ObjectId('662e45a88d8ee4713850f91c')
companyName : "Company B"
companyID : "2"
__v : 0

_id: ObjectId('662e45a88d8ee4713850f91d')
companyName : "Company C"
companyID : "3"
__v : 0
```

Figure 23. Screen capture of "companies" collection accessed via MongoDB Compass.

3.2 Frontend and README

´frontend` directory will serve as a base for project’s frontend. Since React.js was chosen as a framework, appropriate command to create and initialize frontend project and populate it with scaffolding-like assets is in Figure 24.



```
PS C:\Users\yanti\Desktop\messu_webapp\frontend> npx create-react-app messu-webap
p
Need to install the following packages:
  create-react-app@5.0.1
Ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and wi
ll not receive security updates. Please upgrade asap.

Creating a new React app in C:\Users\yanti\Desktop\messu_webapp\frontend\messu-we
bapp.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[#################.] - reify:stylehacks: timing reifyNode:node_modules/bfj Compl
```

Figure 24. Screen capture of frontend initialization terminal output.

´npx create-react-app` serves same function as backend’s ´npm init -y`. It initializes a new React frontend project and generates a ´package.json` file. Node modules dependencies to which are specified in that file can be installed with ´npm install`.

Next step is installing prettier.io, a formatting tool for JavaScript specified in the style guide. Prettier.io parses through the whole project root and removes unnecessary symbols, fixes indentation and spacing issues and enhances readability and consistency of the code. It can be done after navigating to root directory in terminal and running ´npm install –save-dev –save-exact prettier`.

To use a command ´npm run format` specified in style guide, a script needs to be added to a messu_webapp’s root ´package.json` file (Code 12). After updating ´package.json`, project can be formatted using ´npm run format`.

Code 12. Updated ´package.json` file.

```
{
  "devDependencies": {
    "@babel/plugin-proposal-private-property-in-object": "^7.21.11",
    "prettier": "3.2.5"
  },
  "scripts": {
    "format": "prettier --write ."
  }
}
```

The final step in setting up frontend is installing and setting up Tailwind CSS, a framework for creating custom designs without having to write custom CSS. To do that, first Tailwind CSS needs to be installed according to installation manual for Create React App, which is a part of Tailwind CSS documentation. Since it's a frontend utility, there is no need to install it to the root, which would add the dependencies to root ´package.json`. Instead, appropriate way to install it is navigating to the frontend directory and running ´npm install -D tailwindcss@npm:@tailwindcss/postcss7-compat postcss@^7 autoprefixer@^9 `.

PostCSS is a modular tool that allows transforming CSS with JavaScript plugins. Autoprefixer is a PostCSS plugin that is adding vendor prefixes automatically.

Since Create React App does not currently support PostCSS 8, it's version needs to be specified to a PostCSS 7 compatibility build. In order to re-configure PostCSS, CRACO is also required, because Create React App does not have a native re-configuration option. CRACO can be installed using ´npm install @craco/craco` in frontend directory terminal.

After installation, frontend's ´package.json` needs to be updated to use ´craco` scripts instead of ´react-scripts` with the only exception being ´eject` (Code 13).

Code 13. Updated frontend's ´package.json` file.

```
"scripts": {
  "start": "craco start",
  "build": "craco build",
  "test": "craco test",
  "eject": "react-scripts eject"
},
```

Next, ´craco.config.js` configuration file needs to be created in the frontend root directory. ´tailwindcss` as well as ´autoprefixer` need to be added as PostCSS plugins (Code 14).

Code 14. Creation of ´craco.config.js`.

```
module.exports = {
  style: {
    postcss: {
      plugins: [
        require('tailwindcss'),
        require('autoprefixer'),
      ],
    },
  },
}
```

After CRACO configuration file is created, another configuration file needs to be generated, a ´tailwind.config.js`. It can be done with ´npx tailwindcss-cli@latest init` command.

In the generated file, option ´purge` needs to be configured with path to all of the frontend components (Code 15). This is done in order to enable Tailwind to get rid of unused styles, this process is referred to as “tree shaking”.

Code 15. Creation of tree shaking access.

```
purge: ['./src/**/*.{js,jsx,ts,tsx}', './public/index.html'],
```

Finally, contents of prefabricated ´index.css` file that were generated by Create React App need to be replaced with new directive to include Tailwind’s own styles (Code 16).

Code 16. Including Tailwind's own styles in ´index.css`.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Now to confirm that tailwindcss is all set up, it should be tested.

3.2.1 ReactDOM and JSX

Create React App setup’s connection between ´index.js` and ´index.html` is established with entry points.

HTML entry point, ´index.html` file is a file that gets loaded when React application is opened.

Inside of this file, typically a <div> element is found, that contains an id attribute set to ´root` (Code 17).

Code 17. HTML entry point's placeholder element.

```
<div id="root"></div>
```

This is where React application is going to be mounted.

JaveScript entry point, ´index.js` file serves as an entry point for React application’s JavaScript code. Typically, in ´index.js` ReactDOM is used to render a React component into DOM element using id of ´root`.

Code 18 is ´src/App.js` file, a main React component, which will be rendered by ReactDOM.

Code 18. Main React component ´App.js`.

```
import "./App.css";

export default function App() {
  return (
    <div>
        <h1>Example text</h1>
    </div>
  );
}
```

Code 19 is ´public/index.html`example file, an entry point for the application. <div> element with an ´id` attribute set to ´"root"` behaves as a placeholder for ReactDOM to render into.

Code 19. Entry point for ReactDOM to render into.

```html
<!doctype html>

<html lang="en">
  <body>
    <div id="root"></div>
  </body>
</html>
```

Lastly, ´src/index.js` file uses ´ReactDom.createRoot` and ´root.render()` to render component (Code 20).

Code 20. Rendering the component into a ´root` placeholder found in HTML entry point.

```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

To simplify even further, ´index.js` selects root element defined in ´index.html`. ReactDOM then renders the main React component into that element. Essentially, connecting React application to the HTML entry point.

To test whether or not tailwindcss took effect, a small adjustment can be made into a main component file, for example dramatically inflating text size. It can be done using JSX syntax (Code 21).

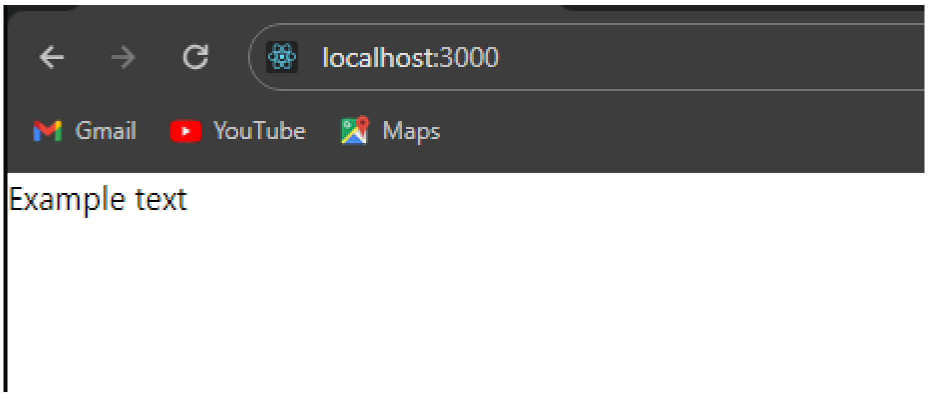
Code 21. JSX snippet showing Tailwind styles application to "Example text".

```
import "./App.css";

export default function App() {
  return (
    <div>
        <h1 className="text-5xl">Example text</h1>
    </div>
  );
}
```

JavaScript XML or JSX is an extension of the JavaScript language syntax. The React library is an extension to write XML-like code for elements and components. (Geeksforgeeks 2023.) It is commonly used in combination with React.

It allows developers to build UI faster and with less directory bloating. JSX makes it possible to combine HTML structure and JavaScript behavior of components in a single file, practically eliminating the need to separate HTML and CSS files for each component. This makes code easier to maintain, makes overall application structure easier to grasp, which can and will be a considerable time saver in more complex applications.

Now that Tailwind CSS can be tested in action, ´npm run start` in frontend directory is run and browser window at localhost:3000 is open. However, Tailwind CSS does not function as expected (Figure 25).



Example text

Figure 25. Screen capture of a browser window with malfunctioning Tailwind CSS

3.3 Trouble and troubleshooting

Tailwind CSS simply refused to work in combination with React Create App, regardless of its documentation specifications and developers' guides.

The first step to solve potential problems is to check for console errors and warnings, most advanced software has integrated error handling pathways similar to ones that were created for seeding script.

With the help of console error messages most often a solution to a problem can be found, however in this specific case no error messages were displayed. Warnings, that are in most cases not relevant considering the vast number of packages, were not related to Tailwind CSS whatsoever.

The first theory was the inappropriate installation of node module packages. This can be tackled with simply deleting ´node_modules` directory from the frontend directory and re-installing modules again with ´npm install`. Doing that did not solve the issue.

Second theory was misconfiguration of ´tailwind.config.js` or misconfiguration of PostCSS configuration files, as well as missing ´package.json` dependencies. After confirming that configuration files for both Tailwind CSS and PostCSS are built correctly and ´package.json` is in order, Tailwind CSS continued to malfunction.

The third theory was an inappropriate build process, so frontend folder got deleted and whole process from creating a frontend directory to setting up Tailwind CSS configuration was repeated from scratch to yet again no avail.

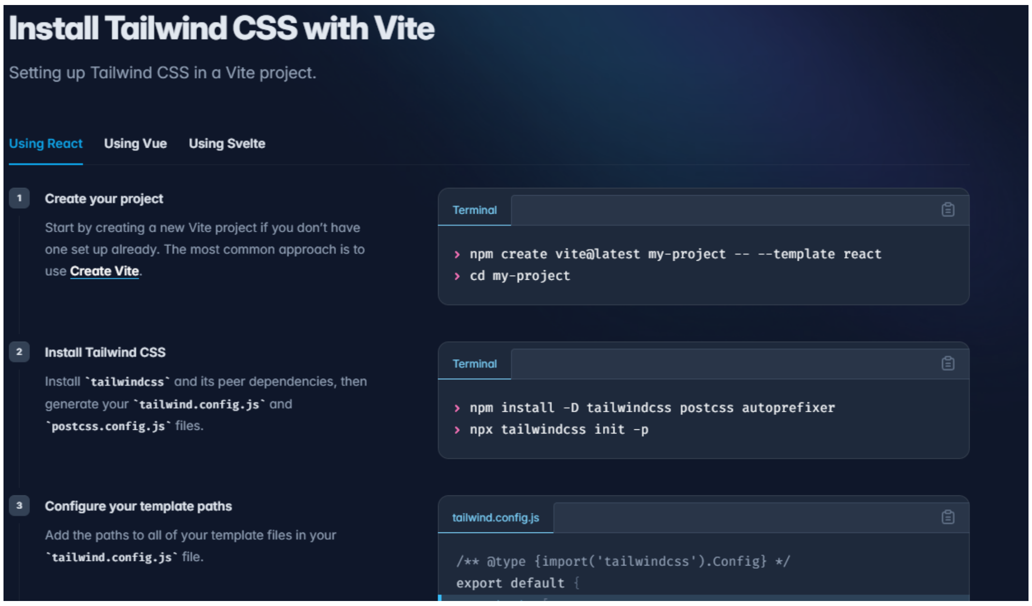
The amount of time spent on reading forum messages, watching video tutorials, consulting other developers about this issue, and testing proposed solutions amassed to approximately 30 hours, which stalled the development process noticeably.

An executive decision has then been made to search for smoother integration software or method for Tailwind CSS.

Out of many options one stood as the most appropriate for the project's needs – Vite, a popular tooling software that enhances and speeds up the development process.

For instance, "Vite improves the dev server start time by first dividing the modules in an application into two categories: dependencies and source code" (Vite 2024).

Using Tailwind CSS with Vite installation manual, frontend project directory was rebuilt, configuration files generated and edited in accordance with the manual (Figure 26).



Install Tailwind CSS with Vite

Setting up Tailwind CSS in a Vite project.

Using React | Using Vue | Using Svelte

1 Create your project

Start by creating a new Vite project if you don't have one set up already. The most common approach is to use Create Vite.

Terminal

```
> npm create vite@latest my-project -- --template react
> cd my-project
```

2 Install Tailwind CSS

Install `tailwindcss` and its peer dependencies, then generate your `tailwind.config.js` and `postcss.config.js` files.

Terminal

```
> npm install -D tailwindcss postcss autoprefixer
> npx tailwindcss init -p
```

3 Configure your template paths

Add the paths to all of your template files in your `tailwind.config.js` file.

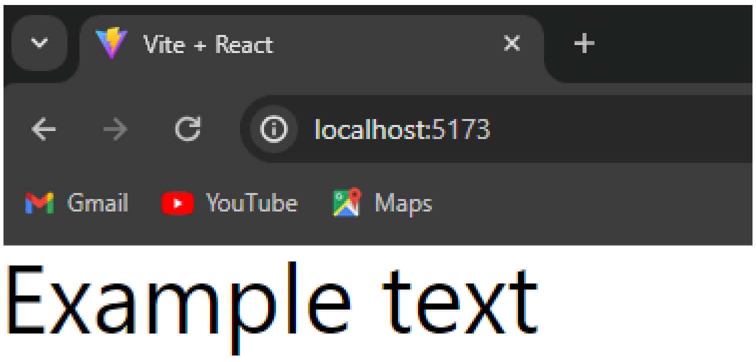
tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
export default {
```

Figure 26. Screen capture of Tailwind CSS installation and setup manual

Installation and setup process took less than 10 minutes, frontend scaffolding was re-initialized using Vite and dependencies installed right after.

JSX syntax that was used during testing of Tailwind CSS with Create React App was reused, frontend project booted with new command, ´npm run dev`, which is essentially a Vite substitute for a CRACO script. Tailwind CSS was finally functioning as intended (Figure 27).



Example text

Figure 27. Screen capture of a browser window with functioning Tailwind CSS

This should and was perceived as a painful learning experience, rather than an opportunity to bash oneself for spending so much time trying to optimize something that should not have existed in the first place.

3.4 README

Typically, “README.md (.md stands for Markdown format) file is often the first file that the users read. It is a text file that contains information for the user about the software, project, code, or game, or it might contain instructions, help, or details about the patches or updates” (Great Learning Team 2022). There can be multiple README files in different project directories, depending on scale and project team size.

Both backend and frontend projects are ready for further development, last thing before committing changes to GitHub is updating README file with instructions for deploying the project locally on another machine.

README serves as a tutorial and introduction to project’s essentials, but for purposes of thesis README can be considered an abstract of the work done on the project up until first Git commit.

A rather sophisticated process of setting up the environment and optimizing its deployment locally from any other system is summarized and condensed into a single document. Contents of the README file can be viewed below.

```
# messu_webapp

## Description

Digital collection and voting management system.

## Software installation

1. MongoDB:

   - download and MongoDB Community Server from
´https://www.mongodb.com/try/download/community`
   - during installation make sure to check MongoDB Compass (GUI)
installation checkbox
   - to insure that installation was performed correctly, run ´mongosh`
in command prompt, you should see at least two lines:
   - ´Using MongoDB: <version>`
   - ´Using Mongosh: <version>`
   - if not the case, repeat the installation process

2. MongoDB Shell:

   - download and extract MongoDB Shell zip archive from
´https://www.mongodb.com/try/download/shell`
   - use either MongoDB Shell or MongoDB Compass to interact with the
database

3. Node.js:

   - download Node.js from ´https://nodejs.org/en/download`
   - install Node.js runtime environment

4. Editor:

   - download and install preferred code editor, for instance VSCode from
´https://code.visualstudio.com/download`
   - open the project's root in preferred editor
```

```
## Backend

1. navigate to directory via terminal ´cd backend`
2. run ´npm install mongoose` for integration with MongoDB
3. run ´npm install` to install necessary backend dependencies
4. populate the database with mock data using ´node seed.js`
5. start backend project using ´node server.js`

## Frontend

1. navigate to directory via terminal ´cd frontend`
2. run ´npm install` to install necessary backend dependencies
3. run ´npm run dev` to start up frontend
4. o + Enter to open browser window at localhost

## Troubleshooting

- missing module -> ´npm install`
- failure to connect to database -> check MongoDB Server service, needs
to be running (Win + R -> services.msc on Wndows OS)

## License

MIT License

## Authors

- Yan Zborovskij (Timoshkin)

## Software

Programming languages: JavaScript, HTML, CSS
Frameworks: Vite
Libraries: Express.js, Mongoose.js
Development tools: Visual Studio Code, Git, GitHub
Database software: MongoDB, MongoDB Compass, MongoDB Shell
Additional tools or services: npm, Node.js
```

3.5 First Git commit

The ´git commit` command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as “safe” versions of a project—Git will never change them unless you explicitly ask it to. (Atlassian

2024.) In essence, committing to Git is recording changes made to a project and storing these changes in an online Git repository.

After changes have been committed, the project can be pulled from repository to any other machine locally, which makes it possible for developers to switch to their own development branch and work on the project individually.

In version control systems such as Git, branches are parallel lines of program's development. Master branch or main branch serves as a base for "cloning" the project, which allows independent development without affecting main codebase. Simply put, developers pull the main branch to their local machine, switch to a development branch of their own, add new features or even re-write the whole project from scratch without affecting the main branch. When a new feature is ready, developers can merge their development branch back into the main branch by creating a pull request and solving potential merging conflicts. After changes are merged into the main branch, the cycle continues until the project's scope and goals are achieved.

Pull requests are used to review code on branches before it reaches master. Code review is also one of the most difficult and time-consuming parts of the software development process, often requiring experienced team members to spend time reading, thinking, evaluating, and responding to implementations of new features or systems. (Riosa 2019.)

Pull request is a request to merge one branch into another. Typically to merge a feature branch into the master branch in a Git repository. Pull requests get reviewed by other developers, get feedback, and collaborate on code changes if necessary. Pull request and reviews are a vital safety procedure that ensures the integrity of the codebase, prevents master branch from getting compromised, as well as keeps developers in bigger teams on track with development process.

In order to commit changes into online repository, first a small update needs to be done to ´.gitignore` file. This file allows Git to ignore some changes done locally (such as packages that get installed when developer is running ´npm install` for instance). There is no need to artificially inflate repository size with thousands of packages since they can be installed locally during local deployment. Virtual Studio Code or VSCode, a popular code editor, updates ´.gitignore` automatically.

After ´.gitignore` is updated, changes need to be staged. This either can be done with built-in VSCode features or using a terminal. Since different developers use different editors, for sake of demonstration first commit will be done using terminal.

Firstly, depending on where work is done and changes made, developers would need to navigate to the according directory. Current changes have been made to both frontend and backend, so the best approach to stage all the changes is to navigate to root directory.

After navigating to root, a command ´git add .` needs to be run in order for changes to be staged. Next step is committing staged changes, which can be done with ´git commit -m “Developer’s message”. According to the style guide, commit messages need to follow Conventional Commits- guidelines. In this case, changes were made to build process, auxiliary tools, libraries, so appropriate developer’s message according to guidelines should be for example:

´feat: Integrate Tailwind CSS for styling`

´build: add vite.config.js`

Additional changes were made to backend processing and frontend infrastructure, these changes can be referred to as either ´refactor`, ´maintenance` or ´build`.

For instance:

´feat: Create seed script to populate database with mock data`

´refactor: Improve pass model schema`

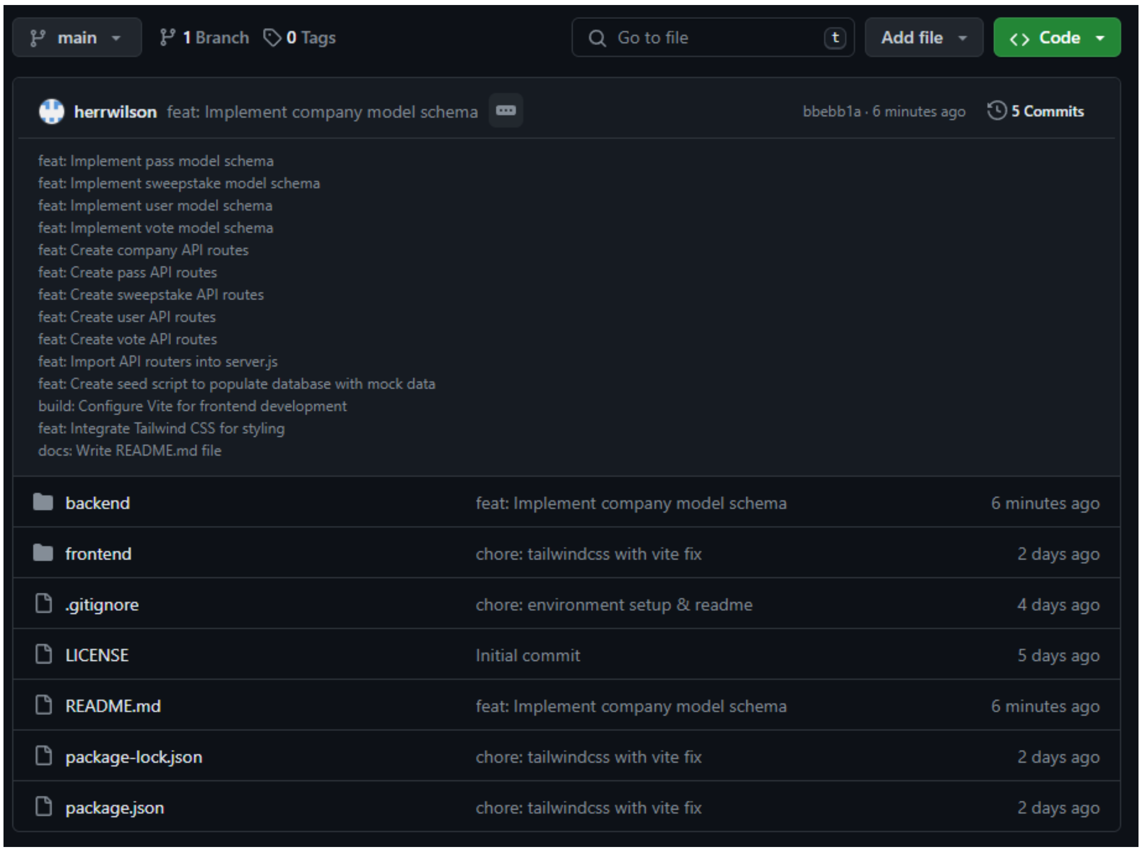
´maintenance: Reorganize API endpoints`

´build: add vite.config.js`

Lastly, ´style: format`, since ´npm run format` script was used in order for prettier.io to format project's codebase and ´docs: Write README.md file` to address newly created documentation.

When commit messages are written in detail, ´git push` command is used to push committed changes from local repository to a remote one, in this case ´messu_webapp` stored in GitHub.

After successfully pushing changes to the main branch into an online repository, developers can clone the main branch locally, initialize the project according to instructions provided in ´README.md`, switch to their own development branch and start working on the project. Git will neatly organize and show commit messages (Figure 28).



herrwilson feat: Implement company model schema bbebb1a · 6 minutes ago

```
feat: Implement pass model schema
feat: Implement sweepstake model schema
feat: Implement user model schema
feat: Implement vote model schema
feat: Create company API routes
feat: Create pass API routes
feat: Create sweepstake API routes
feat: Create user API routes
feat: Create vote API routes
feat: Import API routers into server.js
feat: Create seed script to populate database with mock data
build: Configure Vite for frontend development
feat: Integrate Tailwind CSS for styling
docs: Write README.md file
```

backend	feat: Implement company model schema	6 minutes ago
frontend	chore: tailwindcss with vite fix	2 days ago
.gitignore	chore: environment setup & readme	4 days ago
LICENSE	Initial commit	5 days ago
README.md	feat: Implement company model schema	6 minutes ago
package-lock.json	chore: tailwindcss with vite fix	2 days ago
package.json	chore: tailwindcss with vite fix	2 days ago

Figure 28. Screen capture of Git UI containing commit messages

4 Creation of prototype’s first page

Project’s central feature is the stamp pass; therefore, it has been chosen to be the first addition to the prototype. Documenting stamp pass prototype page’s creation will illustrate the process of creating backend logic, implementing frontend design as well as configuring asynchronous communication between client and server.

4.1 Frontend setup

Process of setting up frontend is started with creation of ´passForm.jsx` component file into a new frontend ´components` directory. Component file shall contain definition of functional ´PassForm` component itself, as well as shall include change handling and placeholders for handling change and submission events.

Code 22 shows defining functional component ´PassForm`, initializing ´userID` and ´companyID` fields with empty strings. Following Code 23 demonstrates creation of an event handler that will update ´formData` according to input values.

Code 22. Defining ´PassForm` component and initializing fields.

```
const PassForm = () => {
  const [formData, setFormData] = useState({
    userID: '',
    companyID: ''
  });
```

Code 23. Event handler creation.

```
const handleChange = (e) => {
  setFormData({
    ...formData,
    [e.target.name]: e.target.value});};
```

Creating a placeholder event handler for pass submission (Code 24). ´e.preventDefault();` will prevent default form submission behavior, which in HTML typically triggers a page reload or redirecting to a new URL. This behavior is often not desirable and preventing it allows gaining control over what happens after form is submitted.

Code 24. Creation of event handler placeholder for pass submission.

```
const handleSubmit = (e) => {
  e.preventDefault();
  // placeholder for submission handling
  console.log(formData);
};
```

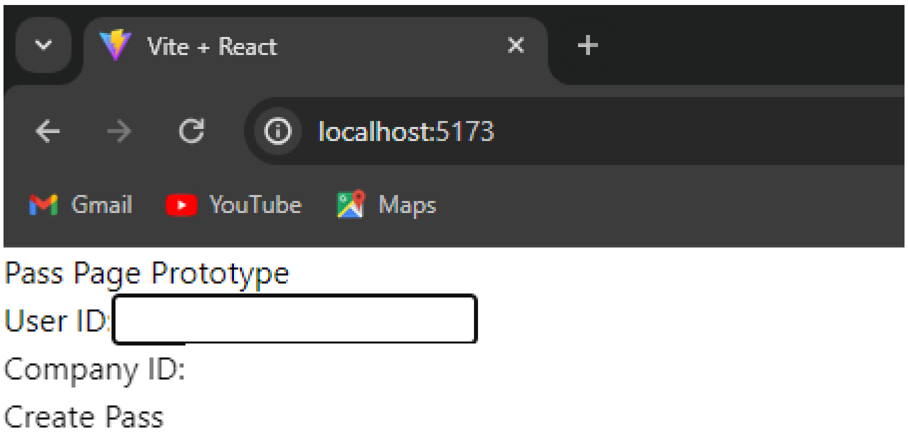
After the component has been defined and initialized and its event handlers created, component can be rendered, and its event handler functions can be called using JSX syntax. The expected return is demonstrated in Code 25.

´PassForm` can be either imported and rendered directly in ´main.jsx` or use main component ´app.jsx` as a “middleware”.

Implemented frontend changes are displayed in real time at localhost, Figure 29 depicts browser window opened at ´localhost:5173` that contains previously created pass form.

Code 25. Creation of component rendering function.

```
return (

 <form onSubmit={handleSubmit}>
   {/*userID input field*/}
   <div>
     <label htmlFor="userID">User ID:</label>
     <input
       type="text"
       id="userID"
       name="userID"
       value={formData.userID}
       onChange={handleChange}
     />
   </div>
   {/* companyID input field */}
   <div>
     <label htmlFor="companyID">Company ID:</label>
     <input
       type="text"
       id="companyID"
       name="companyID"
       value={formData.companyID}
       onChange={handleChange}
     />
   </div>
   {/* submit button here */}
   <button type="submit">Create Pass</button>
 </form>
```



Pass Page Prototype
User ID:
Company ID:
Create Pass

Figure 29. Screen capture of browser window with new component render

To make HTTP requests from frontend, a convenient library ´Axios` can be used. Axios is a fairly popular third-party library commonly used instead of the built-in Fetch API. It can be installed with ´npm install axios`.

4.2 Backend setup

Only a few changes need to be made to the backend infrastructure. Namely, a new pass creation method needs to be added to the pass router file ´passRoutes.js` (Code 26). Updated block can be used to both run seeding script locally in the backend as well as receive POST requests from client.

Code 26. Updating ´passRoutes.js`.

```
router.post("/api/pass", async (req, res) => {
  try {
    const { userID, companyID } = req.body;
    const newPass = await Pass.create({ userID, companyID });
    res.status(201).json(newPass);
  } catch (err) {
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

To avoid potential CORS errors, ´npm install cors` is run. CORS (Cross-origin Resource Sharing) errors happen when a webpage makes a request to a different domain than the one that served the page, and the server responds with an HTTP error because the “Origin” header in the request is not allowed by the server’s CORS configuration. (Maldonado 2023.) For instance, if POST request is made from PORT 5000 to PORT 3000, chances are the request will result in a 404-Error.

In addition, body-parser middleware that was installed as one of Express dependencies will be useful. Both are required inside of ´server.js`, are used as a middleware (Code 27).

Code 27. Addition of body-parser and CORS middleware.

```
const bodyParser = require("body-parser");
const cors = require("cors");

// middleware
app.use(express.json());
app.use(bodyParser.json());
app.use(cors());
```

4.3 Connecting frontend and backend

Previously unattained section of event handling logic in ´postForm.jsx` gets completed by adding an ´axios.post()` method, which sends asynchronous POST request to a specified URL and contains ´formData`, a variable that holds the data being sent in the request body. When a request is sent, the backend responds with either requested data or an error message specified in error handling or a default console error message (Code 28).

Code 28. Updating ´postForm.jsx` with new CRUD method.

```
 const handleSubmit = async (e) => {

  e.preventDefault();
  try {
  const response = await axios.post('http://localhost:5000/api/pass',
formData);
console.log(response.data);
} catch (err) {
  console.error(err);
}
};
```

4.4 Displaying and deleting data

With frontend and backend connected successfully, the next phase involves implementing functionality to display submitted passes and enable users to delete them as deemed necessary. It is a fundamental and vital functionality in a prototype.

This section outlines the steps to achieve this goal, focusing on frontend-to-backend communication and user interaction.

4.3.1 Displaying passes

Backend routing and models previously created do not need to be changed, changes mainly concern the project's frontend that will make requests to the server.

To display the submitted passes on the frontend, new ´PassList` component is utilized. This component fetches pass data from the backend upon mounting using ´useEffect` and renders each pass in the list.

´useEffect` and ´useState` are React features referred to as hooks. Hooks allow interacting with functional components without the need to write a class. Classes are traditional JavaScript constructs that are utilized for managing state methods. How it can be implemented in the frontend is shown below. In Code 29, a component state is created to store submitted passes.

Code 29. Creation of component state.

```
const PassList = ({ onDeletePass }) => {
  const [passes, setPasses] = useState([]);
```

After that, Code 30 demonstrates the creation of a React hook ´useEffect` initiates a GET request in order to fetch pass data from the backend API. Requested data will be stored in ´passes` state.

Code 30. Creation of a React hook.

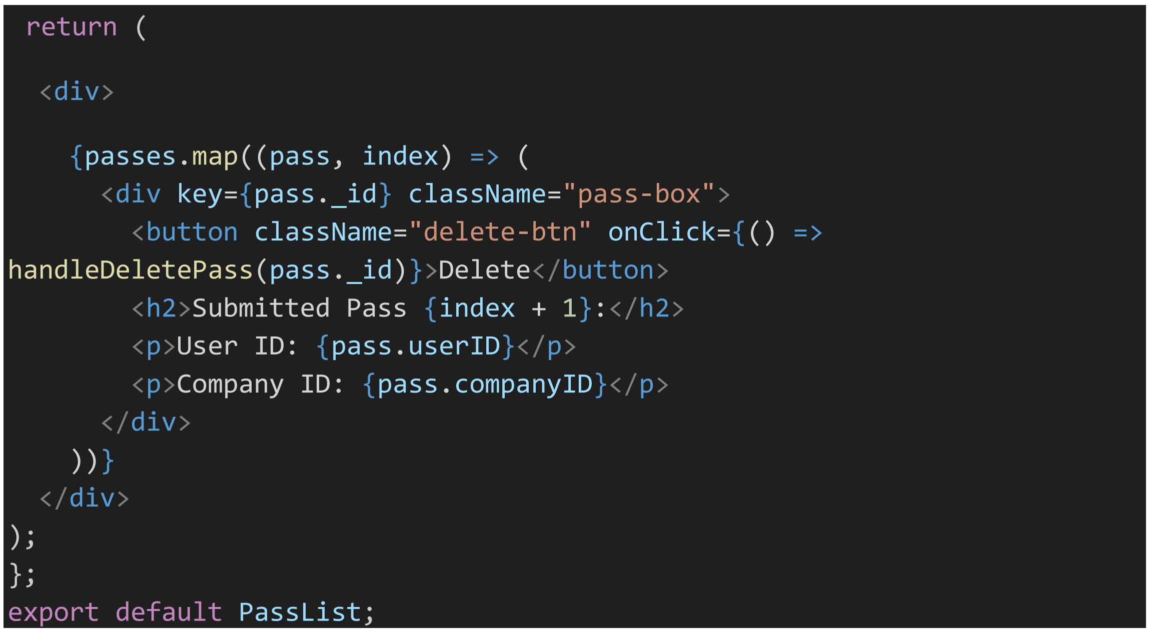
```
useEffect(() => {
  // fetch passes from backend when component mounts
  axios.get('http://localhost:5000/api/pass')
    .then((response) => {
      setPasses(response.data);
    })
    .catch((err) => {
      console.error(err);
    });
}, []); // empty dependency array to ensure effect runs only once
```

Similar to the ´passForm.jsx` event handlers, a new event handler called ´handleDeletePass` is created for ´passList.jsx` (Code 31). The handler will be sending an HTTP DELETE request to the backend API. If successful – deleted pass is removed from ´passes` array. ´passId´ is a unique identifier that MongoDB assigns to each new database entry. To dynamically render submitted passes, a ´map` method can be utilized (Code 32).

Code 31. Creation of DELETE event handler.

```
const handleDeletePass = (passId) => {
  // make DELETE request to backend to delete pass
  axios.delete(`http://localhost:5000/api/pass/${passId}`)
    .then((response) => {
      // remove deleted pass from state
      setPasses(passes.filter(pass => pass._id !== passId));
    })
    .catch((err) => {
      console.error(err);
    });
};
```

Code 32. Creation of dynamic data rendering using mapping method.



```
 return (

  <div>

    {passes.map((pass, index) => (
      <div key={pass._id} className="pass-box">
        <button className="delete-btn" onClick={() =>
handleDeletePass(pass._id)}>Delete</button>
        <h2>Submitted Pass {index + 1}:</h2>
        <p>User ID: {pass.userID}</p>
        <p>Company ID: {pass.companyID}</p>
      </div>
    ))}
  </div>
);
};
export default PassList;
```

´app.jsx`, parent component that deals with submission, display and deletion of passes needs to be updated to handle and pass down relevant functions to its child components (Code 33).

Code 33. Updated ´app.jsx` parent component.

```
const App = () => {
  // state to store submitted passes
  const [submittedPasses, setSubmittedPasses] = useState([]);
  // function to handle pass submission
  const handlePassSubmit = (formData) => {
    setSubmittedPasses([...submittedPasses, formData]);
  };
  // function to handle pass deletion
  const handleDeletePass = (index) => {
    const updatedPasses = submittedPasses.filter((pass, i) => i !==
index);
    setSubmittedPasses(updatedPasses);
  };

  // render PassForm and PassList components
  return (
    <div>
      <PassForm onPassSubmit={handlePassSubmit} />
      <PassList passes={submittedPasses} onDeletePass={handleDeletePass}
/>
    </div>
  );
};

export default App;
```

In Code 34 ´App` component is getting rendered in ´main.jsx` via previously created logic.

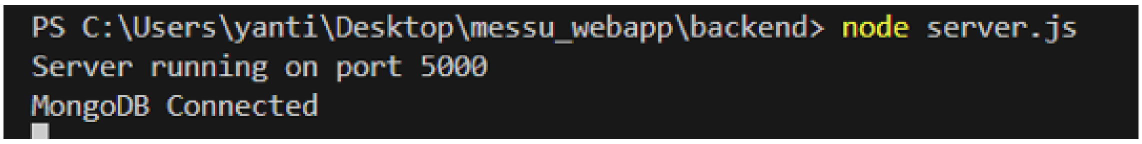
Code 34. Rendering ´App` component in ´main.jsx`.

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./app.jsx";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App/>
  </React.StrictMode>,
);
```

4.4.2 Testing the prototype page

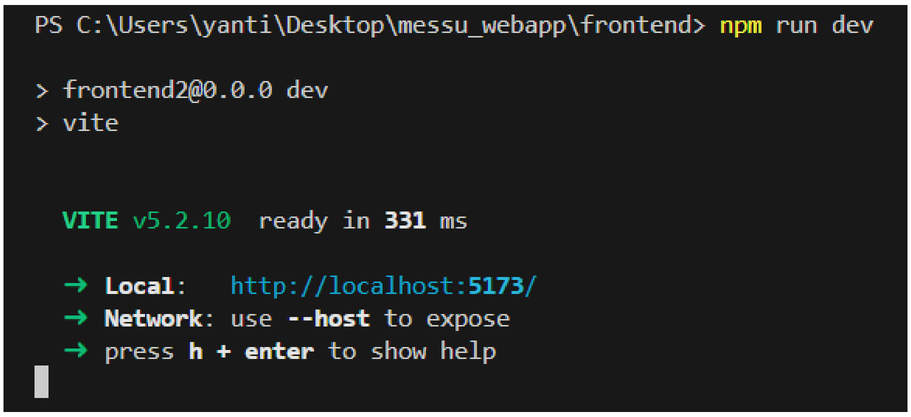
Basic functionalities of stamp pass prototype page can now be tested.

Firstly, by launching backend with ´node server.js` and frontend with ´npm run dev` from terminal in respective project directories. Figure 30 is showing terminal output identical to Figure 14, however in two completely different development stages. Figure 29 is showing successfully launched frontend at ´localhost:5137`, which can be accessed by inputting ´o` or ´open`. Inputting either of these commands will open a browser window at a specified address.



```
PS C:\Users\yanti\Desktop\messu_webapp\backend> node server.js
Server running on port 5000
MongoDB Connected
```

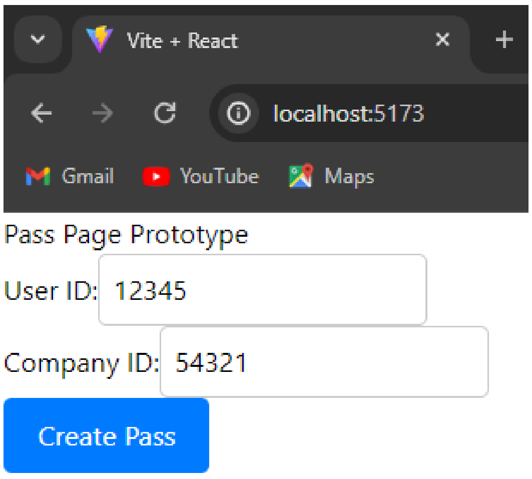
Figure 30. Screen capture of successful connection to a locally hosted database



```
PS C:\Users\yanti\Desktop\messu_webapp\frontend> npm run dev

> frontend2@0.0.0 dev
> vite


  VITE v5.2.10  ready in 331 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Figure 31. Screen capture of successfully started frontend

After successfully booting the project and being prompted to a browser window, both input fields can be filled with mock entries and “Create Pass” button clicked. Figure 32 is a depiction of the first frontend-based pass creation via passForm.js frontend component.



Pass Page Prototype

User ID: 12345

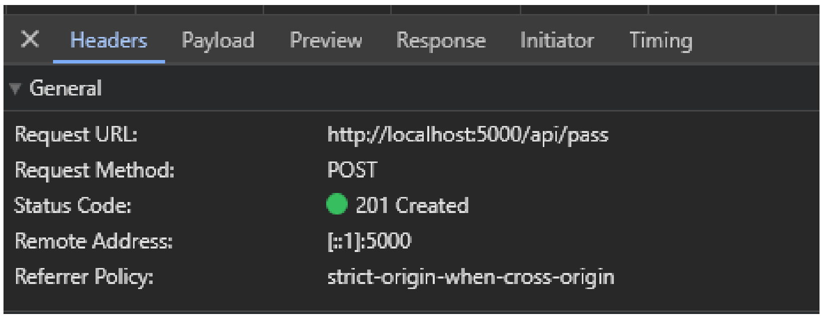
Company ID: 54321

Figure 32. Screen capture of a browser window containing filled form input fields

Clicking the ´Create Pass` button makes a POST request to the specified URL. The path that data takes from the moment of request to successful response form the database can be monitored through browser developer console, which can be typically accessed using “Ctrl + Shift + I” on Windows OS.

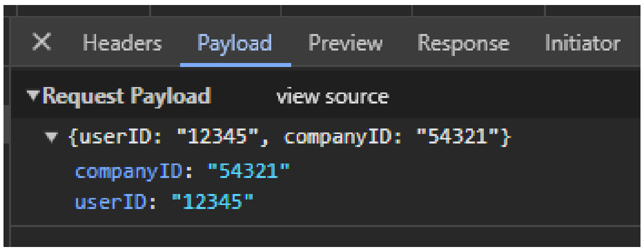
Figure 33 shows a successful POST request creation to a specified API endpoint.

In Figure 34, request payload is specified to ´companyID` and ´userID`, which means that after successful POST request, data that was transmitted contained fields specified in the ´pass.js` model schema.



```
Request URL:        http://localhost:5000/api/pass
Request Method:     POST
Status Code:        ● 201 Created
Remote Address:     [::1]:5000
Referrer Policy:    strict-origin-when-cross-origin
```

Figure 33. Screen capture of browser's developer console's "Network/Headers" tab



▾Request Payload

```
▾ {userID: "12345", companyID: "54321"}
    companyID: "54321"
    userID: "12345"
```

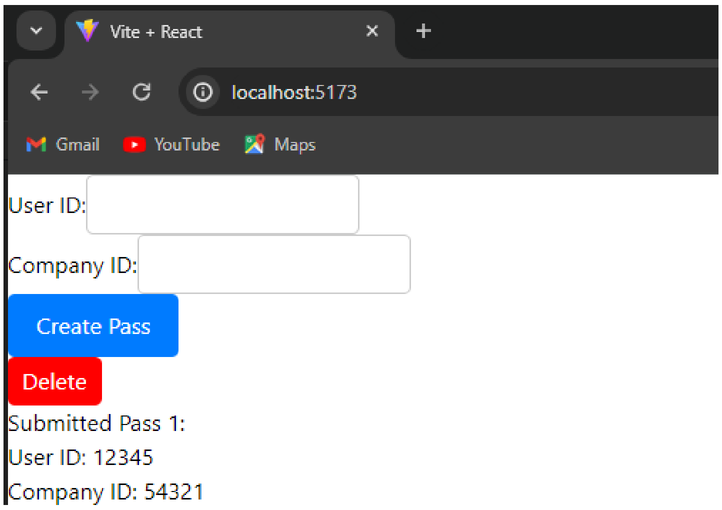
Figure 34. Screen capture of browser's developer console's "Network/Payload" tab

Server then responds to client’s request by creating a new pass with requested fields (Figure 35).

MongoDB assigns a unique database ID to it, as well as responding to a GET request by rendering the pass that was just created.

To confirm that created database entry exists, MongoDB Compass can be used as shown in Figure 36, by simply navigating to corresponding collection and inspecting it.

Lastly, pressing “Delete” makes a DELETE request to backend, removes the pass from rendered component as well as from the database itself. Confirmation of successful DELETE request can be performed via browser developer console by inspecting ´Headers`, ´Payload` and ´Response` tabs, or by simply refreshing and checking ´passes` collection contents via MongoDB Compass GUI (Figure 37).

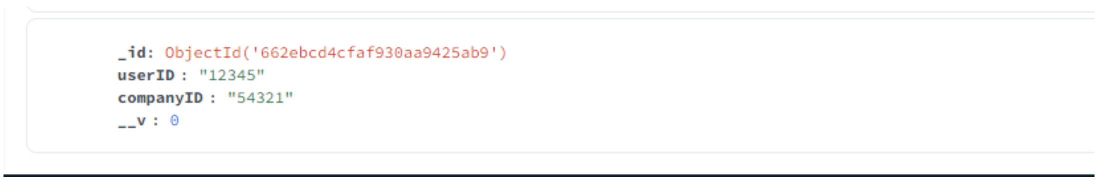


User ID:

Company ID:

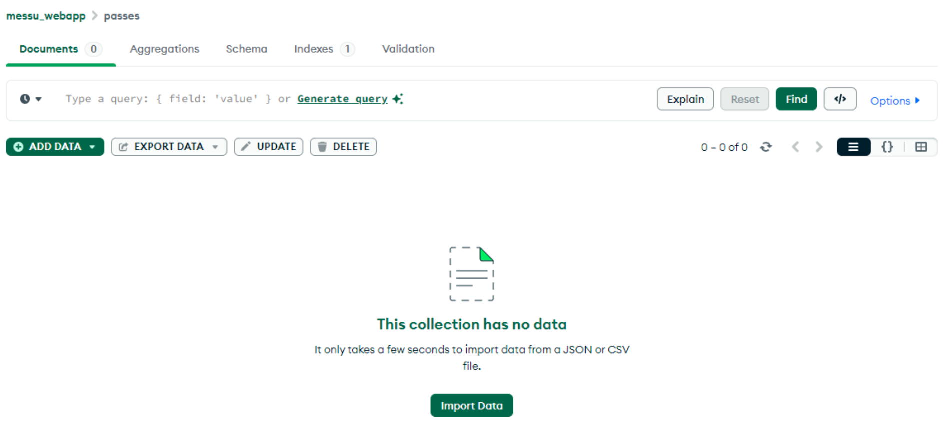
Submitted Pass 1:
User ID: 12345
Company ID: 54321

Figure 35. Screen capture of a browser window depicting successful pass creation.



```
_id: ObjectId('662ebcd4cfaf930aa9425ab9')
userID : "12345"
companyID : "54321"
__v : 0
```

Figure 36. Screen capture of created database entry accessed via MongoDB Compass.



This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Figure 37. Screen capture of empty collection after successful database entry deletion accessed via MongoDB Compass.

The prototype page that was created utilizes all core data handling methods (CRUD operations), as well as model schemas, routes, and event handlers. Using the prototype page as a reference, further development can now be picked up and continued by any developer.

5 Conclusion

In summary, this project has successfully progressed from its conceptualization to the implementation of a functional prototype page for a web application interface. Throughout the project work, agile development methodologies, design principles and in-depth planning were employed to achieve the set objectives.

The project was initiated with the articulation of a comprehensive project plan, which outlines the scope, milestones, and objectives. That, in combination with existing wireframes and mockups, served as what could be referred to as a roadmap for the project’s execution.

The development process was primarily focused on implementing backend and frontend components of the messu_webapp. To streamline the development and implementation process as well as enhance user experience, modern technologies such as Tailwind CSS and Vite were utilized.

The backend of the project was constructed using Node.js and Express.js, with MongoDB serving as the database management system. Model schemas, routing and mock data seeding scripts were developed to provide and assist in data manipulation and management.

On the frontend, Vite and Tailwind CSS were utilized to build a robust user interface, while Axios facilitated asynchronous communication with backend. Integration of backend and frontend components was established through coherent APIs and RESTful principles.

The deployment phase involved development environment setup, ensured seamless communication between the two. Testing and debugging were continuously performed in order to remedy issues that arose during the development.

In conclusion, the project has demonstrated a successful attempt to apply software engineering practices and principles to develop a functional web application prototype. This development process presented challenges that were tackled according to these principles which reflect the developer's expertise and dedication.

Opportunity for future development as well as a solid development infrastructure are established. Moving forward additional features and functionalities can be implemented and refined. Knowledge gained from the project will contribute to the advancement of software engineering, inform future endeavors, and inspire self-study and enhancement of own expertise in the field.

References

Angel, B. 2022. *“How to Create a Programming Style Guide.”* https://www.pullrequest.com/blog/create-a-programming-style-guide/. Referenced on 21.04.2024.

Atlassian 2024. *“Getting started – Git commit.”* https://www.atlassian.com/git/tutorials/saving-changes/git-commit. Referenced on 03.05.2024.

Canadian Centre for Cyber Security 2024. *“Security considerations for QR codes.”* https://www.cyber.gc.ca/en/guidance/security-considerations-qr-codes-itsap00141. Referenced on 17.04.2024.

Cooks-Campbell, A. 2023. *“What is asynchronous communication?”* https://www.betterup.com/blog/asynchronous-communication. Referenced on 13.04.2024.

Coursera 2024. *“NoSQL vs SQL.”* https://www.coursera.org/articles/nosql-vs-sql. Referenced on 15.04.2024.

Geeksforgeeks 2022. *“JavaScript ES2015: Block Scoping.”* https://www.geeksforgeeks.org/javascript-es2015-block-scoping/. Referenced on 19.04.2024.

Geeksforgeeks 2023. *“What are the advantages of using JSX in ReactJS?”* https://www.geeksforgeeks.org/what-are-the-advantages-of-using-jsx-in-reactjs/. Referenced on 30.04.2024.

Geeksforgeeks 2024. *“What is Array?”* https://www.geeksforgeeks.org/what-is-array/. Referenced on 13.04.2024.

Google Cloud 2024. *“What is a relational database?”* https://cloud.google.com/learn/what-is-a-relational-database. Referenced on 27.04.2024.

Great Learning Team 2022. *“README File – Everything you Need to Know.”* https://www.mygreatlearning.com/blog/readme-file/. Referenced on 03.05.2024.

Juviler, J. 2024. *“What Is an API Endpoint? (And Why Are They So Important?).”* https://blog.hubspot.com/website/api-endpoint. Referenced on 29.04.2024.

Lucidchart 2024. *“What is a Flowchart.”* www.lucidchart.com/pages/what-is-a-flowchart-tutorial. Referenced on 23.04.2024.

Mahlamäki, T. 2024. *“Messu App Prototype.”* https://www.figma.com/design/PuEx1dSMu99jCp7LahJwnb/Messu-App-Prototype?node-id=0-1&t=7gh4RRWJmEkaTNpW-0 Referenced on 05.06.2024

Maldonado, L. 2023. *“All you Need to Know About CORS & CORS Errors.”* https://www.telerik.com/blogs/all-you-need-to-know-cors-errors. Referenced on 07.05.2024.

MongoDB 2024. *“Why Use MongoDB and When to Use It?”* https://www.mongodb.com/why-use-mongodb. Referenced on 14.04.2024.

Node.JS 2024. *“About Node.js.”* https://nodejs.org/en/about. Referenced on 15.04.2024.

Nuclino 2024. *“Project Documentation: Examples and Templates.”* https://www.nuclino.com/solutions/project-documentation. Referenced on 16.04.2024.

Raka, J. 2023 *“A Guide to Mocking and Stubbing in Software Development.”* https://medium.com/@julius.prayoga/mock-stub-e7c0eaa801bf. Referenced on 29.04.2024.

Riosa, B. 2019. *“The (written) unwritten guide to pull requests.”* https://www.atlassian.com/blog/git/written-unwritten-guide-pull-requests. Referenced on 05.05.2024

University of Pittsburgh 2023. *“Open Licenses: Creative Commons and other options for sharing your work.”* https://pitt.libguides.com/openlicensing/MIT. Referenced on 25.04.2024.

Vite 2023. *“Why Vite?”* https://vitejs.dev/guide/why.html. Referenced on 02.05.2024.

WebRTC.org 2024. *“Real-time communication for the web.”* https://webrtc.org. Referenced on 19.04.2024.