THE UNIVERSITY OF DANANG
DANANG UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

GRADUATION PROJECT THESIS

MAJOR: INFORMATION TECHNOLOGY

SPECIALTY: SOFTWARE ENGINEERING

PROJECT TITLE:

BUILDING AN APPLICATION THAT SUPPORTS ONLINE FORM CREATION

Instructor: **PhD. NGUYEN VAN HIEU**
Student: **HO THUY TIEN**
Student ID: **102200196**
Class: **20TCLC_DT4**

Da Nang, 06/2024

THE UNIVERSITY OF DANANG
DANANG UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

GRADUATION PROJECT THESIS

MAJOR: INFORMATION TECHNOLOGY

SPECIALTY: SOFTWARE ENGINEERING

PROJECT TITLE:

BUILDING AN APPLICATION THAT SUPPORTS ONLINE FORM CREATION

Instructor: **PhD. NGUYEN VAN HIEU**
Student: **HO THUY TIEN**
Student ID: **102200196**
Class: **20TCLC_DT4**

Da Nang, 06/2024

SUMMARY

Project name: Building An Application That Supports Online Form Creation
Student name: Ho Thuy Tien
Student ID: 102200196
Class: 20TCLC_DT4

Summary: In response to current data collection challenges, my project leverages today's technological advancements, particularly AI, to develop an app that simplifies the creation, customization, and management of online forms. This app tackles the often-time-consuming nature of customization by enabling rapid form creation with preset common questions and a user-friendly drag-and-drop interface. Integrated form management features allow users to organize forms into folders and teams, significantly boosting efficiency. Additionally, the system incorporates OpenAI's chatbot for automatic form generation and supports the importation of existing Google Forms, facilitating seamless continuous data collection. By streamlining the form creation process and enhancing organizational capabilities, my app markedly reduces the time and effort required, providing a comprehensive and user-friendly solution for efficient data collection.

THE UNIVERSITY OF DANANG
UNIVERSITY OF SCIENCE AND TECHNOLOGY
INFORMATION TECHNOLOGY FACULTY

SOCIALIST REPUBLIC OF VIET NAM
Independence – Freedom – Happiness

GRADUATION PROJECT REQUIREMENTS

Student Name: Ho Thuy Tien Student No.: 102200196

Class: 20TCLC_DT4 Faculty: Information Technology Major: Software Engineering

1. *Topic title:*

   Building An Application That Supports Online Form Creation.

2. *Project topic: □ has signed intellectual property agreement for final result.*

3. *Initial figure and data: No data.*

4. *Content of the explanations and calculations:*

   The content of the thesis includes:

   INTRODUCTION – This chapter gives information about the context and purpose of the project as well as giving the scope of the problems which will be focused on the thesis.

   Chapter 1: THEORETICAL BACKGROUNDS– This chapter introduces about all knowledge theories and technologies used in this project.

   Chapter 2: ANALYSIS AND SYSTEM DESIGN – This chapter covers the main features, software requirement specifications and database design of the project.

   Chapter 3: IMPLEMENTATION AND EVALUATION– This chapter shows an implementation of this project, including pictures and a brief explanation for each main function.

   CONCLUSION – The concluding section of the project simultaneously emphasizes the problem solved, as well as presenting issues still unresolved and provides recommendations and suggestions.

   REFERENCES – Presentation about detail of referenced information used in this thesis.

Drawings, charts (specify the types and sizes of drawings): No drawings, no charts.

5. *Supervisor:* PhD. Nguyen Van Hieu
6. *Date of assignment:*
7. *Date of completion:*

Da Nang, date month year 2024

Head of Division……. **Instructor**

ACKNOWLEDGEMENTS

Today marks the conclusion of my senior project after an intense ten-week journey, and I am filled with deep gratitude. This experience has been a period of substantial learning and personal growth.

I owe immense thanks to Dr. Nguyen Van Hieu, my supervisor, whose unwavering support, guidance, and motivation have been crucial, despite their demanding schedule. Their expertise not only advanced my project's progress but also imparted essential skills for navigating research, understanding complex concepts, and tackling challenges with clarity and determination.

Although I have dedicated myself to this project, I acknowledge that there may be errors and limitations. Therefore, I eagerly welcome constructive feedback from the professors of the Faculty of Information Technology to address these areas and enhance the quality of my project.

Sincerely,

Ho Thuy Tien

ASSURANCE

I guarantee:

1. *The contents of this senior project are performed by myself following the guidance of supervisors PhD. Nguyen Van Hieu.*
2. *All references used in this senior project thesis, are quoted with the author's name, project name, time and location to publish clearly and faithfully.*
3. *All invalid copies, educated statute violation or cheating will be borne the full responsibility by myself.*

Students,

Ho Thuy Tien

TABLE OF CONTENT

Page

INSTRUCTOR'S COMMENTS ii
REVIEWER'S COMMENTS i
SUMMARY ii
GRADUATION PROJECT REQUIREMENTS iii
ACKNOWLEDGEMENTS i
ASSURANCE ii
TABLE OF CONTENT iii
LIST OF TABLES AND FIGURES v
LIST OF ACRONYMS viii
INTRODUCTION 1
Chapter 1: THEORETICAL BACKGROUND 3
1.1. Programming language 3
1.1.1 Node.js 3
1.1.2 JavaScript 3
1.1.3 TypeScript 3
1.1.4 HTML, CSS 4
1.2. Framework 4
1.2.1 Express.js 4
1.2.2 ReactJS 5
1.2.3 Tailwind 6
1.3. Library 6
1.3.1 Mantine 6
1.3.2 React grid layout 7
1.4. Database 8
1.5. RESTful programming 9
1.6. OpenAI 10
1.7. Cloudinary 10
1.8. Tools to support during project implementation 10
1.9. Conclusion 13
Chapter 2: ANALYSIS AND SYSTEM DESIGN 14
2.1. Requirement analysis and design 14
2.1.1 User interaction 14

2.1.2 Decomposition diagram 15
2.1.3 Main features 15
2.2. Use case diagram 16
2.2.1 Use case diagram for the system 16
2.2.2 Use case diagram for user to manage account 17
2.2.3 Use case diagram for user to manage form 17
2.2.4 Use case diagram for user to manage folder 18
2.2.5 Use case diagram for user to manage team 19
2.2.6 Use case diagram for user to manage response 20
2.3. Use case description 20
2.4. Sequence diagrams 41
2.4.1 Sequence diagrams for user to login to system 41
2.4.2 Sequence diagram for user to import form from Google Form 42
2.4.3 Sequence diagram for user to auto-generating form by chatbot 42
2.4.4 Sequence diagram for user to enable form on specific date 43
2.5. Database design 43
2.5.1 Database design model 43
2.5.2 Relationship between table 44
2.6. Conclusion 44
Chapter 3: IMPLEMENTATION AND EVALUATION 45
3.1. Development environment 45
3.1.1 Web service 45
3.1.2 Software Development Tools 45
3.1.3 Setup to send email 45
3.1.4 Get an Oauth 2.0 Client ID 46
3.1.5 OpenAI 48
3.1.6 Cloudinary 50
3.2. Demo main features and evaluation 51
CONCLUSION 75
1. Achievement 75
2. Future work 75
REFERENCES 76

LIST OF TABLES AND FIGURES

1. List of figures

Figure 1. Draw.io tool *11*
Figure 2. Editor VSCode tool *11*
Figure 3. PgAdmin tool *12*
Figure 4. Postman tool *12*
Figure 5. Github *13*
Figure 6. Trello *13*
Figure 7. The Online Form Creation Support System Decomposition Diagram *15*
Figure 8. Use-case diagram for the system *16*
Figure 9. Use-case diagram for user to manage account *17*
Figure 10. Use-case diagram for user to manage form *17*
Figure 11. Use-case diagram for user to manage folder *18*
Figure 12. Use-case diagram for user to manage team *19*
Figure 13. Use-case diagram for user manage response *20*
Figure 14. Sequence diagram for login *41*
Figure 15. Sequence diagram to import form from Google Form *42*
Figure 16. Sequence diagram to auto-generate form by chatbot *42*
Figure 17. Sequence diagram for user to enable form on specific date *43*
Figure 18. Database design *44*
Figure 19: Setup to send email *46*
Figure 20: Get an Oauth 2.0 Client ID *48*
Figure 21. Init the OpenAI and using the chat completions API *49*
Figure 22. Example of the system prompt *49*
Figure 23. API calls to the OpenAI's endpoint *50*
Figure 24. Login screen *51*
Figure 25. Signup Screen *52*
Figure 26. Hompage screen *52*
Figure 27. Create form *53*
Figure 28. Create form from scratch screen *54*
Figure 29. Edit elements screen *55*
Figure 30. Import form screen *56*
Figure 31. Auto-generate form by chatbot *57*

Figure 32. Chatbot allow preview form screen *57*
Figure 33. Move form to team screen *58*
Figure 34. Add form to folder screen *58*
Figure 35. View Trash form *59*
Figure 36. delete form permanently screen *59*
Figure 37. View share with me screen *60*
Figure 38. View favorite form screen *60*
Figure 39. Publish form screen *61*
Figure 40. Form settings screen *62*
Figure 41. Set disable on specific date screen *62*
Figure 42. Form notification status *63*
Figure 43. Preview form screen *63*
Figure 44. Add collaborator screen *64*
Figure 45. Create folder screen *65*
Figure 46. Change folder color screen *65*
Figure 47. Delete folder screen *66*
Figure 48. Create team screen *66*
Figure 49. Team workspace *67*
Figure 50. Manage member to team screen *67*
Figure 51. Add form to folder screen *68*
Figure 52. Create respone screen *69*
Figure 53. View responses screen *69*
Figure 54. Export response to excel file *70*
Figure 55. Report response screen *71*
Figure 56. Manage account screen *72*
Figure 57. Change password screen *72*
Figure 58. Forgot password screen *73*
Figure 59. Notificate email has been sent screen *73*
Figure 60. System sends email screen *74*
Figure 61. Reset password screen *74*

2. List of tables

Table 1. Use-case description for Login 20
Table 2. Use-case description for Signup 21
Table 3. Use-case description for Show account 21
Table 4. Use-case description for Update personal information 22
Table 5. Use-case description for Update password 22
Table 6. Use-case description for Reset password 23
Table 7. Use-case description for creating form 24
Table 8. Use-case description for importing form 24
Table 9. Use-case description for auto-generating form by chatbot 25
Table 10. Use-case description for Updating form 27
Table 11. Use-case description for user to View all forms 27
Table 12. Use-case description for user to view form detail 28
Table 13. Use-case description for user to add form to favorite form 28
Table 14. Use-case description for adding form to folder 29
Table 15. Use-case description for moving form to team 30
Table 16. Use-case description for deleting form 31
Table 17. Use-case description for purge form 31
Table 18. Use-case description for view shared forms 32
Table 19. Use-case description for search form 33
Table 20. Use-case description for sort form 33
Table 21. Use-case description for sharing form with link 34
Table 22. Use-case description for set form status 34
Table 23. Use-case description for receiving email when the form has a new response 35
Table 24. Use-case description for preview form 36
Table 25. Use-case description for preview form 37
Table 26. Use-case description for delete folder 37
Table 27. Use-case description for create team 37
Table 28. Use-case description for update team name 38
Table 29. Use-case description for view folder of team 38
Table 30. Use-case description for view all forms of team 39
Table 31. Use-case description for add folder to team 39
Table 32. Use-case description for view all responses 40
Table 33. Use-case description for create response 40

LIST OF ACRONYMS

No.	Items	Description
1	AI	Artificial Intelligence
2	MVC	Model-View-Controller
3	JSX	JavaScript XML
4	DOM	Document Object Model
5	HTTP	HyperText Transfer Protocol

INTRODUCTION

1. Project overview

1.1. Context

Today's technological advancements, especially the rise of AI, signify major progress for the global community and the tech industry. Data is a crucial component, particularly in the tech sector. Common tools like Google Sheets and Google Forms are often used for efficient user data collection. However, despite their convenience, customization can be quite time-consuming. To address this, we have developed an application that facilitates rapid online form creation, reducing the effort required for customization while ensuring comprehensive form completion.

1.2. Statement of problem

Adjustments can be time-consuming for users, and storing and searching for forms can be inconvenient since it often requires navigating to another Google platform like Drive for management. To tackle these challenges, we have developed a system that allows users to easily customize forms and provides a set of common data collection questions for quick use, minimizing the need for extensive adjustments.

Additionally, our system includes built-in form management features and integrates OpenAI's chatbot for automatic form creation. Furthermore, if you have existing Google Forms, our system allows for easy import, streamlining regular data collection.

2. Purpose and responsibility

2.1. Purpose

This app helps users save time and effort by quickly and easily creating forms. Users can drag and drop various elements, such as: fullname, email, phone,… to meet specific needs, reducing the time spent designing and arranging forms from scratch. Managing forms by folder and team helps organize them better, and overall form management is enhanced.

2.2. Responsibility

- Research on the challenges and issues faced when creating a form with chatbot using OpenAI API and importing a form from Google Form.
- Analysis and designing of information systems.
- Designing a database for the system.
- Identifying the actors and constructing the database.
- Learning and researching the use of NodeJS programming languages and tools, JavaScript with ReactJS frameworks, and how to build WebAPI following the RESTful mechanism.
- Researching and utilizing PostgreSQL as a database platform.
- Developing apps with core functionalities.
- Prepare the report.

3. Structure of the thesis

INTRODUCTION – This chapter gives information about the project's context and purpose and the scope of the problems which will be focused on the thesis**.**

Chapter 1: THEORIES AND TECHNOLOGIES – This chapter introduces all knowledge theories and technologies used in this project.

Chapter 2: ANALYSIS AND SYSTEM DESIGN – This chapter covers the main features, software requirement specifications and database design of the project.

Chapter 3: IMPLEMENTATION AND EVALUATION– This chapter shows an implementation of this project, including pictures and a brief explanation for each main function.

CONCLUSION – The concluding section of the project emphasizes the problem solved, presents issues still unresolved and provides recommendations and suggestions**.**

REFERENCES – Presentation about detail of referenced information used in this thesis.

Chapter 1: THEORETICAL BACKGROUND

1.1. Programming language

1.1.1 Node.js

Definition

- **Node.js** is an open-source, cross-platform, JavaScript runtime environment that allows developers to execute JavaScript code outside of a web browser. It is built on the V8 JavaScript engine and is designed for building scalable network applications. Node.js enables developers to use JavaScript for server-side scripting, thus unifying the development stack. [1]

Key Features of Node.js

- **Asynchronous and Event-Driven:** Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, suitable for data-intensive real-time applications.
- **Single Programming Language:** Developers can write both the client-side and server-side code in JavaScript, fostering code reuse and reducing the context-switching between different programming languages.
- **Scalability:** Node.js is designed to build scalable network applications, with capabilities to handle many simultaneous connections with high throughput.
- **Rich Ecosystem:** Node.js has a rich library of various JavaScript modules, which simplifies the development of web applications and APIs.

1.1.2 JavaScript

Definition

- **JavaScript** is a versatile, high-level programming language primarily used to create interactive and dynamic content on web pages. It is a core technology of the World Wide Web, alongside HTML and CSS. JavaScript is used both on the client side (in browsers) and on the server side (with environments like Node.js). It supports event-driven, functional, and imperative programming styles. [2]

1.1.3 TypeScript

Definition

- **TypeScript** is a strongly typed superset of JavaScript that adds static type definitions. Developed by Microsoft, TypeScript compiles down to plain JavaScript, ensuring compatibility with all browsers and environments that support JavaScript. The addition of types improves code quality and developer

productivity by enabling better tooling, such as autocompletion and type-checking, which can catch errors at compile time rather than at runtime. [3]

1.1.4 HTML, CSS

Definition

- **HTML** (Hyper Text Markup Language) is the standard markup language used to create and structure content on the web. It provides the basic building blocks for web pages, defining elements such as headings, paragraphs, links, images, and other multimedia content. HTML uses tags to denote different elements within the content, which browsers interpret to display web pages correctly. [4]
- **CSS** (Cascading Style Sheets) is a style sheet language used to describe the presentation of a document written in HTML. It controls the layout, colors, fonts, and overall visual appearance of web pages. CSS allows developers to separate content (HTML) from design (CSS), making it easier to maintain and update the look and feel of a website. [5]

1.2. Framework

1.2.1 Express.js

Definition

- **Express.js**, often referred to simply as Express, is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It facilitates the management of server-side logic and APIs, enhancing the functionality and performance of web applications.
- Together, Node.js and Express.js form a powerful technology stack for building server-side applications with JavaScript. [6]

Key Features of Express.js

- **Middleware:** Express.js uses a series of middleware functions to handle requests and responses. Middleware can be used for various purposes like logging, authentication, and data parsing.
- **Routing:** Express.js provides robust routing mechanisms to handle different HTTP methods and URL paths, enabling developers to create RESTful APIs efficiently.
- **Template Engines:** Express.js supports various template engines such as Pug and EJS, allowing dynamic generation of HTML pages based on data from the server.
- **Error Handling:** Express.js includes a built-in mechanism for handling errors, making it easier to manage and debug issues within applications.

- **Extensibility:** Express.js can be extended with various plugins and libraries, enhancing its core functionality without compromising performance.

1.2.2 ReactJS

Definition

- React JS is JavaScript library used for building reusable UI components. According to React official documentation, following is the definition:
- React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding. [7]

JSX

- JSX is a language that allows writing HTML code in JavaScript. JSX performs optimization while compiling JavaScript code. These codes give a much faster execution time than an equivalent code written directly in JavaScript. In contrast to JavaScript, JSX is statically typed, meaning it is compiled before runtime, just like Java and C++. So, errors will be detected during compilation. In addition, it also provides very good debug when compiling. Easier: Easier. Legacy JSX is based on Javascript, so it's very easy for JavaScript programmers to use.

Single-way data flow

- ReactJS does not have modules to do dedicated tasks to process data. So ReactJS will break down the view into small components, they have a close relationship. Why should we care about the structure and relationships between components in ReactJS? Answer by data flows in ReactJS: Data flow is one-way from parent to child. It can be said that ReactJS using one-way data flow will create difficulties. However, this will make ReactJS promote its use and role.

Virtual DOM

- Frameworks using Virtual-DOM are typical ReactJS at the time of Virtual-DOM changes, the programmer does not need to manipulate the DOM directly in the view, but the change is still reflected. Because Virtual-DOM both acts as a Model and a View, so a change on the Model entails a change on the View, of course and vice versa. Although it does not directly affect the DOM elements

in the View, the programmer can still implement the mechanism of data binding. This makes the speed increase in a better way.

- Currently, programmers or enterprise companies are always looking for the best technology within reach to beat the competition, one of them is ReactJS. In a word, ReactJS will improve user experience, higher click and conversion rates. It's not just DOM updates that make apps faster and load better. Businesses using ReactJS are guaranteed to have a better interface than those using conventional frameworks.

Components

- React is built around components, not templates like other frameworks. In React, we build websites using small components. We can reuse a component in many places, with different states or properties, in a component that can contain other components. Each component in React has its own, mutable state, and React will perform component updates based on state changes. Everything React is a component. They help maintain code when working on large projects. A simple react component just needs a render method. There are many other methods available, but rendering is the dominant method.

Props and State

- Props: helps components interact with each other, the component takes input called props, and returns properties that describe what the child component should render. Prop is immutable. State: represents the state of the application, when the state changes, the component also re-renders to update the UI.

1.2.3 Tailwind

Definition

- **Tailwind CSS** is a utility-first CSS framework that provides low-level utility classes to build custom designs directly in your markup. Unlike traditional CSS frameworks that offer pre-designed components, Tailwind focuses on utility classes for styling, giving developers more flexibility and control over their designs. [8]

1.3. Library

1.3.1 Mantine

Definition

- **Mantine** is a comprehensive React component library that provides a rich set of customizable and accessible UI components designed to speed up the development of modern web applications. It includes a wide variety of components such as buttons, forms, modals, tables, and more, along with

utility functions and hooks that enhance the development experience. Mantine is built with TypeScript, ensuring strong typing and robust integration with TypeScript-based projects. [9]

Operation

- **Mantine** operates by offering a collection of pre-built React components that developers can use to construct user interfaces quickly and efficiently. These components are designed with a focus on customization and accessibility, allowing developers to tailor them to fit their application's specific needs. Mantine components come with default styles that can be easily overridden using props, styles, or custom themes.
- To use Mantine in a project, developers typically follow these steps:
  - **Installation**: Install Mantine and its dependencies via a package manager like npm or yarn.
  - Setup: Wrap the application with Mantine's MantineProvider to enable theming and provide context for the components.
  - Usage: Import and use Mantine components in the application.
- Mantine components are designed to be highly customizable. Developers can pass props to change the appearance and behavior of components, use styles for fine-grained control, and define custom themes to ensure consistency across the application.

1.3.2 React grid layout

Definition

- React-Grid-Layout is a responsive grid layout system that supports breakpoints. Specifically, in this project, it enables drag-and-drop functionality. For example, users can easily rearrange and resize grid items by dragging them around the layout. This flexibility allows for dynamic and interactive user interfaces that can adapt to different screen sizes and device capabilities. [10]

Features I applied in my project:

- Draggable widgets
- Static widgets
- Configurable packing: horizontal, vertical, or off
- Widgets may be added or removed without rebuilding grid
- Layout can be serialized and restored
- Responsive breakpoints
- Separate layouts per responsive breakpoint
- Grid Items placed using CSS Transforms

- Compatibility with <React.StrictMode>

1.4. Database

The project uses PostgreSQL database and Prisma.

Definition

- **PostgreSQL** is a powerful, open-source object-relational database management system (ORDBMS) known for its robustness, extensibility, and SQL compliance. Developed by the PostgreSQL Global Development Group, it supports both SQL (relational) and JSON (non-relational) querying, making it a versatile choice for developers. PostgreSQL is widely used in various applications, from web development to data analytics, due to its advanced features like ACID compliance, full-text search, and support for complex queries. [11]
- **Prisma** is an open-source database toolkit that simplifies database access for developers by offering an ORM (Object-Relational Mapping), a migration system, and a query engine. It works with various databases, including PostgreSQL, MySQL, and SQLite. Prisma's main goal is to make database interactions more intuitive and type-safe, especially when working with modern JavaScript and TypeScript applications. [12]

Operation

- **PostgreSQL** operates as a client-server model where the PostgreSQL server (the database engine) processes SQL queries and returns the results to the client applications. It supports various data types, advanced indexing techniques, and complex transactions. PostgreSQL ensures data integrity through mechanisms like foreign keys, triggers, and constraints.

- **Prisma** acts as an intermediary between the database and the application code. It involves three main components:
    - Prisma Client: An auto-generated and type-safe query builder for database operations.
    - Prisma Migrate: A tool for managing schema migrations, ensuring the database schema stays in sync with the application's data models.
    - Prisma Studio: A GUI for exploring and managing the data in the database.
- When using Prisma with PostgreSQL, the workflow typically involves:
    - Defining a data model in the Prisma schema file, which describes the structure of the data and relationships.

- Running migrations to sync the data model with the PostgreSQL database schema.
- Generating the Prisma Client to interact with the database using JavaScript or TypeScript.

1.5. RESTful programming

Definition

- REST stands for **Re**presentational **S**tate **T**ransfer. REST is a web standards-based architecture and uses HTTP Protocol for data communication. It revolves around resources where every component is a resource, and a resource is accessed by a common interface using HTTP standard methods.
- In REST architecture, a REST Server provides access to resources, and the REST client accesses and presents them. REST uses various representations to represent a resource like Text, JSON, and XML. JSON is now the most popular format used in Web Services. [13]

RESTful Web Service

- A web service is a collection of open protocols and standards used for exchanging data between applications or systems.
- Web services based on REST Architecture are known as RESTful Web Services. These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI (Uniform Resource Identifier), which is a service that provides resource representation such as JSON and a set of HTTP Methods.

HTTP methods

- **GET** − Provides a read only access to a resource.
- **PUT** − Used to create a new resource.
- **DELETE** − Used to remove a resource.
- **POST** − Used to update an existing resource or create a new resource.
- **OPTIONS** − Used to get the supported operations on a resource.

RESTful Resources

- REST architecture treats every content as a resource. REST uses various representations to represent a resource where Text, JSON, XML. The most popular representations of resources are XML and JSON.
- A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database.

RESTful Message

- RESTful Web Services make use of HTTP protocols as a medium of communication between client and server. A client sends a message in the form of a HTTP Request and the server responds in the form of an HTTP Response.

1.6. OpenAI

Definition

- OpenAI is a leading artificial intelligence research lab and technology company focused on creating and promoting advanced AI technologies. It offers a variety of AI models and tools, including natural language processing (NLP) models such as GPT-4, which can understand and generate human-like text. OpenAI's mission is to ensure that artificial general intelligence (AGI) benefits all of humanity. Its models can be used in a wide range of applications, including chatbots, content creation, and automated form generation. [14]

Operation

- OpenAI operates by providing access to powerful AI models through APIs, which developers can integrate into their applications to enhance functionality and user experience. These models are trained on vast amounts of data and can understand and generate text, making them ideal for tasks like conversation, text completion, translation, and more. OpenAI models are designed with an emphasis on usability, allowing developers to seamlessly incorporate AI capabilities into their projects.

1.7. Cloudinary

Definition

- Cloudinary is a software solution based on Software-as-a-Service (SaaS). It is primarily used for managing all the media assets present in the cloud

1.8. Tools to support during project implementation

- In the preocess of analyzing and designing UML diagrams, the tool draw.io (accessible at http://app.diagrams.net) is used. This tool simplifies the creation and manipulation of various UML diagrams, such as use case diagrams, activity diagrams, and more.

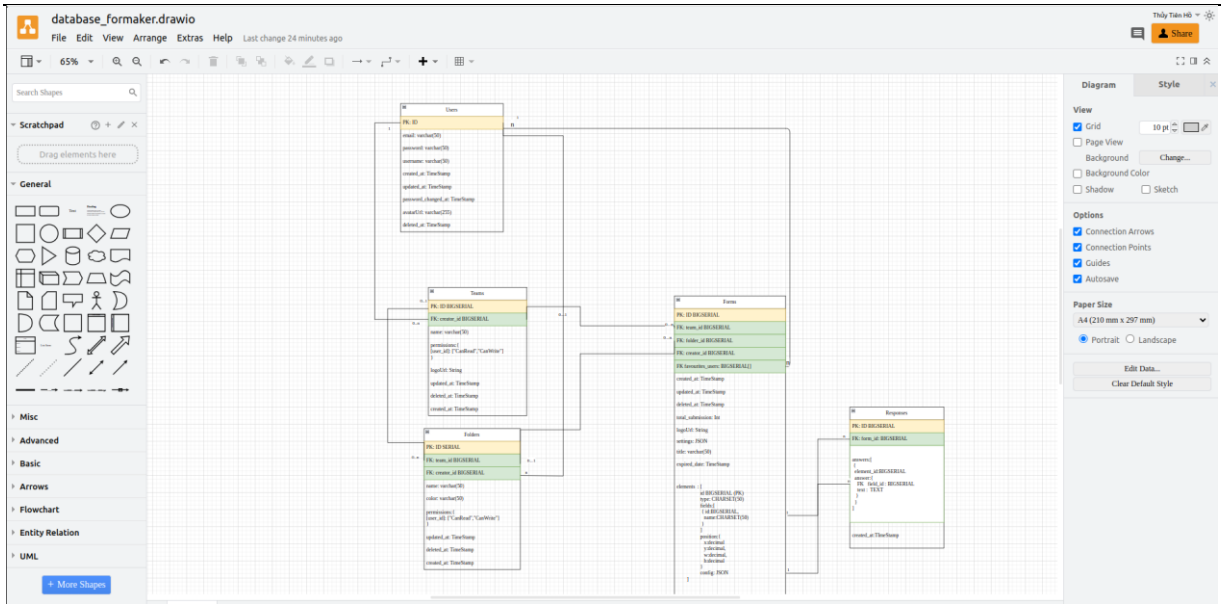

Figure 1. Draw.io tool

- The process of building program code is facilitated by the VSCode editor, known for its simplicity, lightweight nature, and ease of installation. VSCode supports many operating systems and a wide range of programming languages, making it an outstanding tool for developers.

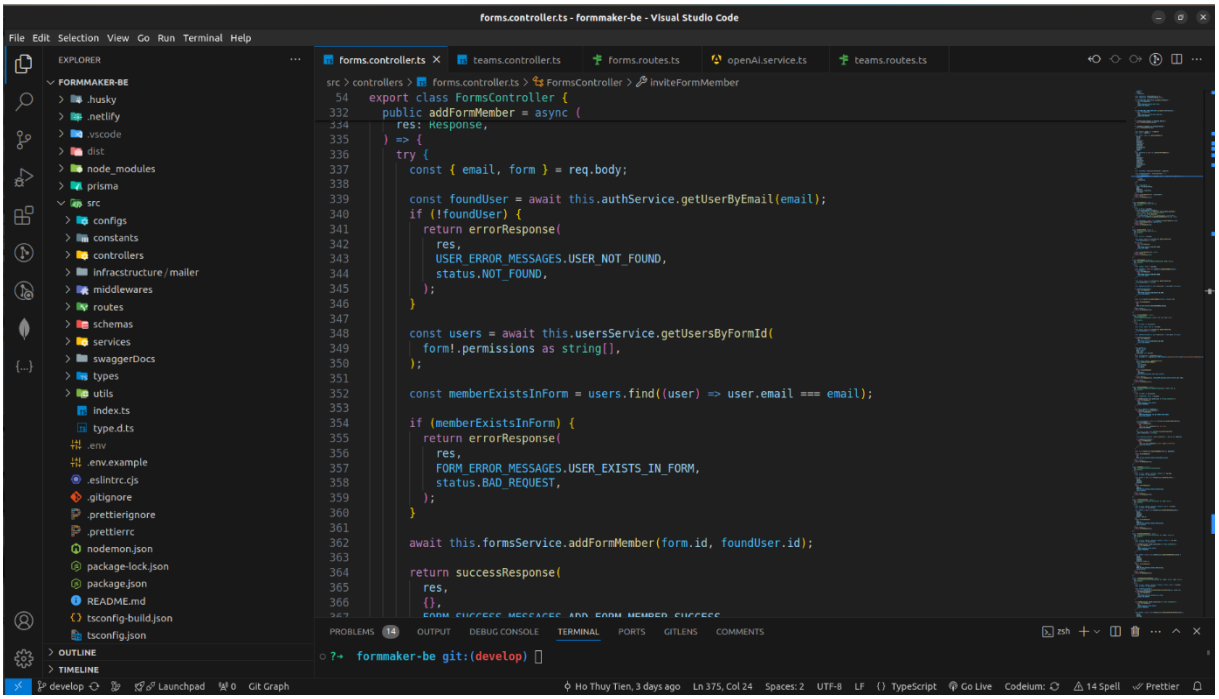

Figure 2. Editor VSCode tool

- The process of storing and querying data is facilitated by PgAdmin. This tool is known for its simplicity, ease of use, and seamless connection setup. It

supports the editing, querying, and viewing of data, making it an advantageous choice for managing PostgreSQL databases.

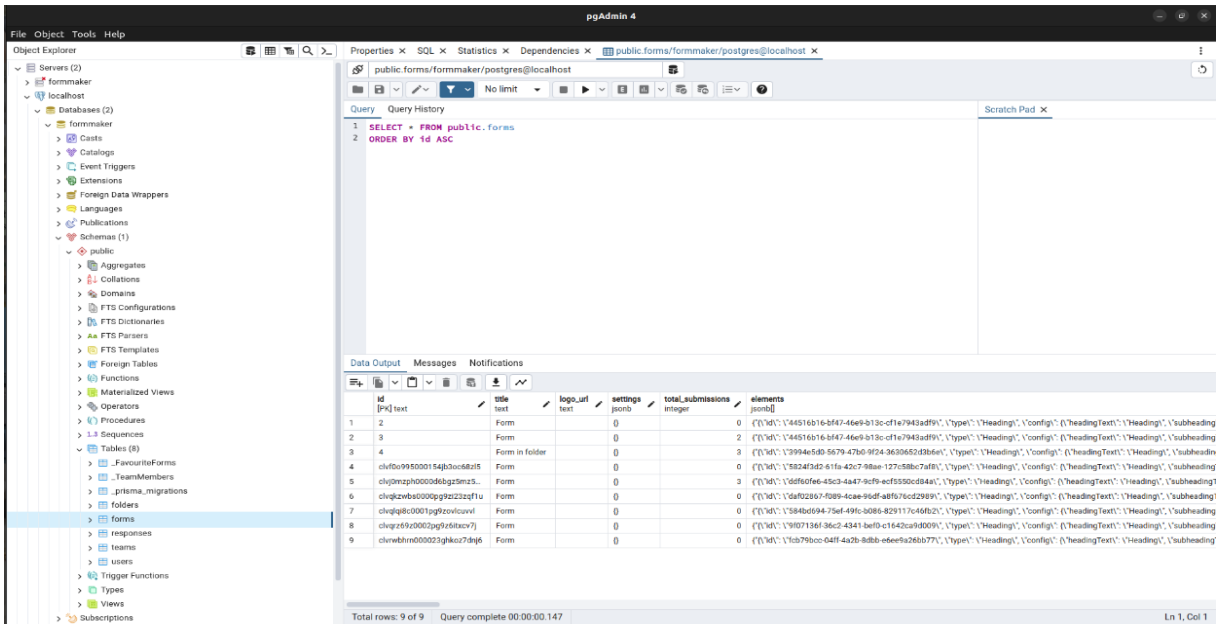

Figure 3. PgAdmin tool

- The process of checking and testing APIs before they are used to handle website requests is facilitated by Postman. This tool is used to ensure that the APIs are functioning correctly and efficiently before deployment.

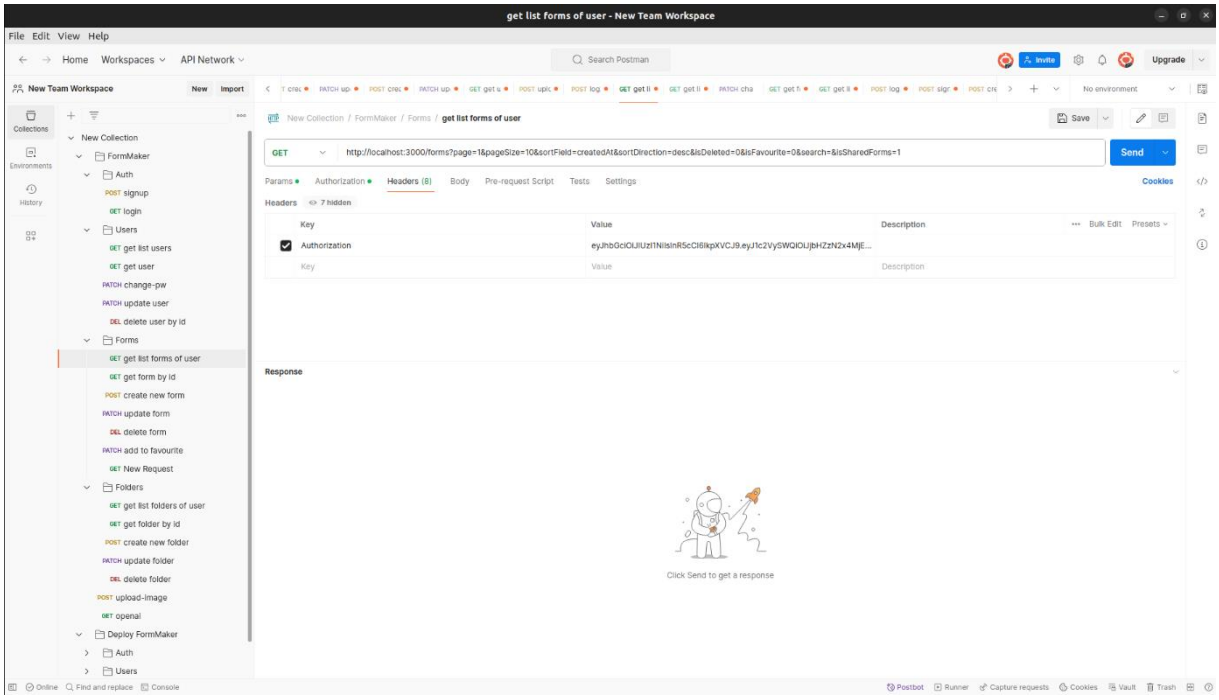

Figure 4. Postman tool

- GitHub is a collaborative platform for software development, offering version control, issue tracking, and collaboration tools. Its user-friendly interface and

extensive language support streamline code sharing, reviewing, and deployment processes for teams and individuals.

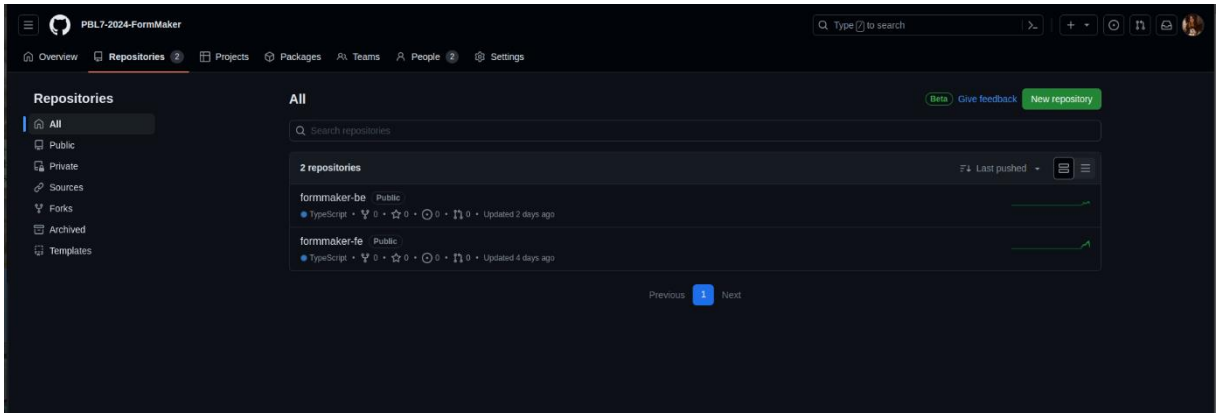

Figure 5. Github

- Trello simplifies project management, allowing us to visualize progress and collaborate effectively. Its flexibility enables customization to suit specific project needs, while features like due dates and checklists enhance productivity and transparency, making it an asset for efficient project management.

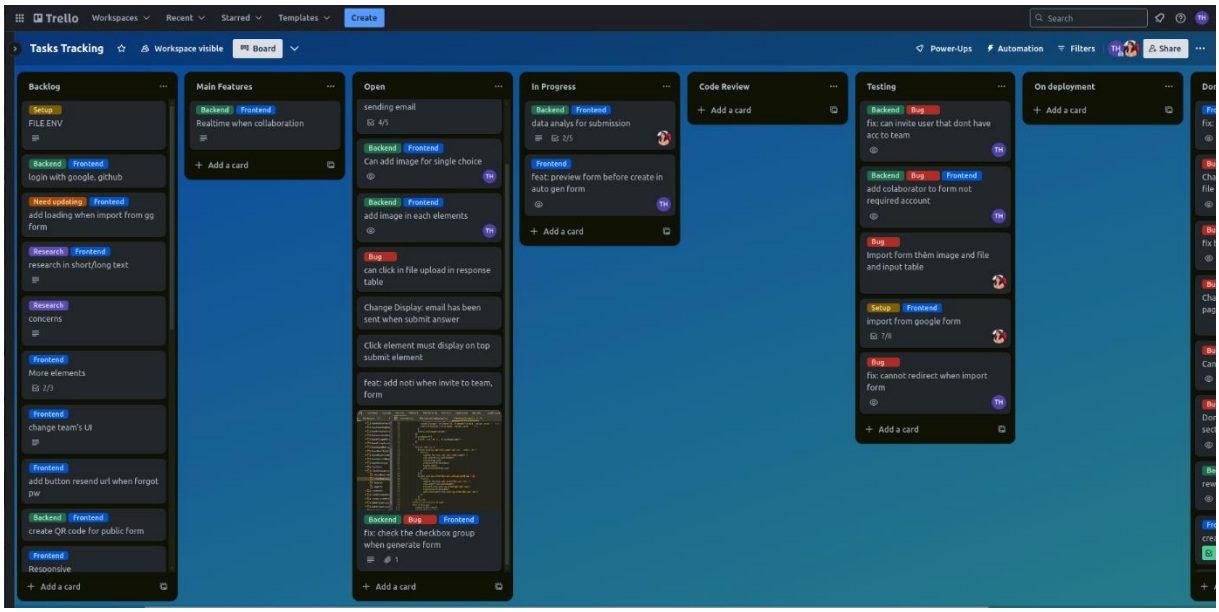

Figure 6. Trello

1.9. Conclusion

- By thoroughly studying and mastering the aforementioned technologies, I effectively applied their principles and functionalities in developing an online learning support system for primary and secondary schools.
- While some of these technologies are not novel, they remain prevalent and influential in the software development industry. Understanding their concepts is crucial for appropriately selecting and applying technology in each project, ultimately enhancing efficiency and usability.

Chapter 2: ANALYSIS AND SYSTEM DESIGN

This chapter will explore the requirements in detail, covering nonfunctional requirements, design constraints, and other factors needed for a comprehensive description of the application requirements. It includes a package with Requirements Specification, Use-Cases of the use-case model, Use Case Specifications, and Activity Diagram. This offers an overview of the application's functions. Additionally, it defines the architecture, modules, and data required for a system to fulfill specified requirements. System design serves as the bridge between system architecture and the implementation of technological system elements that form the physical architecture model of the system, applying systems theory to product development.

The System Design process aims to provide detailed data and information about the system's elements, enabling implementation consistent with the architectural entities defined in models and views of the system architecture. It illustrates the application's components, data table structures, and the relationships between system elements.

2.1. Requirement analysis and design

2.1.1 User interaction

This system has only one actor is user who manage forms, folders, teams, responses.

2.1.2 Decomposition diagram

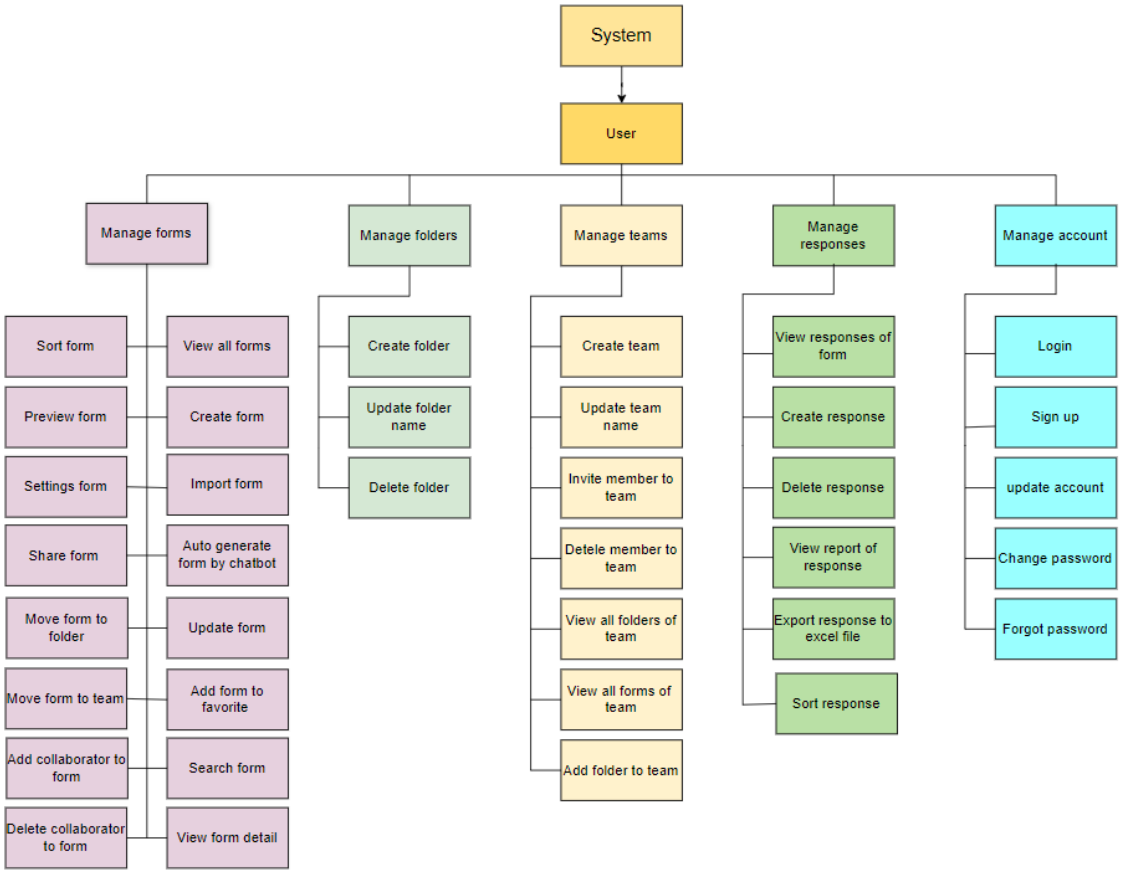

Figure 7. The Online Form Creation Support System Decomposition Diagram

This is the decomposition diagram that describes all main functions of the system.

2.1.3 Main features

- Manage form:
  - Create form form scratch
  - Import the existing form from Google form
  - Auto-generate form by chatbot
  - Store form with folder and team
  - Share form to others
  - Setting form
  - Preview form
  - Add collaborator to form
- Manage folder:
  - Create folder
  - Read folder
  - Update foler

- Delete folder

- Manage team:
  - Invite member to team by email
  - Add folder to team
- Manage response:
  - Create response
  - View Response
  - Sort response
  - Export to excel file
- Manage account:
  - Change password
  - Forgot password

2.2. Use case diagram

2.2.1 Use case diagram for the system

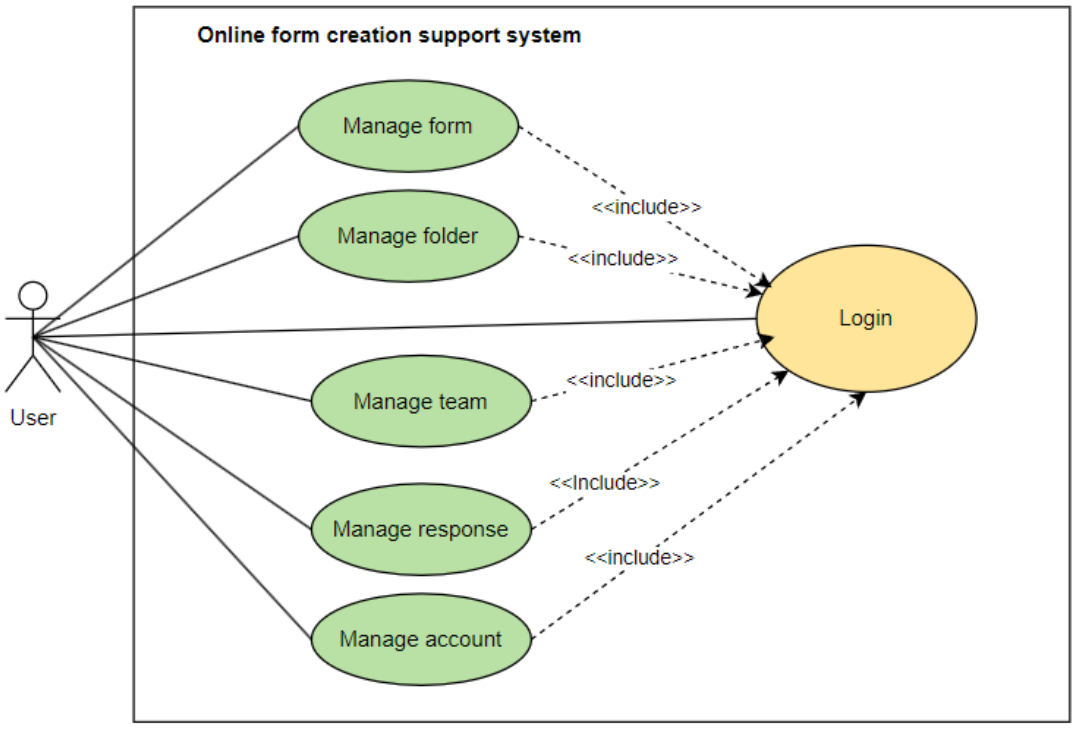

Figure 8. Use-case diagram for the system

2.2.2 Use case diagram for user to manage account

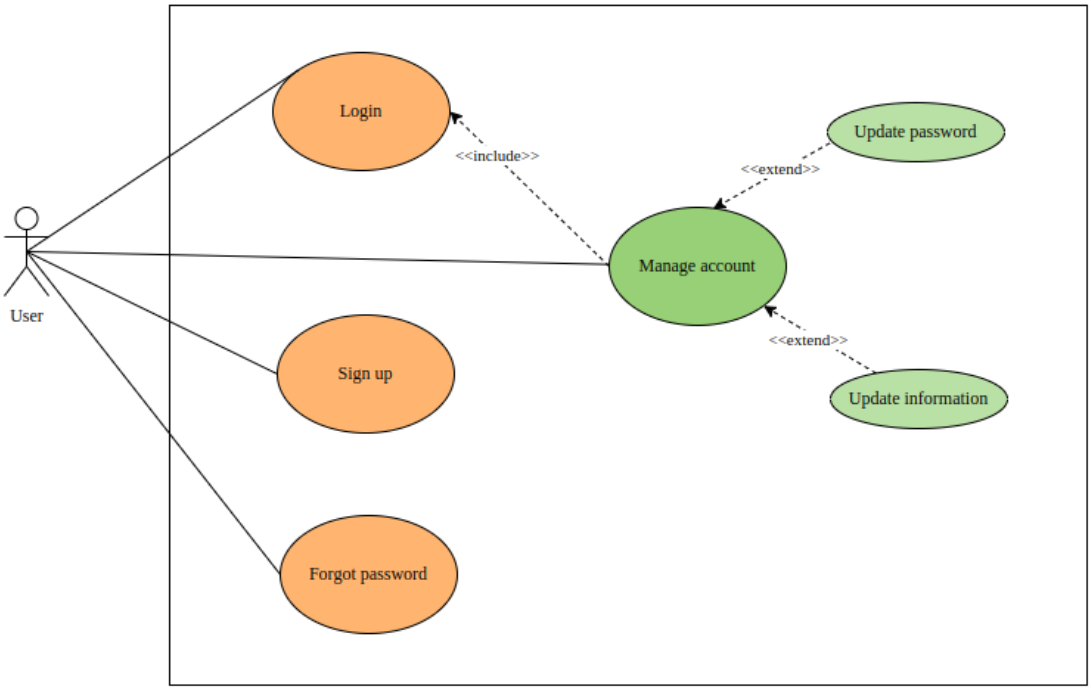

Figure 9. Use-case diagram for user to manage account

2.2.3 Use case diagram for user to manage form

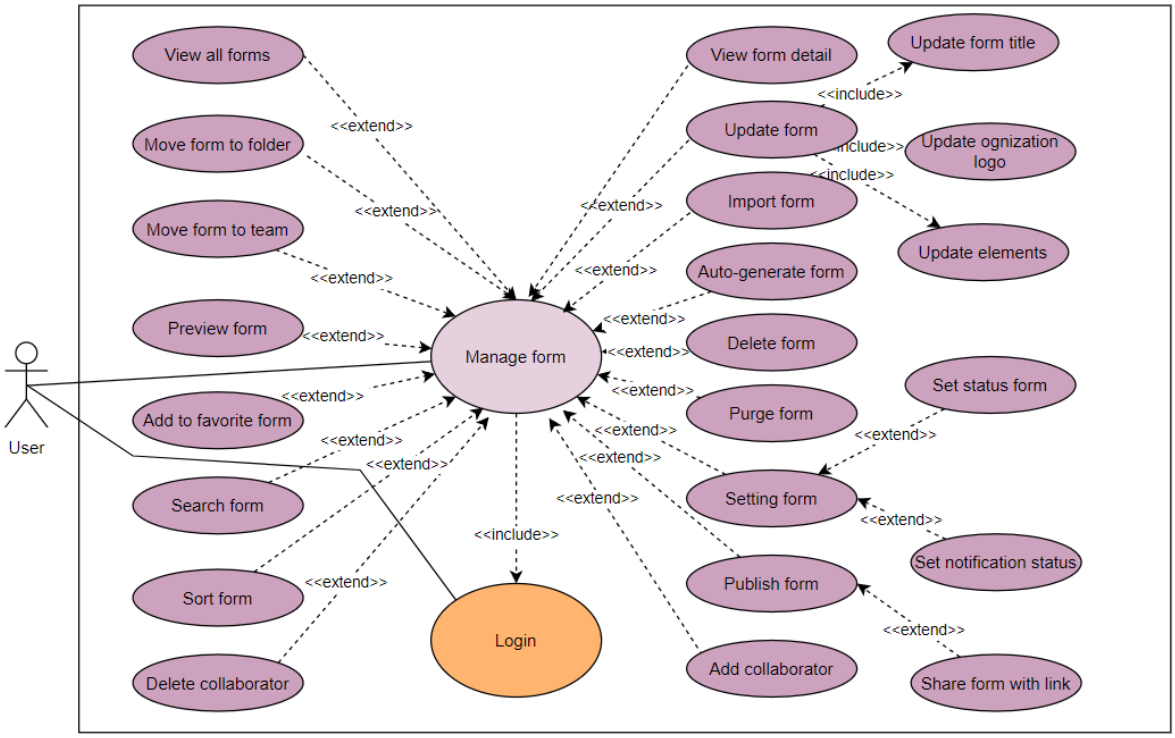

Figure 10. Use-case diagram for user to manage form

2.2.4 Use case diagram for user to manage folder

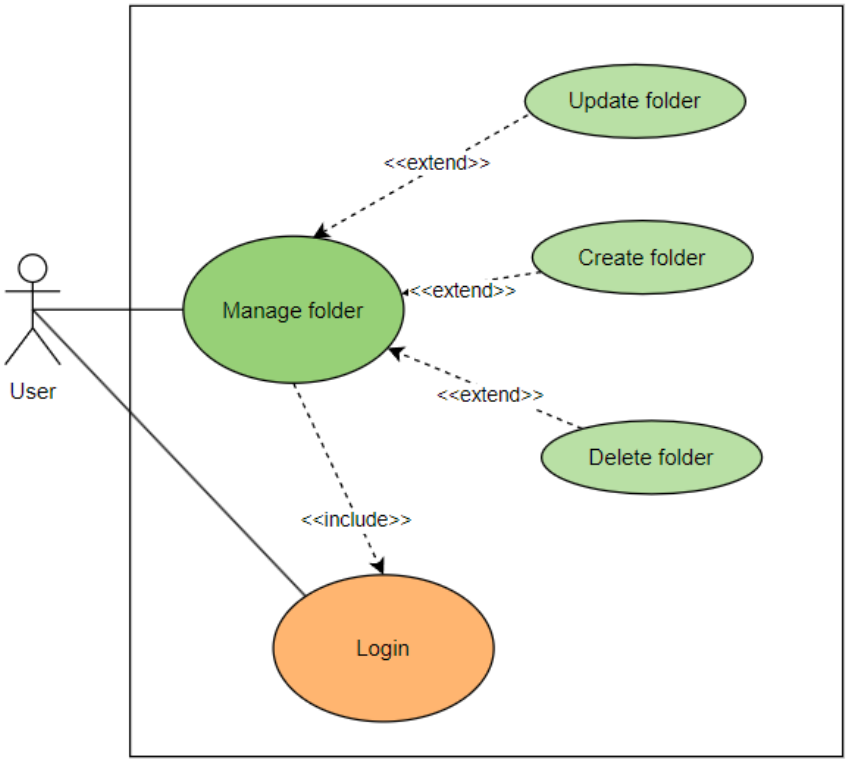

Figure 11. Use-case diagram for user to manage folder

2.2.5 Use case diagram for user to manage team

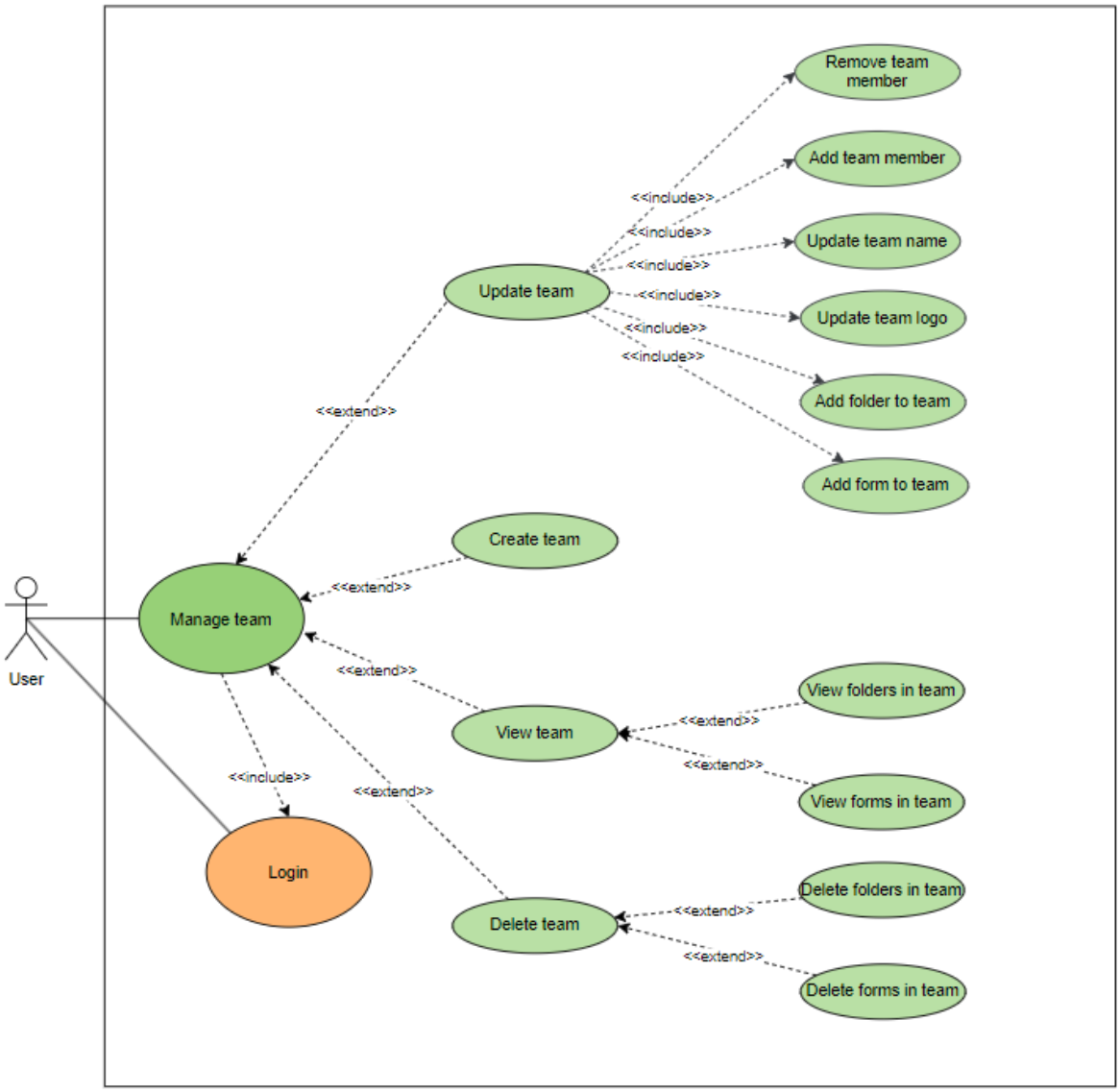

Figure 12. Use-case diagram for user to manage team

2.2.6 Use case diagram for user to manage response

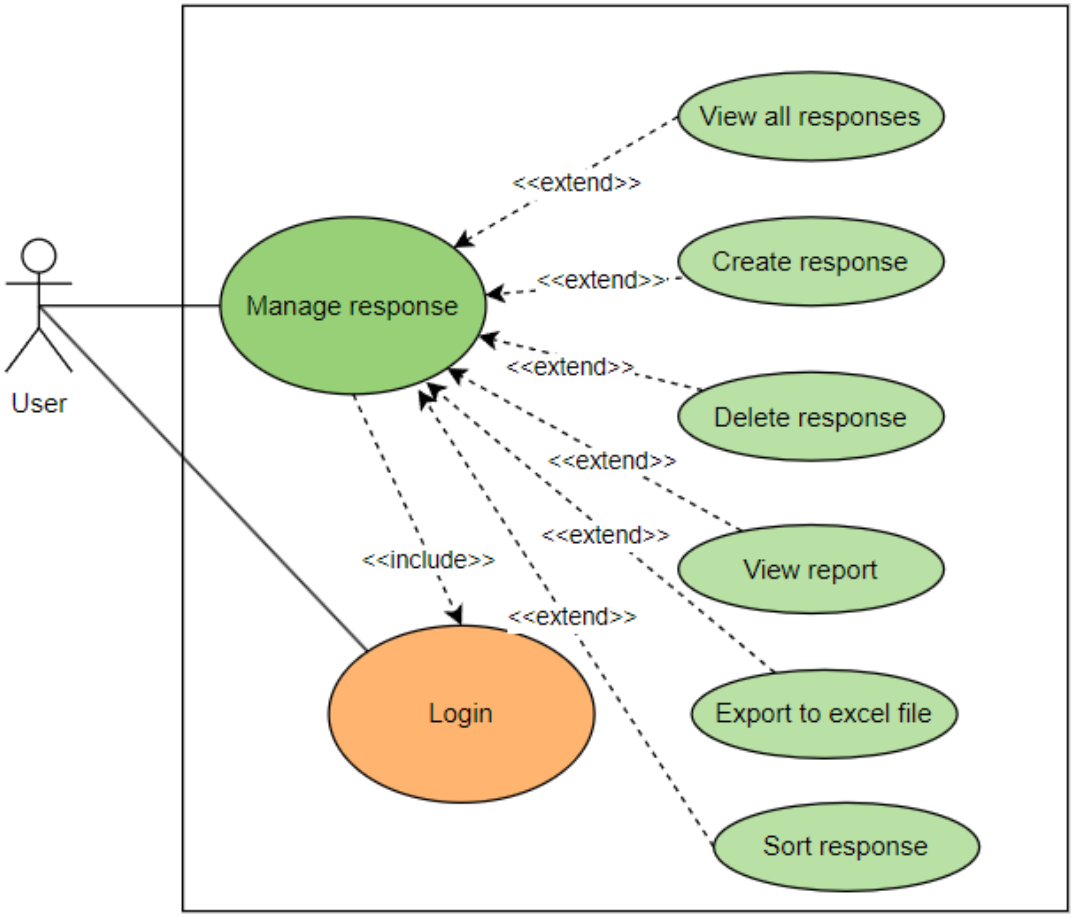

Figure 13. Use-case diagram for user manage response

2.3. Use case description

Table 1. Use-case description for Login

Use Case ID	**UC-01**		
Actor	User		
Brief description	This use-case for logging in.		
Pre-conditions	User who has account to access to application.		
Post-conditions	User logged in successfully.		
Flow of events		**Actor Input**	**System Response**
	1	User opens the website.	
	2		System shows login form.
	3	User fills in the email and password. Click the Login button.	

	4		System validates accounts that user filled and submit. Show overview page.

Table 2. Use-case description for Signup

Use Case ID	**UC-02**		
Actor	User		
Brief description	This use-case for users to register new account.		
Pre-conditions	User who wants to access to application.		
Flow of events		**Actor Input**	**System Response**
	1	User opens the website.	
	2		System shows login form.
	3	User click the Sign Up in Login screen.	
	4		System shows sign up form.
	5	User fills the username, email, password and confirm password again. Click the Sign up button.	
	6		The system validates the information filled in by the user upon submission. And then navigate to the login page.

Table 3. Use-case description for Show account

Use Case ID	**UC-03**		
Actor	User		
Brief description	This use-case for show personal information.		
Pre-conditions	User already logged into the system.		
Post-conditions	Server receives the request from Applicant then gets all information on database.		
Flow of events		Actor Input	System Response

	1	User clicks to Icon “Avatar” in header and click “Account” in dropdown.	
	2		System shows all personal information on a form include: username, avatar, email, organization name.

Table 4. Use-case description for Update personal information

Use Case ID	**UC-04**		
Actor	User		
Brief description	This use-case for updating personal information.		
Pre-conditions	User already logged into the system.		
Post-conditions	Server received the request from Applicant then update new information into database.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks to Icon “Avatar” in header and click “Account” in dropdown.	
	2		System shows update personal information form.
	3	User fills the necessary information to the form.	
	4		System validates data and update data to the database.

Table 5. Use-case description for Update password

Use Case ID	**UC-05**
Actor	User
Brief description	This use-case for update password.
Pre-conditions	User already logged into the system.
Post-conditions	Server received the request from Applicant then update new password into database.

Flow of events		**Actor Input**	**System Response**
	1	User clicks to Icon “Avatar” in header and click “Account” in dropdown.	
	2		System shows update personal information form.
	3	User clicks “Change Password” Button.	
	4		System shows update password form.
	5	User fills the necessary new password to the form and submit form.	
	6		System validates data and update data to the database.

Table 6. Use-case description for Reset password

Use Case ID	**UC-06**		
Actor	User		
Brief description	This use case for resetting password.		
Pre-conditions	Trainee already logged out into the system		
Flow of events		**Actor Input**	**System Response**
	1	User clicks to “Forgot password” in login screen.	
	2		System shows the forgot password page.
	5	User fills the email into the form.	
	6		System validates data and sends mail to reset password.
	7	User enter mail and open system mail, then click button “Reset password” in the last line.	

	8		System navigates to reset password page
	9	User fills new password and confirm password and submit form	
	10		System login in account and navigate to the overview page

Table 7. Use-case description for creating form

Use Case ID	**UC-07**		
Actor	User		
Brief description	This use case is for user to creating form.		
Pre-conditions	User already logged into the system and went to the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	At the overview page. Click “Create form” button	
	2		A modal will display with two options: start from scratch or import form
	3	Click “Start from scratch”	
	4		System will create a form with two default elements: “Heading" and “Submit button" elements. And the navigate to the build form page for user to custom the form

Table 8. Use-case description for importing form

Use Case ID	**UC-08**
Actor	User
Brief description	This use case is for user to import the existing form in Google Form.

Pre-conditions	User already logged into the system and went overview page.		
Flow of events		**Actor Input**	**System Response**
	1	At the overview page. Click "Create form" button	
	2		A form will display with two options: start from scratch or import form
	3	Click "Import form"	
	4		System will show a form to fill Google form link
	5	User fill Google form link. Click "Create form"	
	6		System will validation and get form from Google Form and create a new form in system

Table 9. Use-case description for auto-generating form by chatbot

Use Case ID	**UC-09**		
Actor	User		
Brief description	This use case allows a user to use a chatbot to automatically create personalized content based on given prompts.		
Pre-conditions	User already logged into the system and went to the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	At the overview page. Click the botchat icon at the right-bottom corner.	
	2		A chat modal will display with the text "Hello, please provide your request so I can assist you

				in creating the form.” at default.
	3		User type the request in the input text that have the placeholder “Type a message...”	
	4			The chatbot uses OpenAI's language model to generate the requested form based on the prompt being provided, ensuring it aligns with the user's specifications. The chatbot delivers the generated form to the user for preview.
	5		User click to the “Preview" button to preview the form.	
	6			System will show a modal of form for user to preview.
	7			
		a.1	User satisfied with the form, click the “Create form" button.	
		b.1		System will create the form and provide for user a link to the build form page to view or edit the
		a.2	User not satisfied with the form, click the “Re-generate" button.	
		b.2		System will repeat the process until user is satisfied.

Table 10. Use-case description for Updating form

Use Case ID	**UC-10**		
Actor	User		
Brief description	This use case for updating form.		
Pre-conditions	User already logged into the system and went to the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	User drag and drop the elements provided in the left bar to the form container in the middle of the page or just easily click the element.	
	2		System adds the elements user dragged and dropped (or clicked) to the form and automatically save the form every time the form has changed

Table 11. Use-case description for user to View all forms

Use Case ID	**UC-11**		
Actor	User		
Brief description	This use case for viewing all form of user		
Pre-conditions	Users already logged into the system and went to the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	In the overview left bar, user click to the “All forms" button for view all form, or click to other folder button for the forms in related folder, or click to other team button for the form related team.	
	2		System shows list of forms based on user's request.

Table 12. Use-case description for user to view form detail

Use Case ID	**UC-12**		
Actor	User		
Brief description	This use case for showing form detail.		
Pre-conditions	Users already logged into the system and went to the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	At the form table, for each row, the user should click on "More" and then on "View".	
	2		System navigates to form detail page

Table 13. Use-case description for user to add form to favorite form

Use Case ID	**UC-13**		
Actor	User		
Brief description	This use case for adding form to favorite form		
Pre-conditions	Users have already logged into the system and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	At the form table, for each row, user click icon Star	
	2		The system saves this form to favorites. The empty star on this row will turn yellow.
	3	At side bar, click on “Favorites”	
	4		System GET all favorite forms

Table 14. Use-case description for adding form to folder

Use Case ID	**UC-14**		
Actor	user		
Brief description	This use case is for adding the form to the folder.		
Pre-conditions	Users already logged into the system and accessed the overview page. Server received the request to get all forms		
Flow of events		**Actor Input**	**System Response**
	1	At the form table, the user clicks on the "More" button for each row. Then, they select "Add to folder" from the menu. Alternatively, at the form table, user clicks on a checkbox and then clicks the “Add to folder” button on the top bar.	
	2		System shows a modal list all folder which form can be added
	3	The user selects a folder to add a form to and then clicks on "submit" to add the form to the chosen folder.	
	4		The system will add the form to the selected folder and assign a folder tag name to that form.

Table 15. Use-case description for moving form to team

Use Case ID	**UC-15**		
Actor	User		
Brief description	This use case for moving form to team		
Pre-conditions	Users already logged into the system and accessed the overview page. Server received the request to get all forms		
Flow of events		**Actor Input**	**System Response**
	1	At the form table, the user clicks on the "More" button for each row. Then, they select "Move to team" from the menu. Alternatively, at the form table, user clicks on a checkbox and then clicks the “Move to team” button on the top bar.	
	2		System shows a modal list all folder which form can be added
	3	The user selects a team to move a form to and then clicks on "submit" to move the form to the chosen team.	
	3		System will move that form to chosen team.
	4	User clicks on that team	
	5		System will show all form belong that team

Table 16. Use-case description for deleting form

Use Case ID	**UC-16**		
Actor	User		
Brief description	This use case for deleting forms.		
Pre-conditions	Users already logged into the system and went to the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	At the form table, the user clicks on the "More" button for each row. Then selects the "Delete" button from the menu. Alternatively, at the form table, user clicks on checkbox(s) displayed before the form(s) and then clicks the “Delete” button on the top bar.	
	2		System will move that form(s) to the trash section.

Table 17. Use-case description for purge form

Use Case ID	**UC-17**		
Actor	User		
Brief description	This use case for purging form.		
Pre-conditions	Users already logged into the system, went to the overview page and click to the “Trash" button in the overview left bar to be accessed to the trash section.		
Flow of events		**Actor Input**	**System Response**
	1	At the form table, the user clicks on the "Purge" button for each row. Alternatively, at the form table, user clicks on checkbox(s) displayed before the form(s) and then clicks	

		the “Purge” button on the top bar	
	2		System shows the modal to confirm the request, include two buttons: “Back” and “Confirm”.
	3.1	User clicks “Back” button.	
	4.1		System will close the confirm modal and not execute any actions.
	3.2	User clicks “Confirm” button.	
	4.2		System will delete that form(s) permanently.

Table 18. Use-case description for view shared forms

Use Case ID	**UC-18**		
Actor	User		
Brief description	This use case for viewing shared forms of user		
Pre-conditions	Users already logged into the system and went to the overview page.		
Post-conditions			
Flow of events	1	In the overview left bar, user click to the “Shared forms" button for view the shared forms.	
	2		System shows list of forms based on user's request.

Table 19. Use-case description for search form

Use Case ID	**UC-19**		
Actor	User		
Brief description	This use case for search form.		
Pre-conditions	Users already logged into the system and accessed the overview page. Server received the request to get all forms		
Flow of events		**Actor Input**	**System Response**
	1	Enter the form name in the input	
	2		The system displays forms based on the keywords entered and does not differentiate between uppercase and lowercase letters.

Table 20. Use-case description for sort form

Use Case ID	**UC-20**		
Actor	User		
Brief description	This use case for sort form		
Pre-conditions	Users already logged into the system and accessed the overview page. Server received the request to get all forms		
Flow of events		**Actor Input**	**System Response**
	1	The user clicks the menu button to the left of the Search form input. You can sort in ascending or descending order title, date created, last edit	
	2		The system will sort the form based on the sort type the user chooses

Table 21. Use-case description for sharing form with link

Use Case ID	**UC-21**		
Actor	User		
Brief description	This use case for sharing form with link.		
Pre-conditions	Users already logged into the system and accessed the build form page. The form has been saved successfully.		
Flow of events		**Actor Input**	**System Response**
	1	User choose the “Publish” button in the top bar.	
	2		System shows the “share with link" section in the
	3	User can click to the “Copy link" button to copy the link and send to other users to fill in the form or click to the “Open in new tab” button to open the form in public page.	

Table 22. Use-case description for set form status

Use Case ID	**UC-22**		
Actor	User		
Brief description	This use case for receiving email when the form has a new response.		
Pre-conditions	Users already logged into the system and accessed the form settings page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on “Form settings” on left bar	
	2		System navigates to form settings page.

	3	User set form status form at drop down input. User clicks on “Disable”	
	4		The system will update the form status; this form can no longer receive any responses.
	5	User clicks on “Enable”	
	6		The system will update the form status to indicate the receipt of responses.
	7	User clicks on “Disable on specific date”	
	8		The system will show time input to disable form on specific date
	9	The user specifies the exact date and time for disabling the form.	
	10		The user will set form schedule to disable form

Table 23. Use-case description for receiving email when the form has a new response

Use Case ID	**UC-23**		
Actor	User		
Brief description	This use case for receiving email when the form has a new response.		
Pre-conditions	Users already logged into the system and accessed the form settings page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on “Email” on left bar	
	2		System navigates to email setting page

	3	User clicks the switch to able to receive notification	
	4		The system will send an email to the form owner to announce the new response.
	5	User clicks again the switch to disable to receive notification	
	6		The form owner will not receive the new response unless they log into the system.

Table 24. Use-case description for preview form

Use Case ID	**UC-24**		
Actor	User		
Brief description	This use case for preview form.		
Pre-conditions	Users already logged into the system and accessed the edit form page		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on “Preview form” button	
	2		System shows form to preview form.
	3	User fills the form to answer form	
	4		System will validate data but will not create a response.

Table 25. Use-case description for preview form

Use Case ID	**UC-25**		
Actor	User		
Brief description	This use case for preview form.		
Pre-conditions	Users already logged into the system and accessed the build form page		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on “Preview form” button	
	2		System shows form to preview form.
	3	User fills the form to answer form	
	4		System will validate data but will not create a response.

Table 26. Use-case description for delete folder

Use Case ID	**UC-26**		
Actor	User		
Brief description	This use case for delete folder.		
Pre-conditions	User who has account to access to application and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on the icon on the right folder name. Then, the user clicks “Delete” at the menu.	
	2		System shows modal to confirm deleting the folder.
	3	User clicks on “Confirm”.	
	4		System deletes the folder.

Table 27. Use-case description for create team

Use Case ID	**UC-27**
Actor	User
Brief description	This use case for create team.

Pre-conditions	User who has account to access to application and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on “Create a new team” button on the left bar.	
	2		System shows form to enter folder name.
	3	User enter the folder name and submit	
	4		System will create a new team.

Table 28. Use-case description for update team name

Use Case ID	**UC-28**		
Actor	User		
Brief description	This use case for update team name.		
Pre-conditions	User who has account to access to application and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on the icon on the right team name. Then, user clicks on “Change name” on the menu.	
	2		System shows form to change folder name.
	3	User change the folder name and submit.	
	4		System will update the team name.

Table 29. Use-case description for view folder of team

Use Case ID	**UC-29**
Actor	User
Brief description	This use case for view folder of team name.

Pre-conditions	User who has account to access to application and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on the icon "^" on the right team name.	
	2		System shows folders of team.

Table 30. Use-case description for view all forms of team

Use Case ID	**UC-30**		
Actor	User		
Brief description	This use case for view all forms of team.		
Pre-conditions	User who has account to access to application and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on team name	
	2		System shows forms of team.

Table 31. Use-case description for add folder to team

Use Case ID	**UC-31**		
Actor	User		
Brief description	This use case for add folder to team.		
Pre-conditions	User who has account to access to application and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	User clicks on the icon on the right team name. Then, user clicks on "Add new folder" on the menu.	
	2		System shows form to enter folder name.
	3	User enter the folder name and submit.	

	4		System will create folder to team.

Table 32. Use-case description for view all responses

Use Case ID	**UC-32**		
Actor	User		
Brief description	This use case for view all responses.		
Pre-conditions	User who has account to access to application and accessed the overview page.		
Flow of events		**Actor Input**	**System Response**
	1	At form table, for each form, clicks on "[amount] submissions"	
	2		System shows responses of form.

Table 33. Use-case description for create response

Use Case ID	**UC-33**		
Actor	User		
Brief description	This use case for create response.		
Pre-conditions	User who has account to access to application and accessed the public page.		
Flow of events		**Actor Input**	**System Response**
	1	User fills answers and clicks submit button	
	2		System will validate data and create a response.

2.4. Sequence diagrams

2.4.1 Sequence diagrams for user to login to system

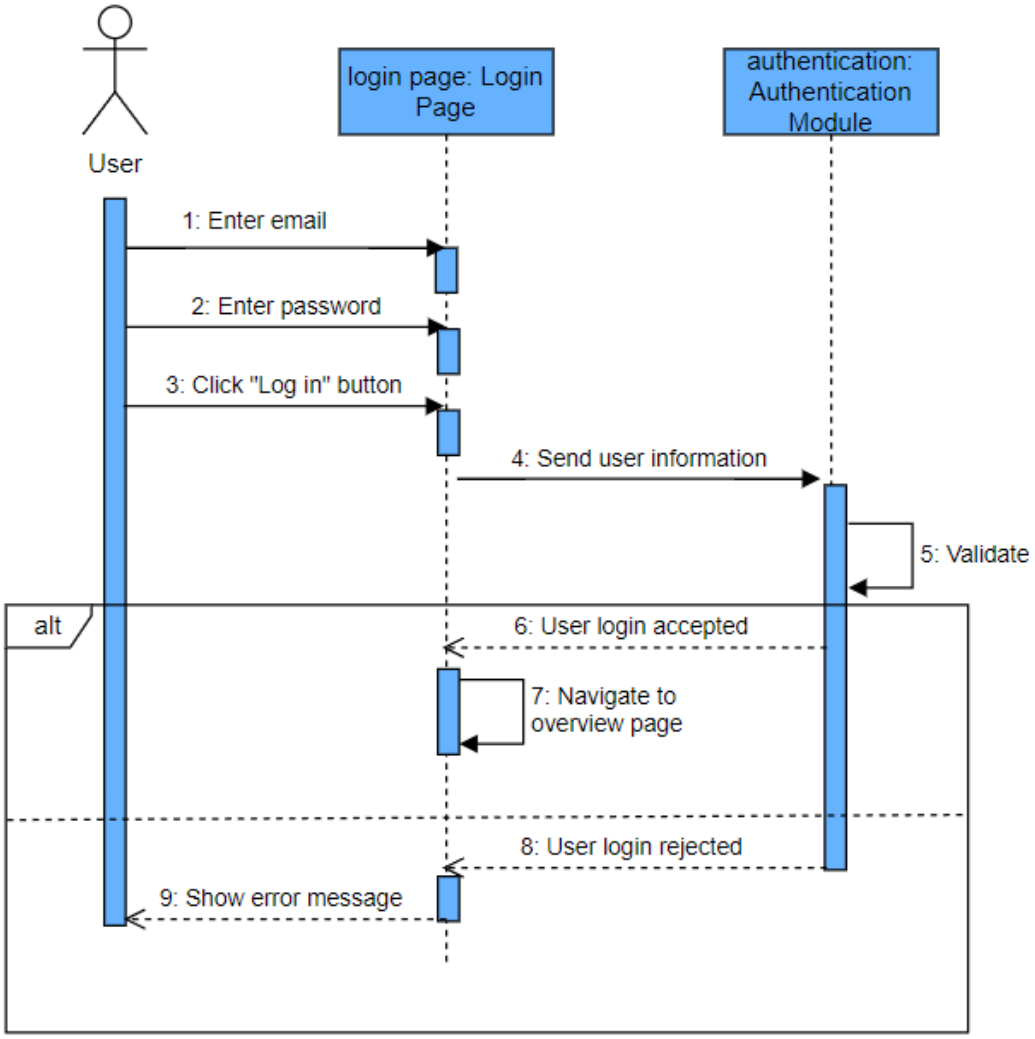

Figure 14. Sequence diagram for login

2.4.2 Sequence diagram for user to import form from Google Form

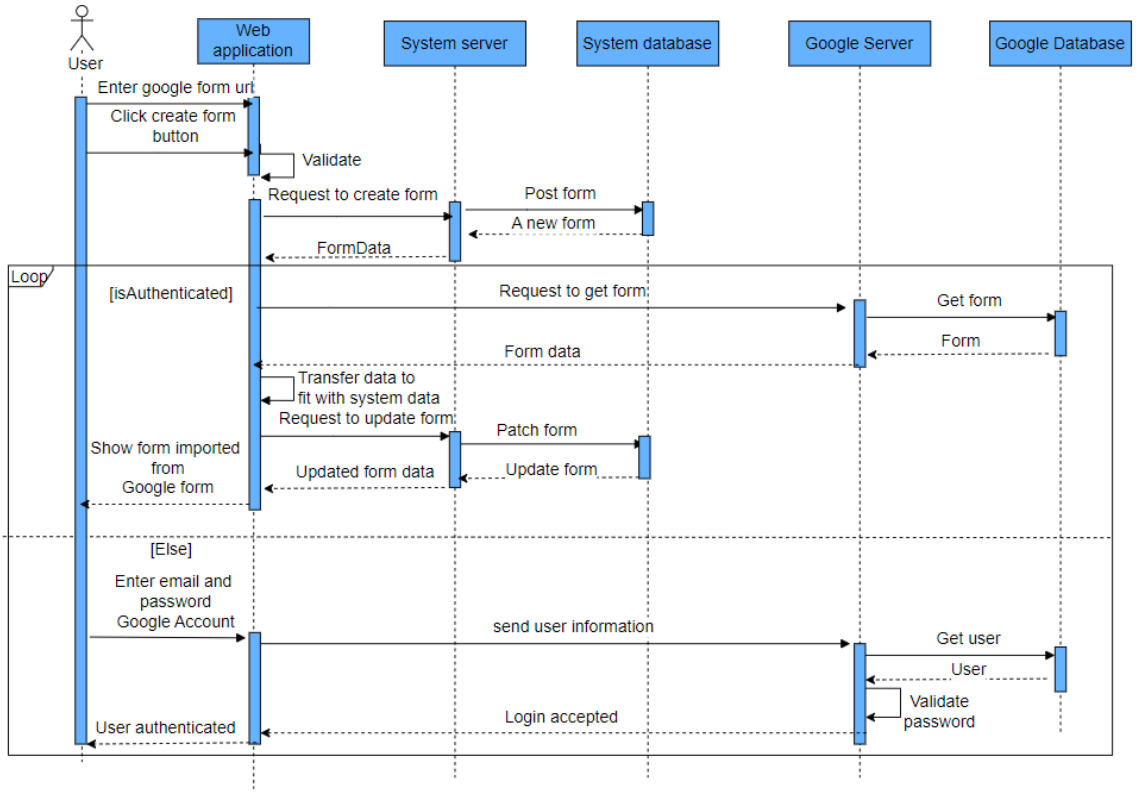

Figure 15. Sequence diagram to import form from Google Form

2.4.3 Sequence diagram for user to auto-generating form by chatbot

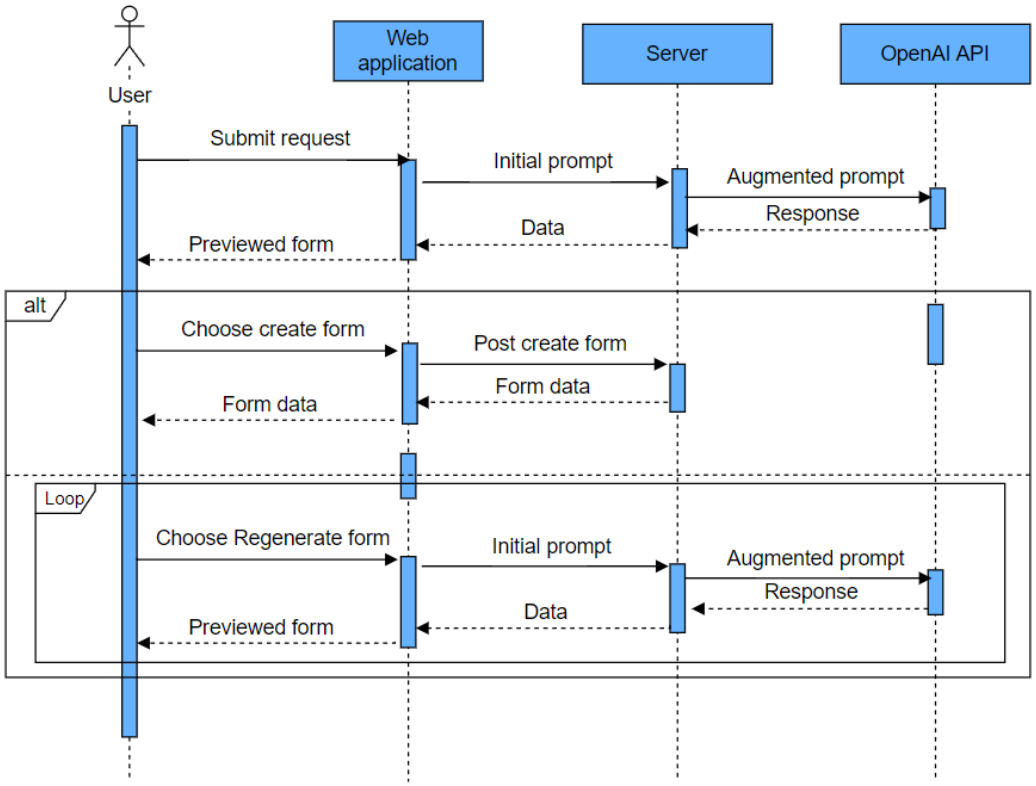

Figure 16. Sequence diagram to auto-generate form by chatbot

2.4.4 Sequence diagram for user to enable form on specific date

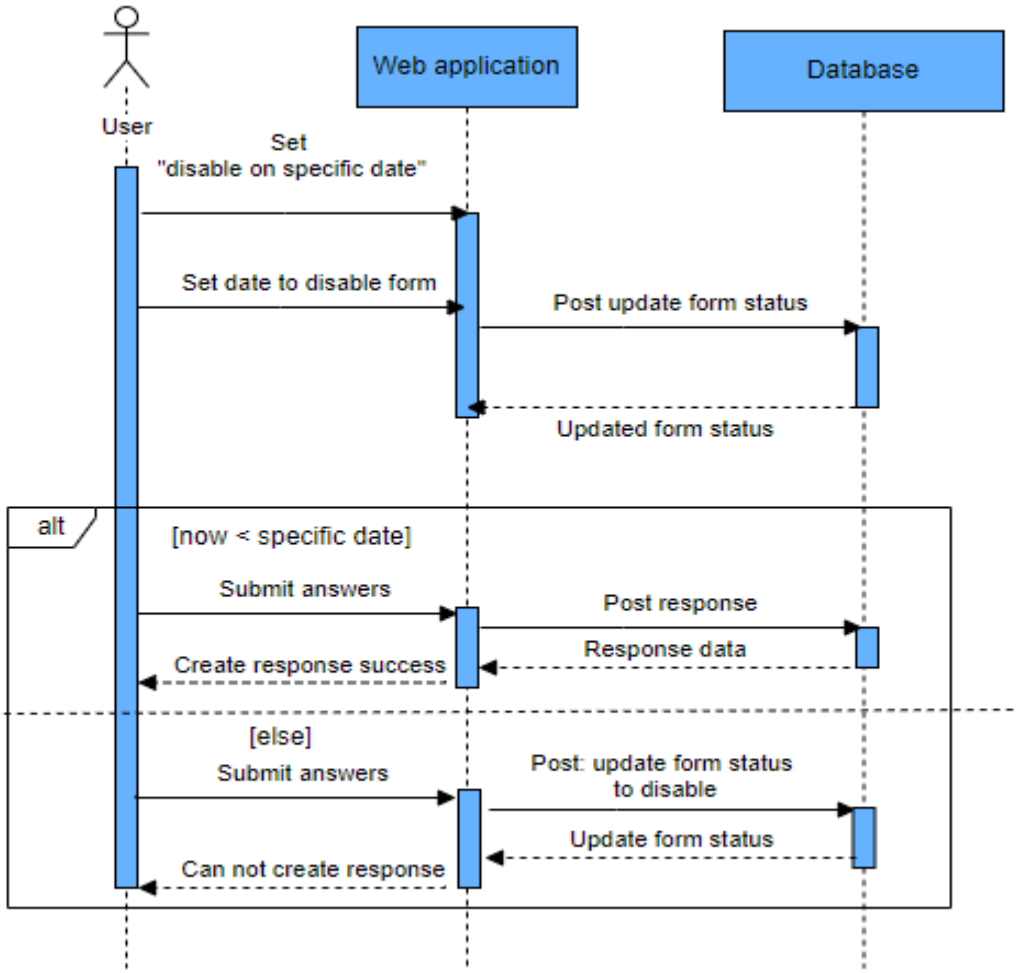

Figure 17. Sequence diagram for user to enable form on specific date

2.5. Database design

2.5.1 Database design model

The system database will include the following tables:

- User: table to save user's profile information in system.
- Form: table to save form's information in system.
- Folder: table to save folder's information in system.
- Team: table to save team's information in system.
- Response: table to save response's information in system.

2.5.2 Relationship between table

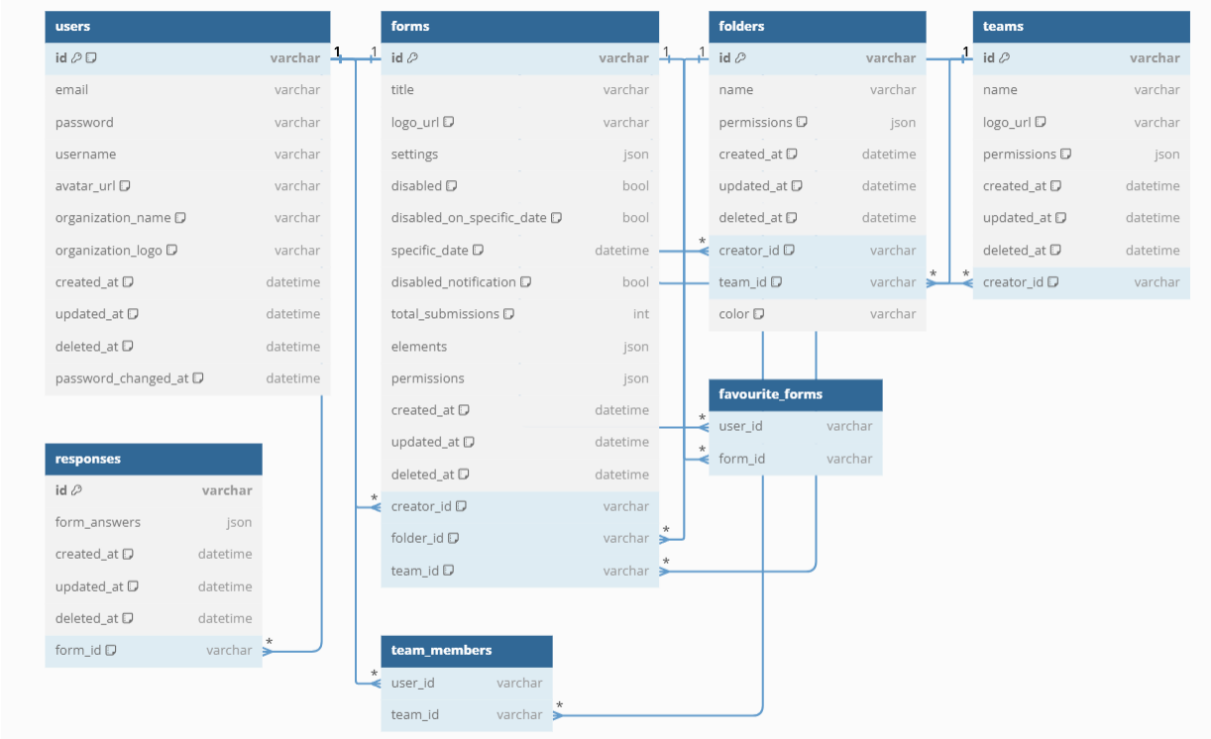

Figure 18. Database design

2.6. Conclusion

This chapter presented the requirements specification that the system could meet the user' demands. Follow the requirements, the use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. The action sequences as well as the interactions between user and user, user and system. Thereby, the overview and the activity streams of the system are fully presented.

Besides, it also describes the system structure, as well as the action sequences for each function. By that, it facilitates the testing phase, the tester can go back the sequence diagrams to follow the action sequences and create the function tests and the input data as well. Furthermore, it shows the database and some UI application designs to help the reader have clearer views of the system. By that, we can consider and evaluate the complexity of this system.

Chapter 3: IMPLEMENTATION AND EVALUATION

3.1. Development environment

Project github link: https://github.com/PBL7-2024-FormMaker

3.1.1 Web service

The web service is a RESTful API. It is built by NodeJS and Express, based on representational state transfer (REST) technology. I use PostgreSQL to store data.

3.1.2 Software Development Tools

- Programming languages and framworks:
    - Back-end: ExpressJS (TypeScript)
    - Front-end: ReactJS, TypeScript, Tailwind
- Database: PostgreSQL & Prisma
- Manage resource: Git, Github
- Deployment:
    - Front-end: Netlify
    - Back-end: Koyeb
    - Database: RDS service
- Manage image and file: Cloudinary
- Test API: Postman

3.1.3 Setup to send email

- Library use: Nodemailer
- Approach to send email with Nodemailer:
    - Include the nodemailer module in the code using require('nodemailer').
    - Use **nodemailer.createTransport()** function to create a transporter who will send mail. It contains the service name and authentication details (user ans password).
    - Declare a variable mailDetails that contains the sender and receiver email id, subject and content of the mail.
    - Use mailTransporter.sendMail() function to send email from sender to receiver. If the message being sent failed or contains error then it will display error message otherwise message send successfully.
- Below is basic example to show how to send mail using nodejs

```
const nodemailer = require('nodemailer');

let mailTransporter =
    nodemailer.createTransport(
        {
            service: 'gmail',
            auth: {
                user: 'xyz@gmail.com',
                pass: '************'
            }
        }
    );

let mailDetails = {
    from: 'xyz@gmail.com',
    to: 'abc@gmail.com',
    subject: 'Test mail',
    text: 'Node.js testing mail for GeeksforGeeks'
};

mailTransporter
    .sendMail(mailDetails,
        function (err, data) {
            if (err) {
                console.log('Error Occurs');
            } else {
                console.log('Email sent successfully');
            }
        });
```

Figure 19: Setup to send email

3.1.4 Get an Oauth 2.0 Client ID

To import form from Google form, I will get an OAuth 2.0 Client ID for my project, following these steps:

1. Go to the Google Cloud Console:
    - Open Google Cloud Console.
2. Create a New Project (if you don't have one already):
    - Click on the project drop-down menu at the top of the page.
    - Select "New Project".
    - Enter your project name and other details.
    - Click "Create".
3. Enable the Google Forms API:
    - Navigate to the API & Services dashboard.
    - Click "Enable APIs and Services"
    - Search for "Google Forms API" and select it.
    - Click "Enable".
4. Configure OAuth Consent Screen:

- Go to the "OAuth consent screen" tab.
- Select "External" if your app will be used by users outside of your organization.
- Click "Create".
- Fill out the required fields (App name, User support email, etc.).
- Add your email address to "Developer contact information".
- Click "Save and Continue".

5. Create OAuth 2.0 Credentials:
    - Go to the "Credentials" tab.
    - Click "Create Credentials" and select "OAuth Client ID".
    - Configure the consent screen if prompted.
    - Select "Application type". For a web application, choose "Web application".
    - Fill in the required fields:
        - Name: Name your OAuth 2.0 client.
        - Authorized redirect URIs: Add the redirect URL where users will be sent after they authorize with Google. This should be the URL that your app will handle the OAuth 2.0 callback.
    - Click "Create".
6. Get Your OAuth 2.0 Client ID and Secret:
    - After creating the credentials, you will see a dialog with your Client ID and Client Secret.
    - Note down these credentials as you will need them for your application.
7. Add Your Credentials to Your Application:
    - Use the Client ID and Client Secret in your app's OAuth 2.0 configuration.
    - Here is an example of how you might use these credentials in a Python application:

Here is an example to init client:

```
const initClient = () => {
  gapi.client
    .init({
      apiKey: API_KEY,
      clientId: CLIENT_ID,
      discoveryDocs: DISCOVERY_DOCS,
      scope: SCOPES,
    })
    .then(
      () => {
        const authInstance = gapi.auth2.getAuthInstance();
        authInstance.isSignedIn.listen(updateSigninStatus);
        updateSigninStatus(authInstance.isSignedIn.get());
      },
      (error: GoogleFormResponse) => {
        setContent(error.result);
      },
    );
};

gapi.load('client:auth2', initClient);
```

Figure 20: Get an Oauth 2.0 Client ID

3.1.5 OpenAI

- To use OpenAI's models in a project, I follow these steps:
    - **Installation**: Sign up for API access and obtain an API key from OpenAI. Install any necessary dependencies for making API requests.
        - Run this command in the project to install the package for OpenAi Node API: “npm install openai”

- **Setup**: Configure the application to include the OpenAI API key and set up any required API endpoints.

```
const openai = new OpenAI({
  apiKey: OPENAI_API_KEY,
});
const completion = await openai.chat.completions.create({
  model: OPENAI_MODEL,
  max_tokens: 1024,
  temperature,
  messages: [
    {
      role: 'system',
      content: systemPrompt,
    },
    {
      role: 'user',
      content: userPrompt,
    },
  ],
});
```

Figure 21. Init the OpenAI and using the chat completions API

- System prompt is the prompt is a set of instructions designed for the application. The prompt outlines the structure and requirements for these form elements, guiding the assistant on how to properly format and respond to user requests.

```
export const systemPrompt = `
You are Fo, an assistant for Formmaker. You will be provided a list of elements with their purpose and configs.
Your job is to generate a complete form based on the specified request of users. The form should have a title related to its purpose, an empty settings object, and a list of elements.
If the user asks a question unrelated to creating a form, respond with "Invalid question. Please ask about creating a form."
The provided elements will be in the format:
\`\`\`
[
    {
        "Element Name" : element purpose
        config: element config
    }
]
\`\`\`
Response will be in JSON format, which is a complete form with a title related to the form's purpose, settings, and a list of elements.

list of elements:
[
    {
        Email: For inputting the email address
        config: {
            fieldLabel: element title (required),
            required: indicates if the email is mandatory (required),
            sublabel: sub-label for email input field (optional, default value: 'Type your email address'),
        }
    },
    {
        Fullname: For inputting the name
        config: {
            fieldLabel: question relates to name (required),
            required: indicates if the full name is mandatory (required),
            sublabels: {
                firstName: sublabel for the first name input (optional, default value: 'First Name'),
                lastName: sublabel for the last name input (optional, default value: 'Last Name')
            }
        }
    },
```

Figure 22. Example of the system prompt

- User prompt is the request of user to pass in the function to help users generate complete forms based on the that.

- **Usage**: Make API calls to OpenAI's endpoints, passing in prompts and other parameters to generate responses.

```
export const openAiApi = rootApi.injectEndpoints({
  endpoints: (build) => ({
    getElementsFromQuestion: build.mutation({
      query: ({ questions }: { questions: string }) => ({
        url: `${API_URL.OPEN_AI}/${API_URL.GET_QUESTIONS}`,
        data: { questions: questions },
        method: 'POST',
      }),
    }),
  }),
  overrideExisting: false,
});
```

Figure 23. API calls to the OpenAI's endpoint

- For instance, when building a feature-rich chatbot that includes auto form creation by prompting, we can leverage OpenAI's capabilities to interpret user inputs and dynamically generate forms based on the context of the conversation.

3.1.6 Cloudinary

- In my project, I use cloudinary to manage image and file. To get started with Cloudinary in your Node.js application, follow these steps:
  - Install the Cloudinary Node Module
  - Import and Configure Cloudinary in Node.js Application: need to configure it with Cloud Name, API Key, and API Secret
  - Uploading Images

3.2. Demo main features and evaluation

Figure 24. Login screen

The user enters an email and password to login to the system

Figure 25. Signup Screen

The user registers an account by entering a username, email, password, and confirming the password.

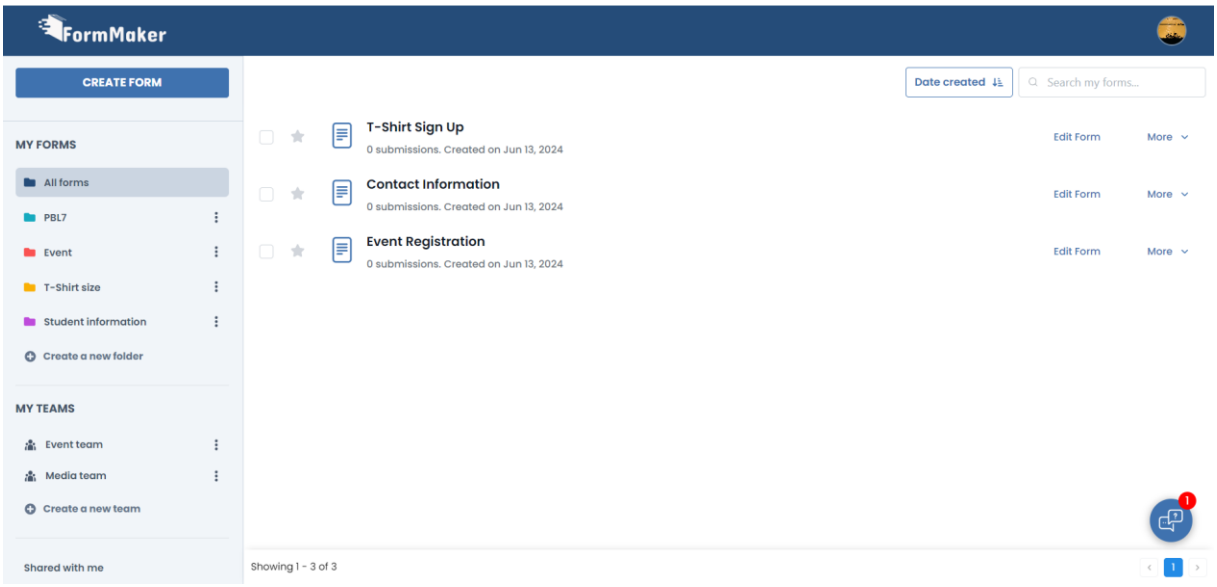

Figure 26. Hompage screen

When the user successfully logs in, the system will redirect to the homepage. Here, the user will see an overview of all the forms, folders, and teams being managed. The user can quickly search for and sort forms. Notably, the user can create a form by clicking the “Create form” button or by clicking the chatbot icon in the bottom right corner of the screen to automatically create a form based on the user's requirements.

The advantage of this feature is that the user has an overview of the forms they are managing, with a clear and organized storage system, making it easy to manage each form.

The disadvantage of this feature is that having many actions on one screen can make it difficult for users to remember.

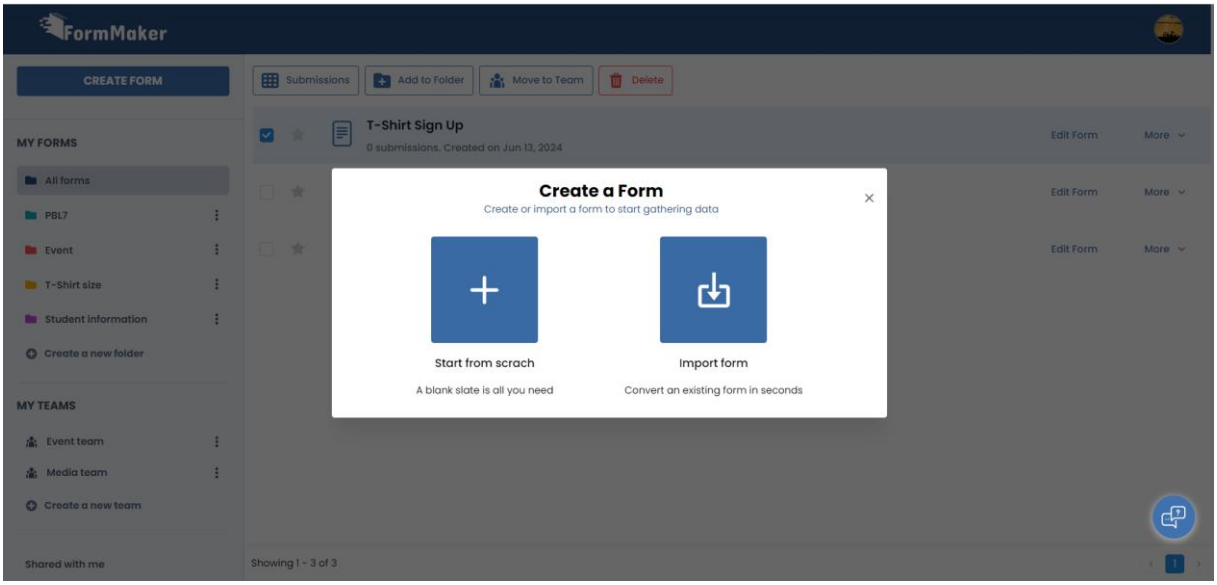

Figure 27. Create form

When the user selects the "Create form" button, a modal will appear to choose the method of form creation. The system provides three ways to create a form:

- Start from scratch: A blank slate is all you need.
- Import form: Insert a form from Google Forms.
- Chatbot (at the bottom right corner of the screen): Create a form based on user requests, meaning the system will automatically generate a form based on the user's natural language input.

Advantage: The system offers a variety of methods for creating forms, depending on the context the user wants to use.

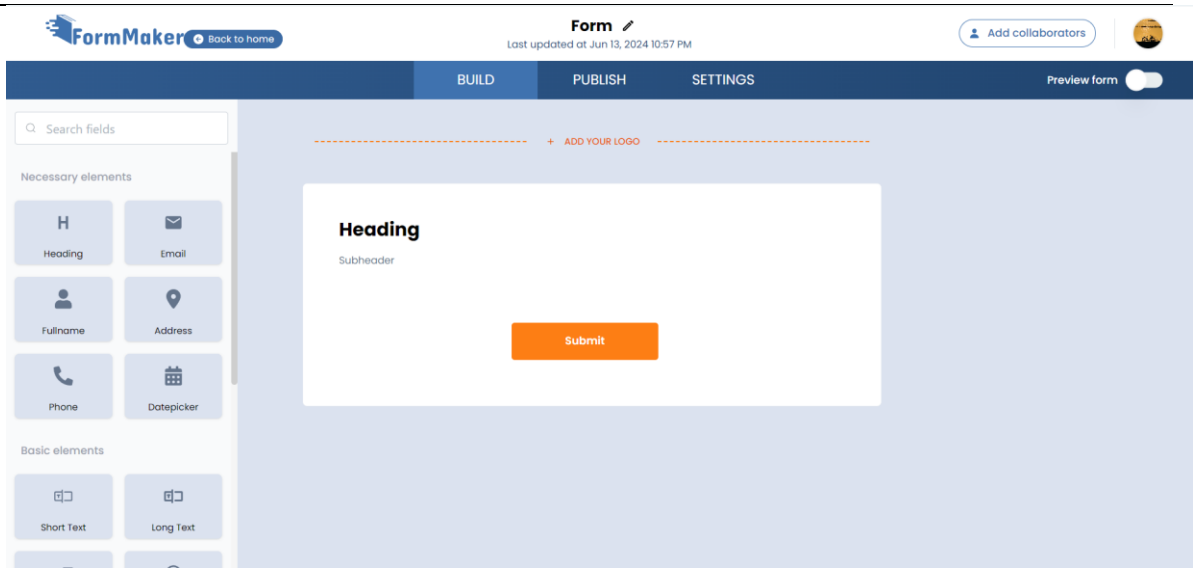

Figure 28. Create form from scratch screen

When selecting "Start from scratch," the system will redirect to the form creation page. A form includes: Form Name, Form Logo, and Elements.

The default form name is "Form." The user can click the edit button to modify the form name. The user can select "Add your logo" and choose an image to set as the form logo.

The system provides a variety of elements for the user, such as:

- **Heading**: Used to create a heading and description for the form.
- **Email**: Used to collect email addresses.
- **Full Name**: Used to collect first name and last name.
- **Address**: Collects information about street, ward, district, and city.
- **Phone**: Collects phone numbers (only Vietnamese phone numbers).
- **Datepicker**: Collects dates.
- **Short Text**: Collects short information in one line; the user cannot press enter to move to a new line.
- **Long Text**: Collects longer information; the user can press enter to move to a new line.
- **Dropdown**: Allows selecting one option from many options.
- **Single Choice**: Allows selecting one option from many options.
- **Multiple Choice**: Allows selecting multiple options.
- **Time**: Collects time in the format Hour: Minute.

- **Image**: Inserts an image into the form to describe a specific context.
- **File Upload**: Allows users submitting the form to upload files.
- **Scale Rating**: Collects rating scores from 1 to 5.

The user can drag and drop the necessary elements into the container.

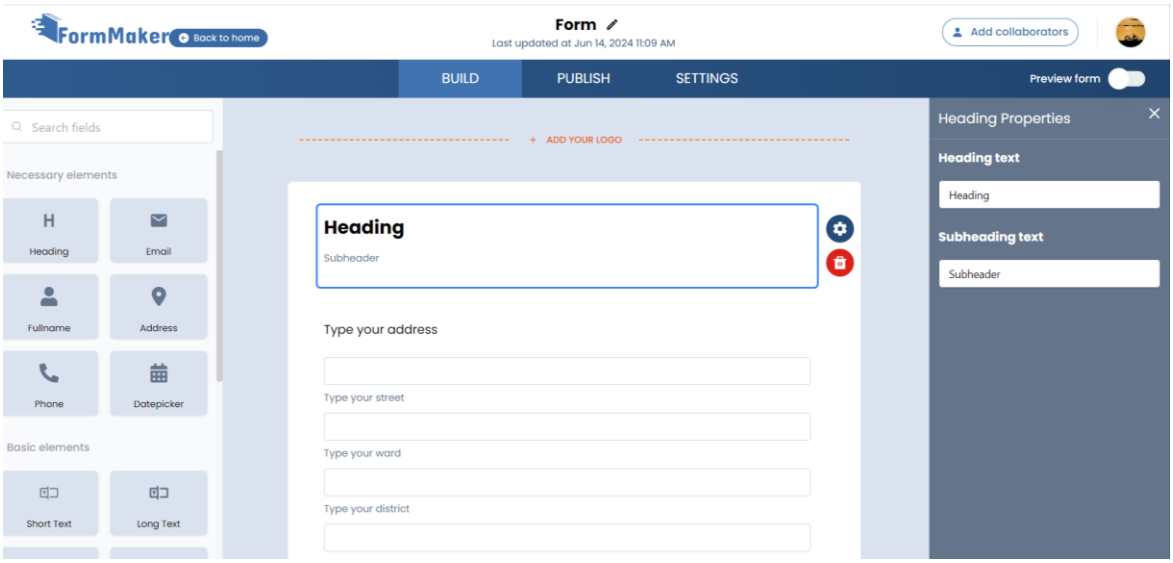

Figure 29. Edit elements screen

The user clicks the settings icon on the right of each element to change the title or properties of each element, which are displayed in the right sidebar. This applies similarly to other elements.

The advantage of this feature is that the system allows real-time creation. Any property changes made to the form are saved immediately without needing to press save. Additionally, the system provides a complete set of elements for the user, saving time in designing elements.

The system validates the property data of each element. Therefore, users cannot leave any field empty in the right sidebar, ensuring users are reminded to complete the form fully.

The disadvantage is that it may not fully meet the user's needs in certain cases where a property is unnecessary. For example, the Heading element requires both a Heading text and Subheading text. If the user only wants a heading without a subheading, the system will not allow it and will display an error, requiring the user to enter a subheading to successfully create the form.

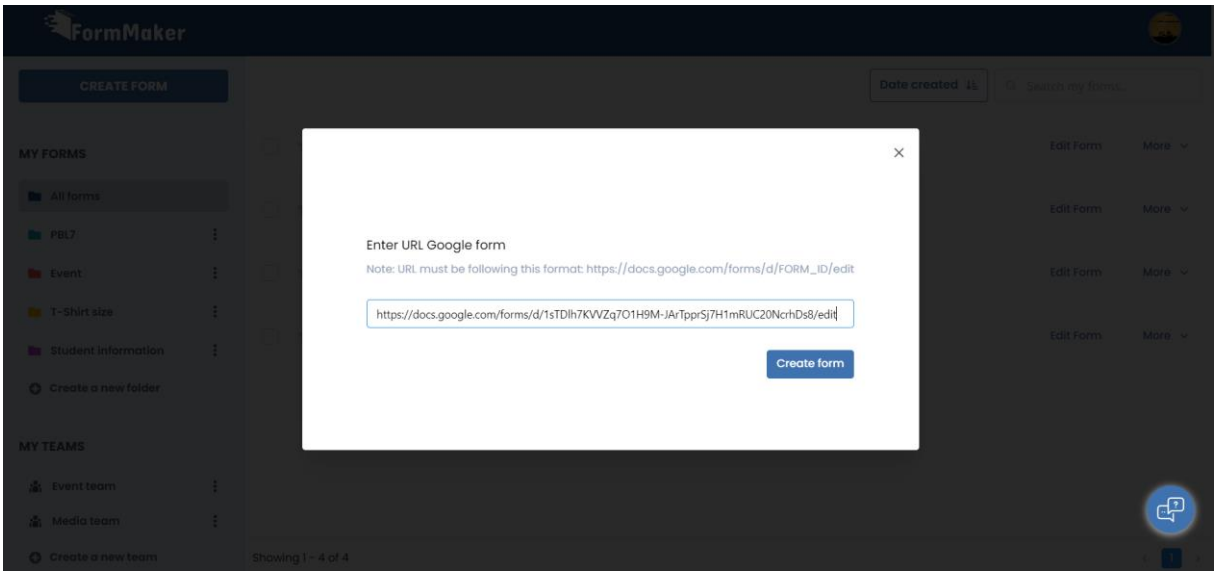

Figure 30. Import form screen

The user must enter/paste the Google Forms link in the correct format: https://docs.google.com/forms/d/FORM_ID/edit to proceed to the next step. After entering the link, the system will require you to authenticate the account that authorized the form. The user logs in with their authorized Google account, and once authentication is successful, the form will be successfully imported. For subsequent imports, the user will not need to authenticate again; they only need to enter the link, and the system will return the imported form.

The advantage of this feature is that it allows the user to quickly create a form from an existing one without having to do it manually, saving time and effort.

The disadvantage of this feature is that the user must navigate to the Google Forms platform, copy the link, and paste it into the system, which poses a UX issue.

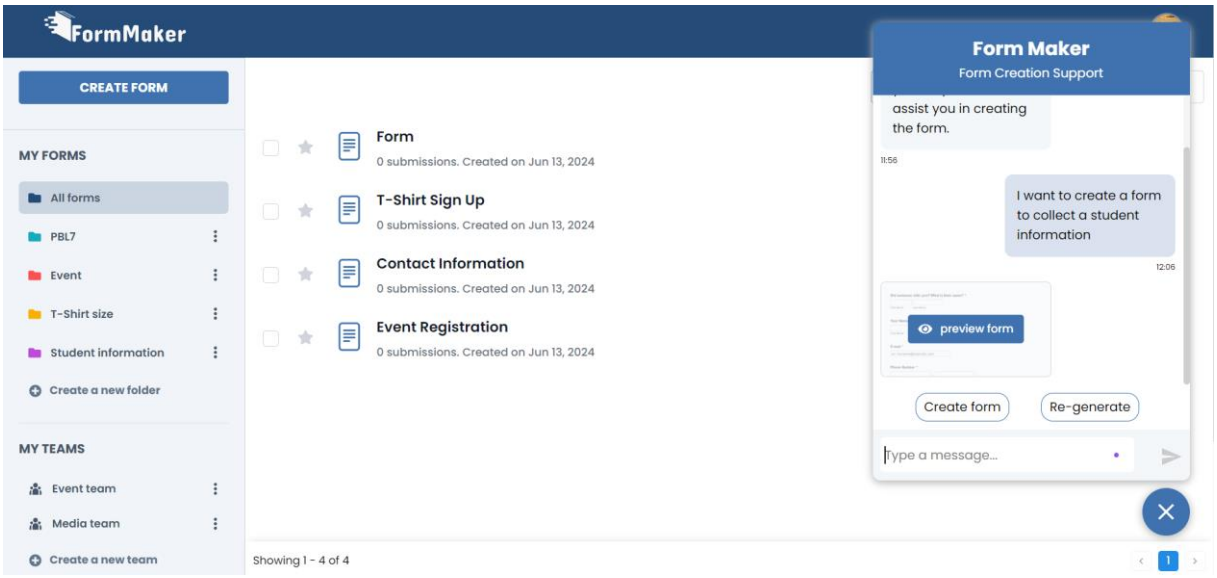

Figure 31. Auto-generate form by chatbot

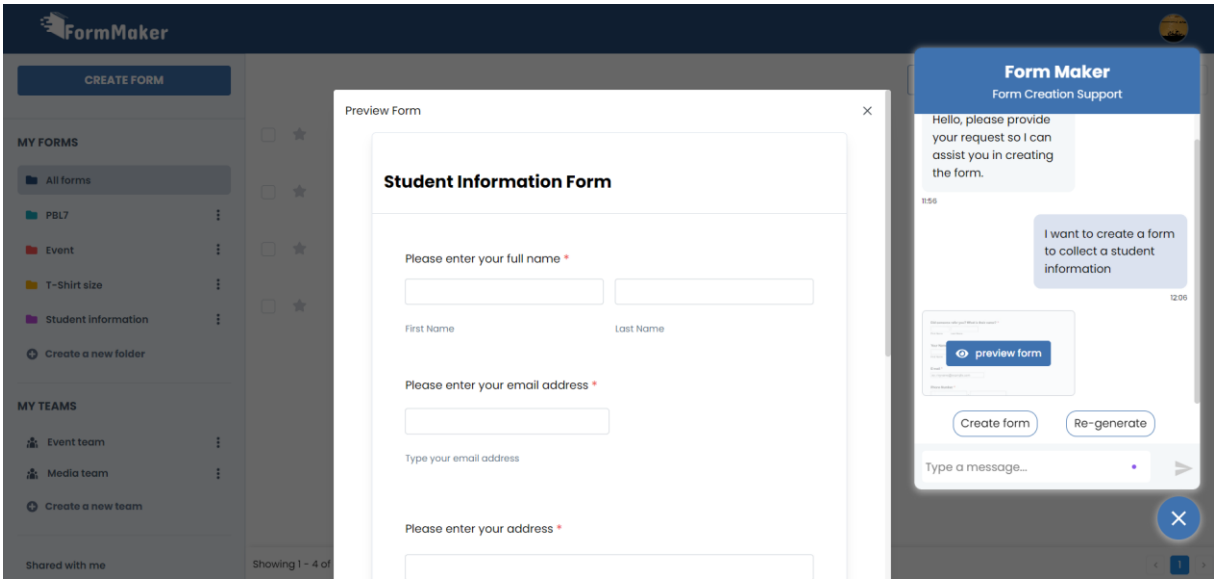

Figure 32. Chatbot allow preview form screen

The user enters their request into the chat and presses enter. The system will return a form based on the user's request. The user can preview the form to decide whether to create it or not.

The advantage of this use case is that it is suitable for users who have a request to create a form but are unsure of the form's components. This feature helps users save time and generate ideas. The system provides results that are relatively accurate to the user's initial request.

The disadvantage of this feature is that the chatbot only has the functionality to create forms automatically and does not have the capability to edit an existing form based

on user requests. Secondly, the chatbot cannot respond based on the context of multiple user requests but only based on a single request at a time.

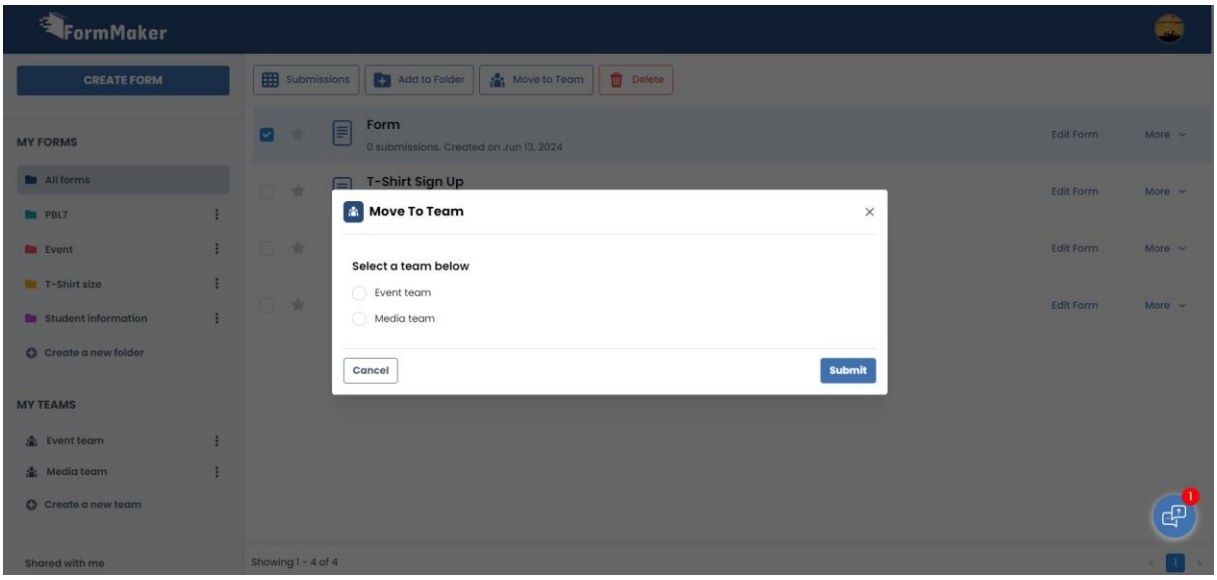

Figure 33. Move form to team screen

The user selects a team to move a form to the desired team.

The advantage of this feature is that it makes form storage more flexible.

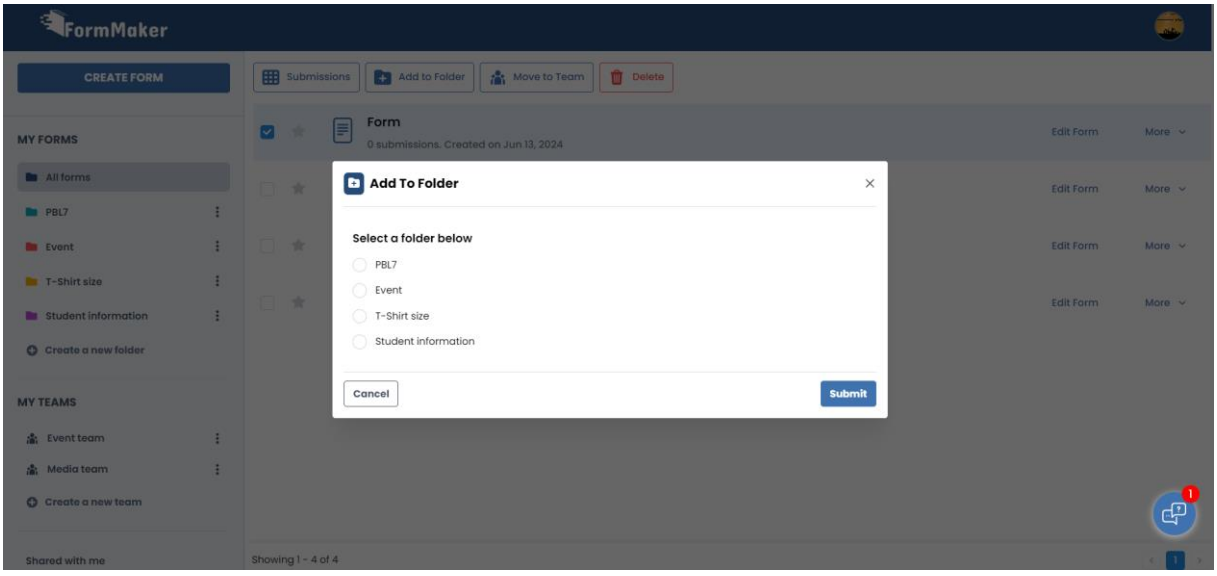

Figure 34. Add form to folder screen

The user selects a folder to move a form to the desired folder.

The advantage of this feature is that it makes form storage more flexible.

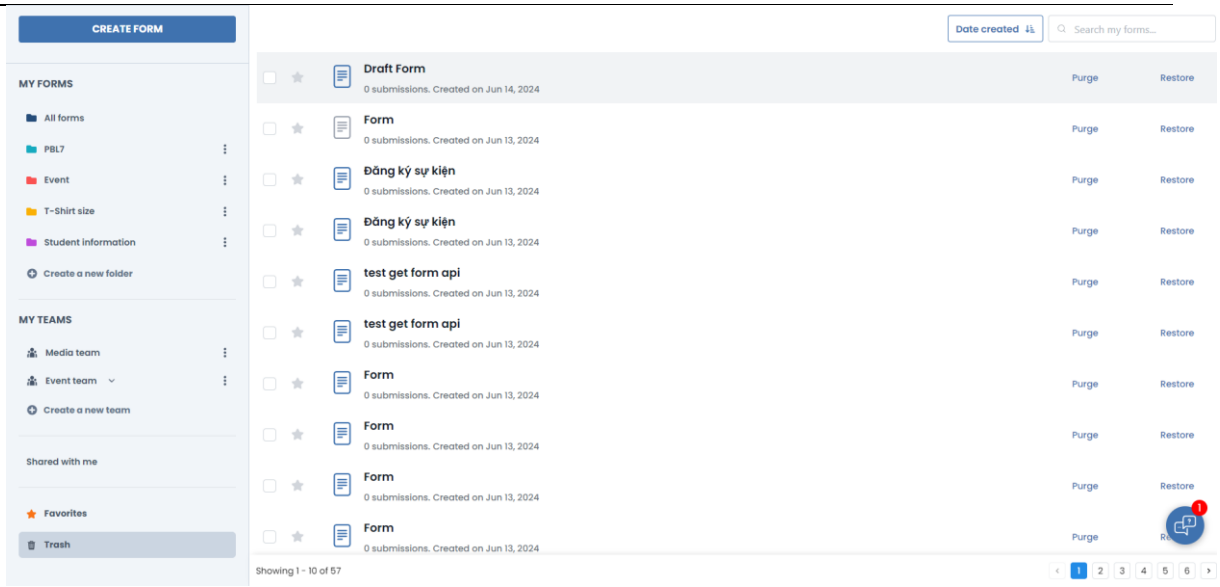

Figure 35. View Trash form

When the user presses "delete form," the form will not be permanently deleted from the system but will be stored in the draft forms folder.

Advantages: This feature helps users clear out forms that are rarely used. If a form is accidentally deleted, there is still a place to store forms for restoration.

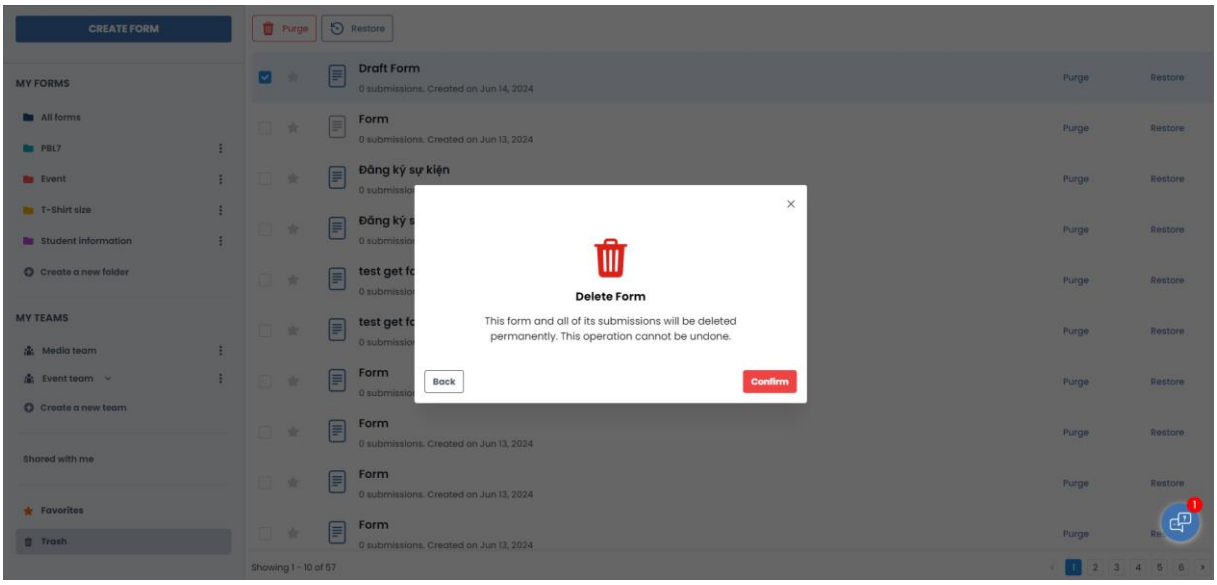

Figure 36. delete form permanently screen

The user presses the "Confirm" button to delete the form permanently.

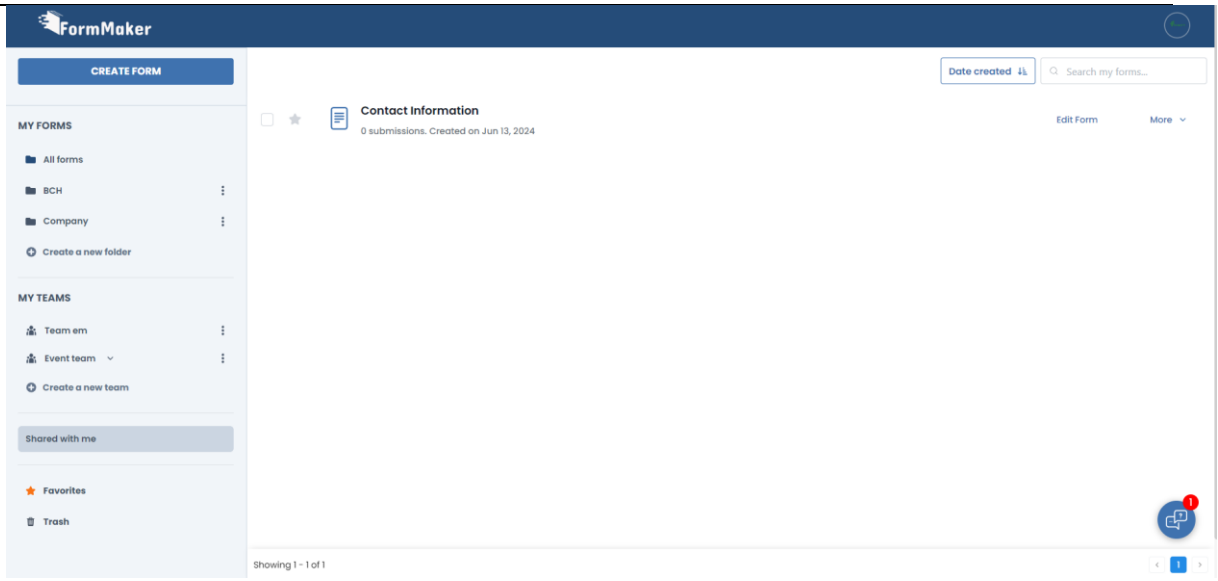

Figure 37. View share with me screen

The user clicks on the "Share with me" button to see forms that have been shared by others.

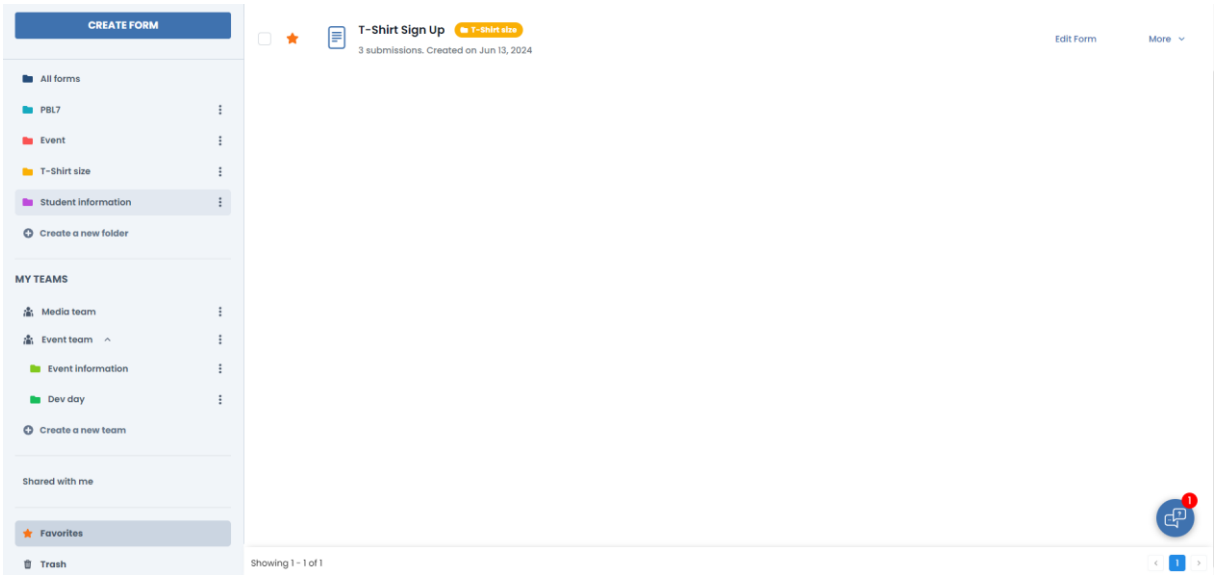

Figure 38. View favorite form screen

The user clicks on the "Favorite" button to view favorite forms.

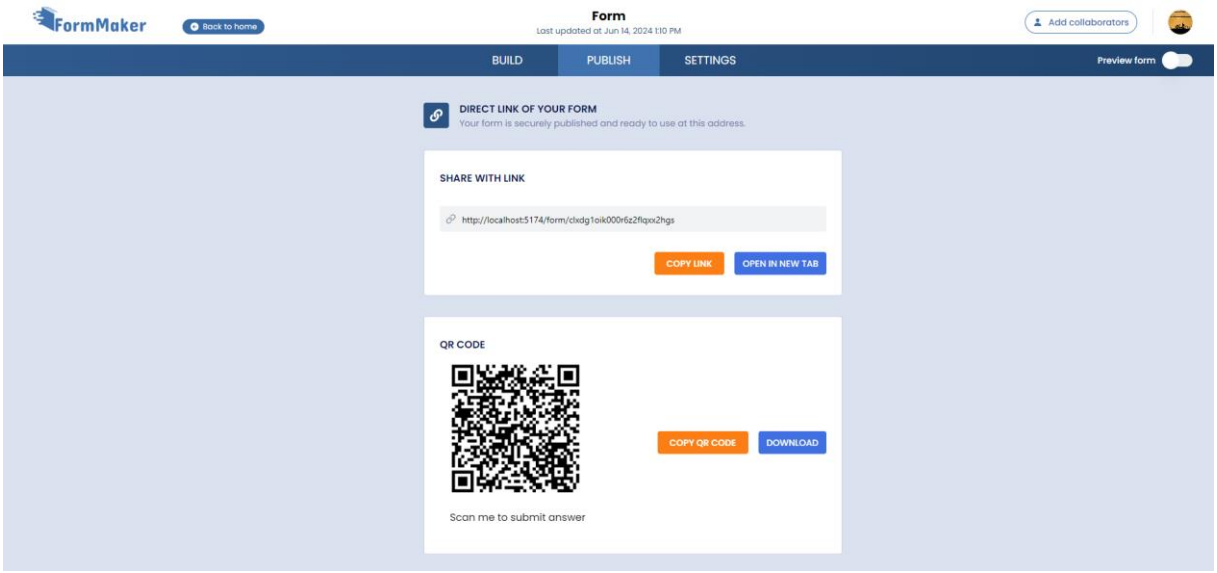

Figure 39. Publish form screen

Users can share a form for others to submit responses in two ways: via a link or a QR code.

- **Share with link**: Users click "Copy link" and then "Open new tab" to open the form in a new tab.
- **Share with QR code**: Users click "Copy QR code" or "Download" to save the QR code image for sharing with others.

Advantages: This feature makes it easy for others to access the form and submit responses. Sharing the form with a QR code enhances convenience for respondents who can submit using their mobile phones.

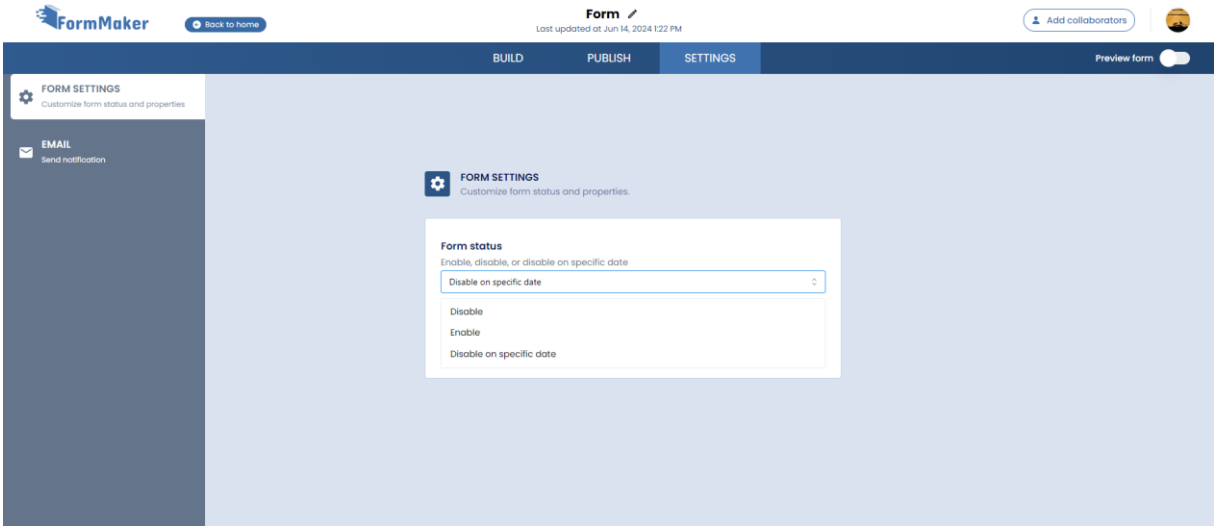

Figure 40. Form settings screen

A form has 3 states: Disabled, Enabled, and Disable on specific date.

- **Disabled**: The form cannot receive responses.
- **Enabled**: The form can receive responses.
- **Disable on specific date**: The form will be unable to receive responses at a specific date and time.

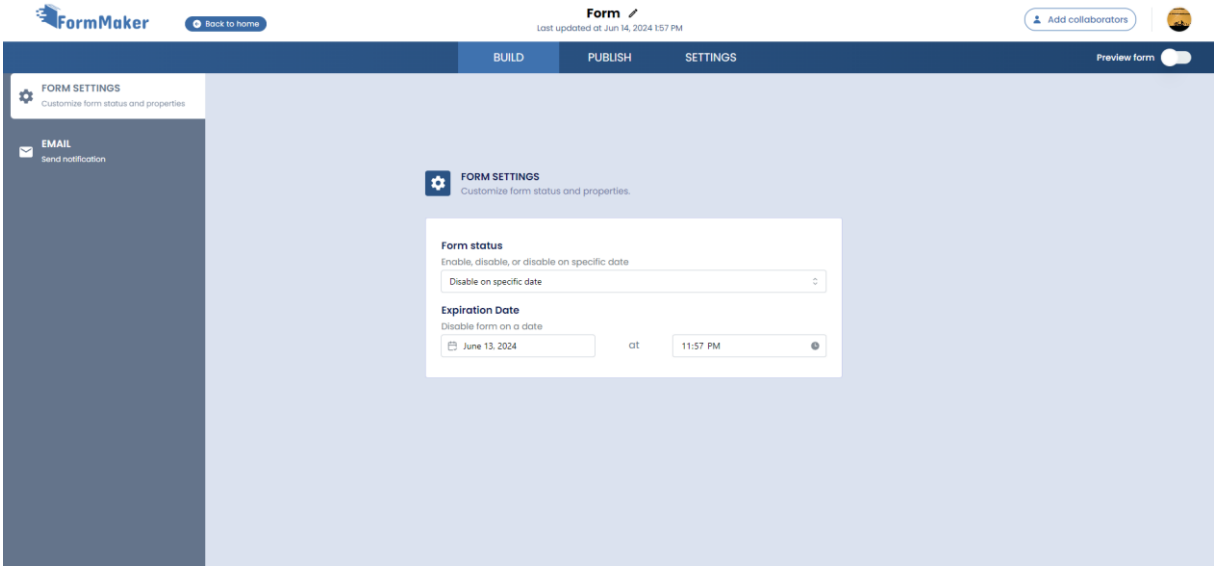

Figure 41. Set disable on specific date screen

Users select a date to disable the form. The default time is the form creation time plus 1 hour.

The advantage of this feature is that it allows users to set a deadline for forms, automating the process and saving users time and effort.

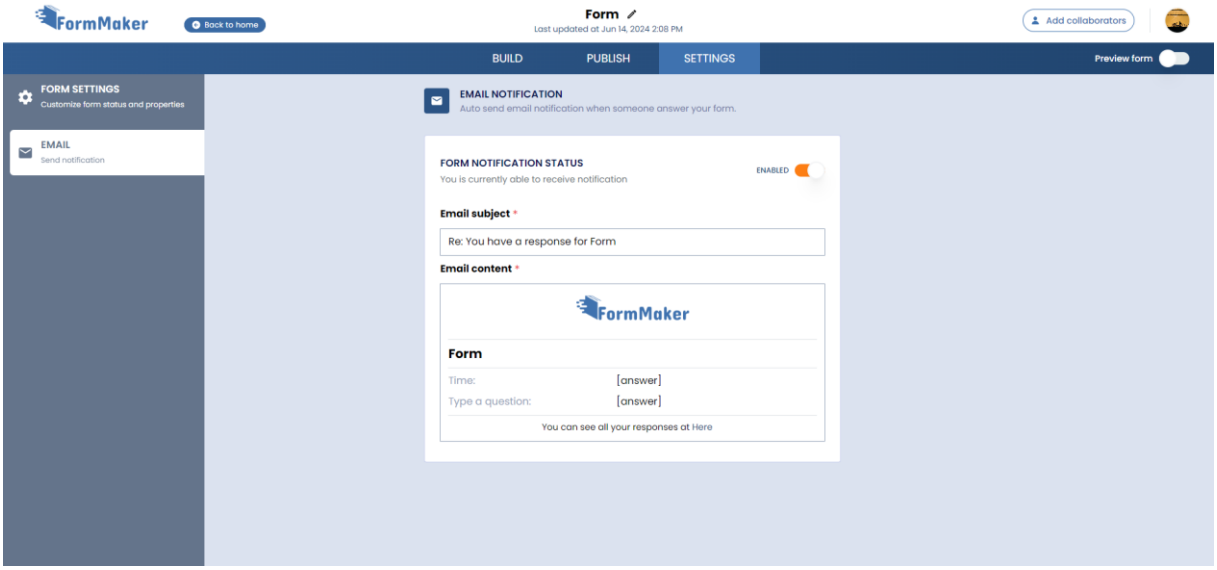

Figure 42. Form notification status

This feature is used to enable notifications whenever a form receives a new response. Notifications are sent to the owner's Gmail account.

Advantages: It allows the form owner to receive notifications of responses without needing to log into the system.

Disadvantages: The system still cannot send all responses directly to email; the owner needs to follow a link from the email to the system to view all responses.

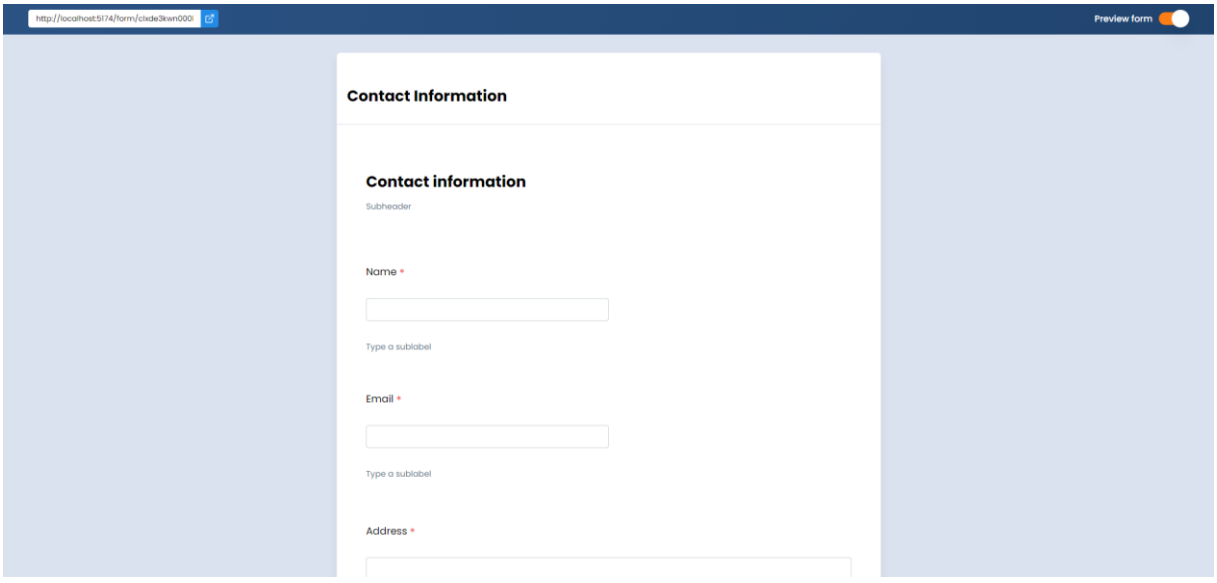

Figure 43. Preview form screen

Users click the "Preview form" button to preview the form. Users can interact with the form as if they were a respondent. They can click "Submit," but the system will not create a response.

The advantage of this feature is that it allows the owner to have an overview of their form before sending it out for others to submit. Users can test the form to avoid mistakes before sending it.

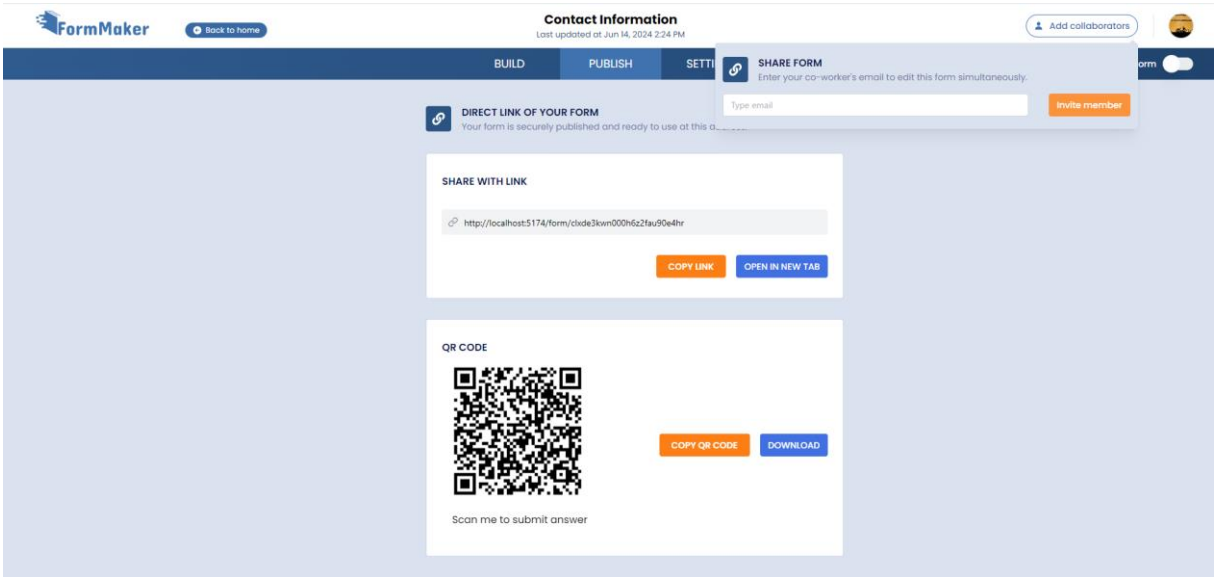

Figure 44. Add collaborator screen

Users select the "Add collaborator" button to invite others to manage a form. They enter the collaborator's Gmail address and click "Invite." The system sends an email to the collaborator. When the collaborator clicks "View form" in the email, the system updates the form's managing members.

Advantages: Users can assign forms to others and also remove collaborators from the form.

Disadvantages: If a form has too many members editing it simultaneously, it may be difficult to track the editing history of each member, leading to conflicts.

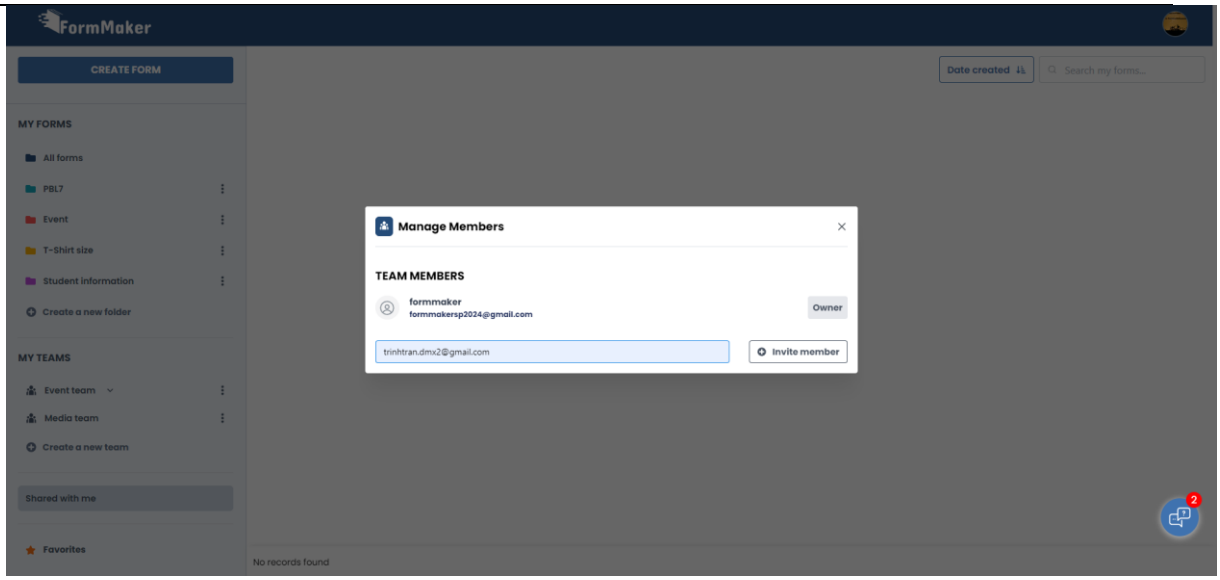

Figure 45. Create folder screen

The user clicks the "Create new folder" button, then enters the folder name and clicks "Submit" to create the folder.

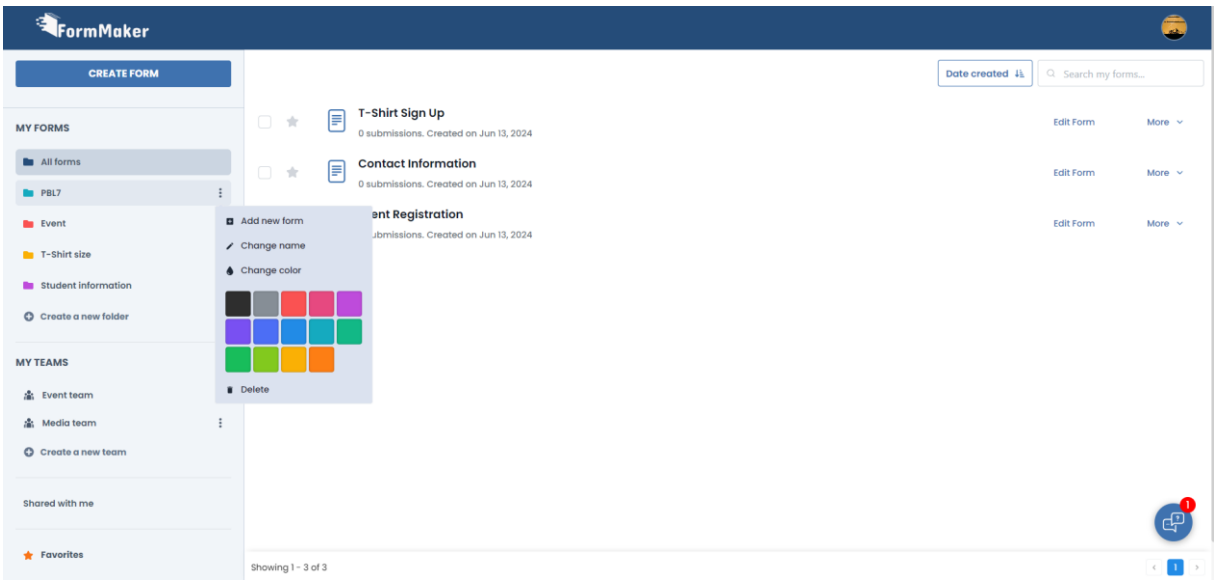

Figure 46. Change folder color screen

Users click on the ellipsis icon at the end of each folder and select the folder icon color.

Advantages: Enhances UI by allowing users to easily distinguish folders using colors.

Disadvantages: Currently, the system only allows changing to a few predefined colors provided by the system.

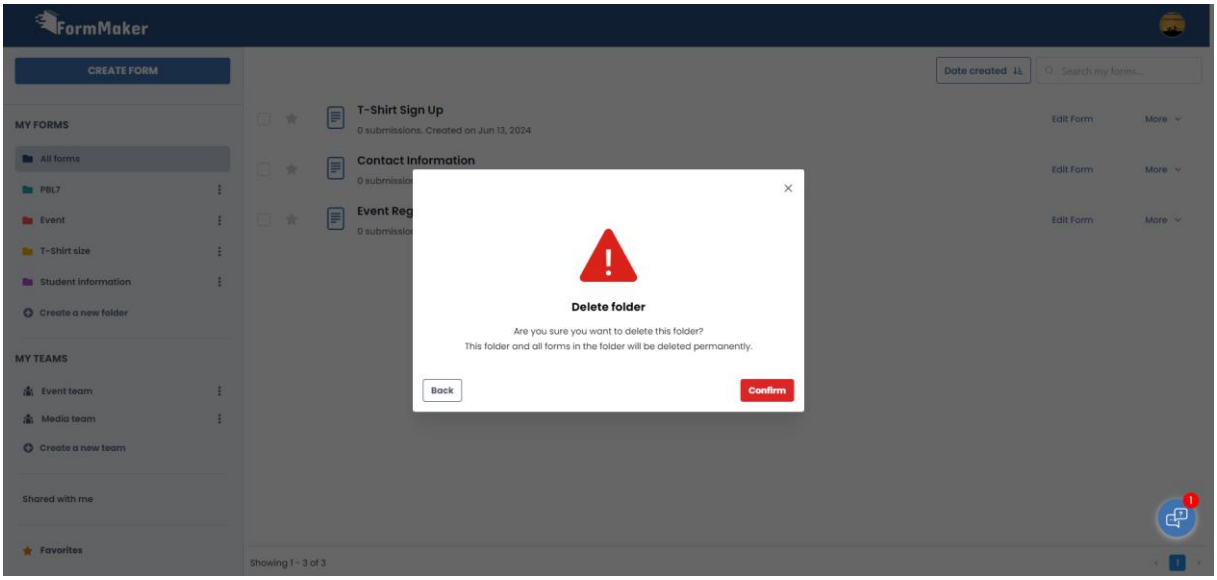

Figure 47. Delete folder screen

Users select "delete folder" from the menu of each folder and press "confirm" to delete the folder.

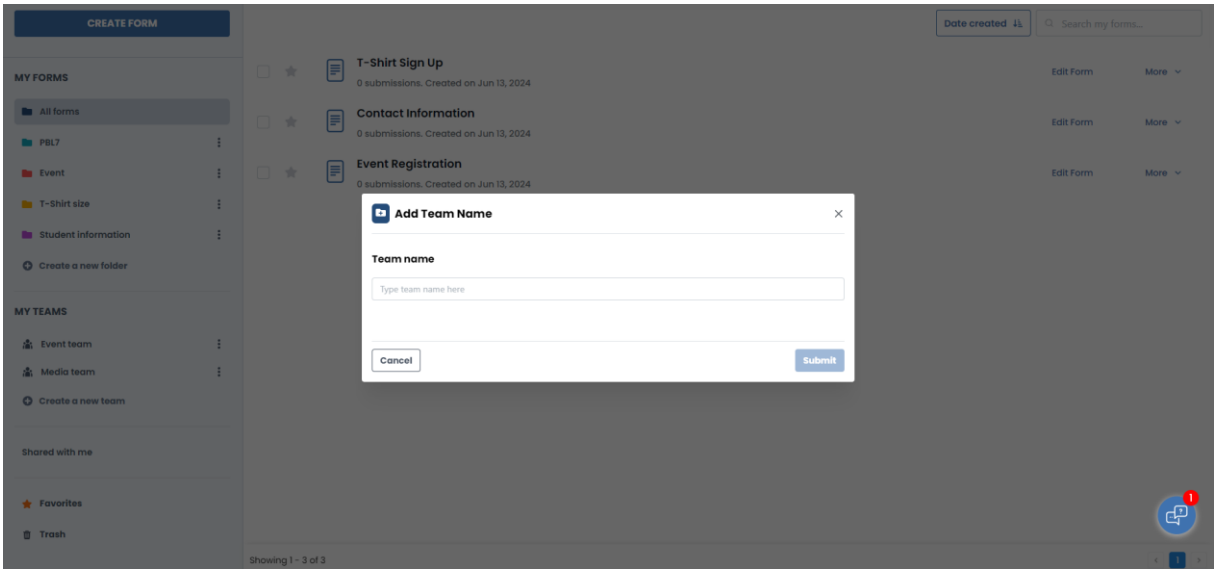

Figure 48. Create team screen

Users click the "Create new team" button, enter the team name in the input field, and submit to create a new team.

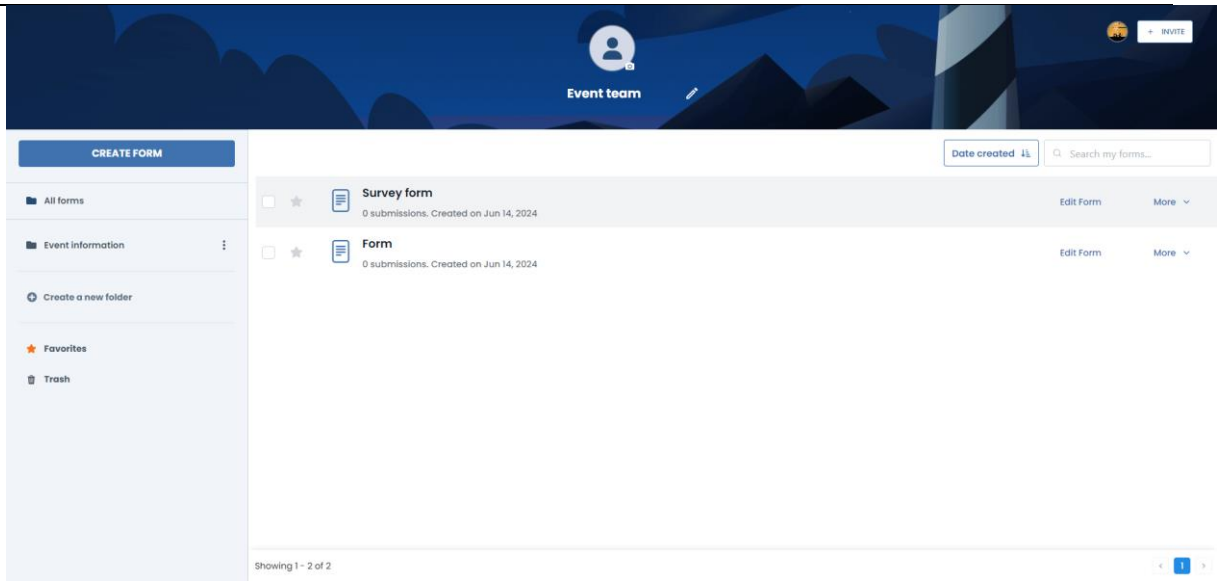

Figure 49. Team workspace

Users select "Team workspace" from the team menu. The system redirects to the team workspace page where users can manage the team with all features available as on the home page. Users can change the team name and avatar.

Advantages: It provides users with a spacious area to easily manage the team and perform operations efficiently.

Disadvantages: Currently, creating forms within the team does not allow importing forms from Google Forms or creating forms via chatbot.

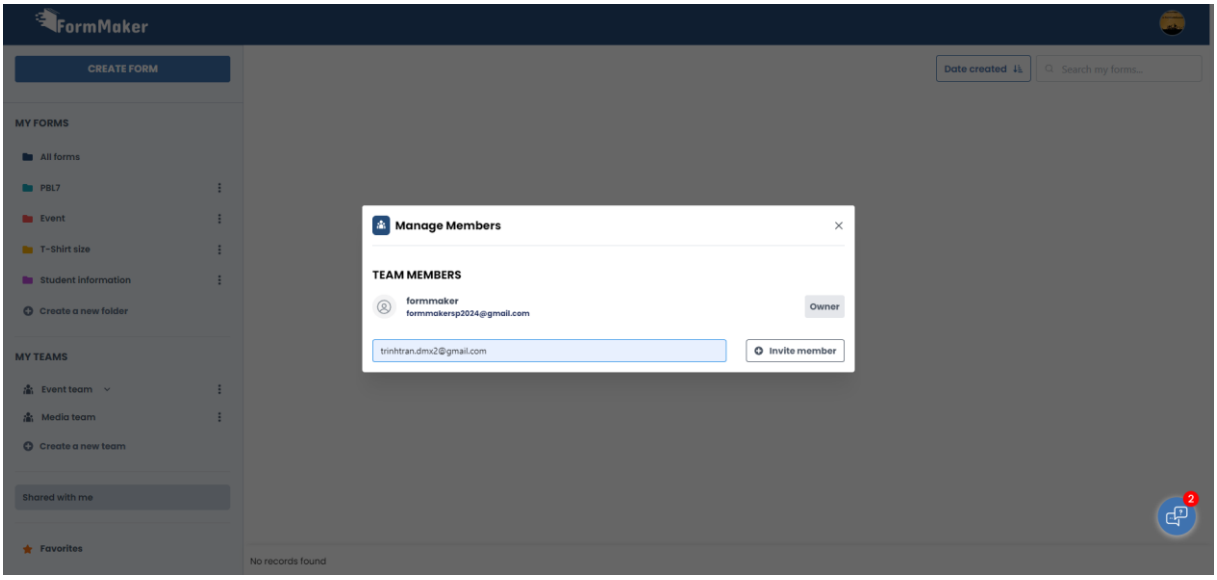

Figure 50. Manage member to team screen

Users select "Manage members" from the team menu. They enter the Gmail address of the member and click "Invite member." The system sends an email to the member. The member then clicks "View invitation" and "Accept invitation" to join. Only the team owner can remove members from the team.

Advantages: This feature helps the owner easily manage team members.

Disadvantages: The system currently lacks the ability to transfer ownership to another member within the team.

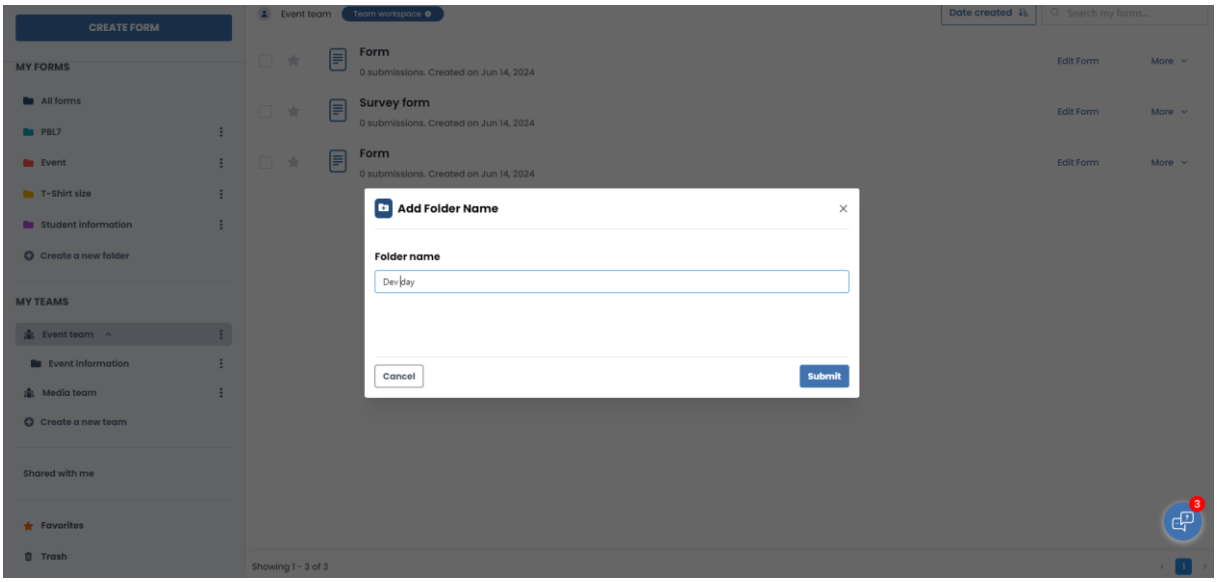

Figure 51. Add form to folder screen

Users select "Add new folder" from the team menu. They enter a name and submit to create a new folder within the team.

Advantages: This helps the team manage forms by organizing them into folders, making form storage easy and organized.

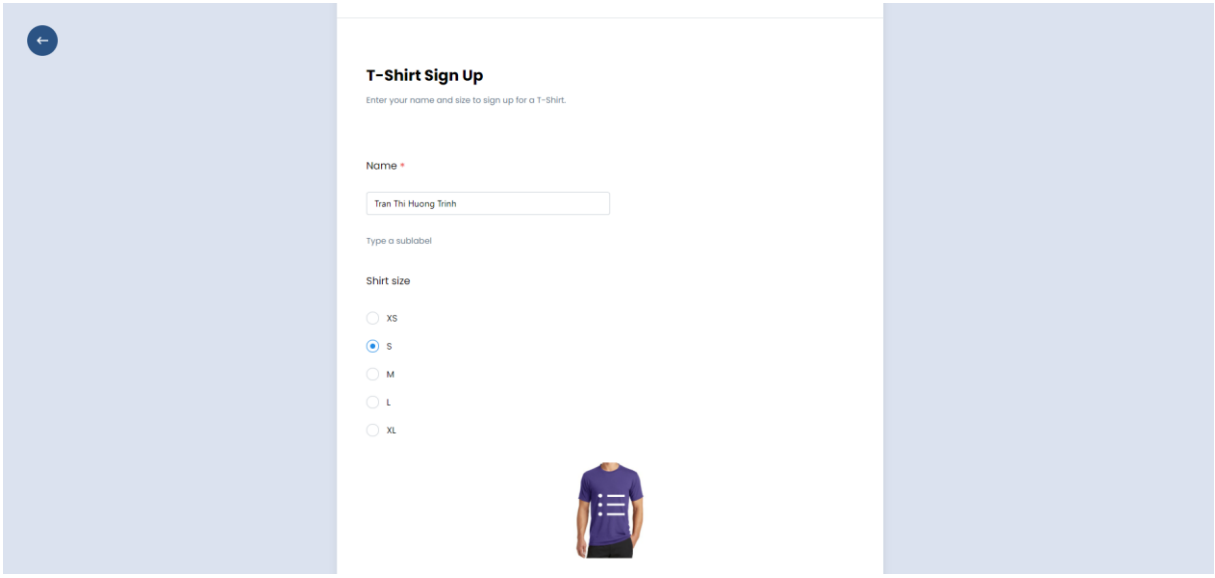

Figure 52. Create respone screen

Users select the "Publish" tab and choose "Open in new tab." They fill out all the responses of the form and click submit. The system validates the responses to check if they meet the form's requirements. If correct, the system redirects to the thank you page, confirming successful submission of the response.

Advantages: The system validates input data to ensure it meets the required format, criteria, or completeness.

Disadvantages: When users enter invalid data, the system may not accurately render the position of the error, making it difficult to identify where the information is incorrect. This can reduce user experience (UX).

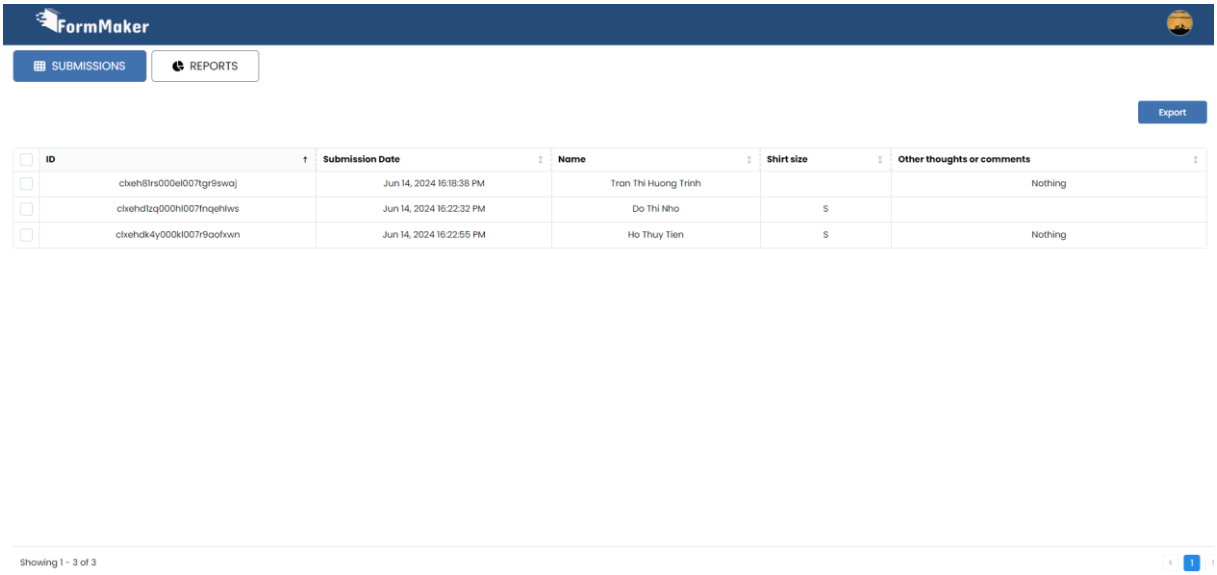

Figure 53. View responses screen

Users select a form and click "View submissions." They can view a submission table that includes ID, submission date, and answers from the form. The table can be resized as desired by the user. Users can sort responses or delete multiple responses.

For responses that include file uploads, the system allows users to click on the link to access the uploaded file.

Advantages: Users can interact with responses, paginate them for easier management, and access uploaded files directly from the submission table.

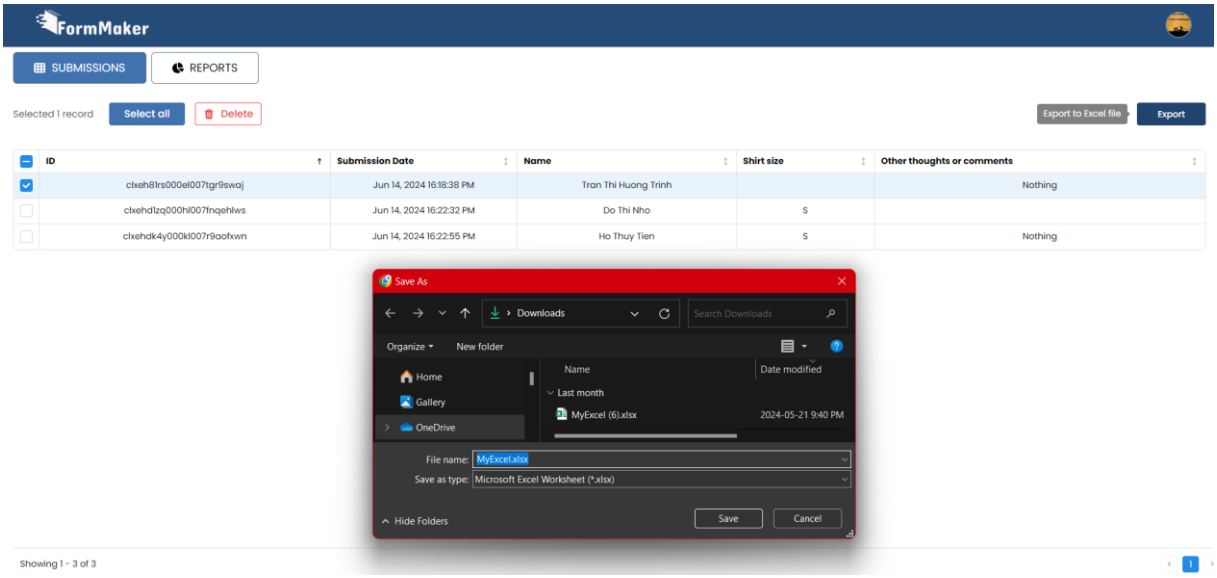

Figure 54. Export response to excel file

Users can export data to an Excel file for convenient data analysis and manipulation.

Disadvantage: The system only allows users to export all columns from the response table. Sometimes, users may only want to export specific fields of interest.

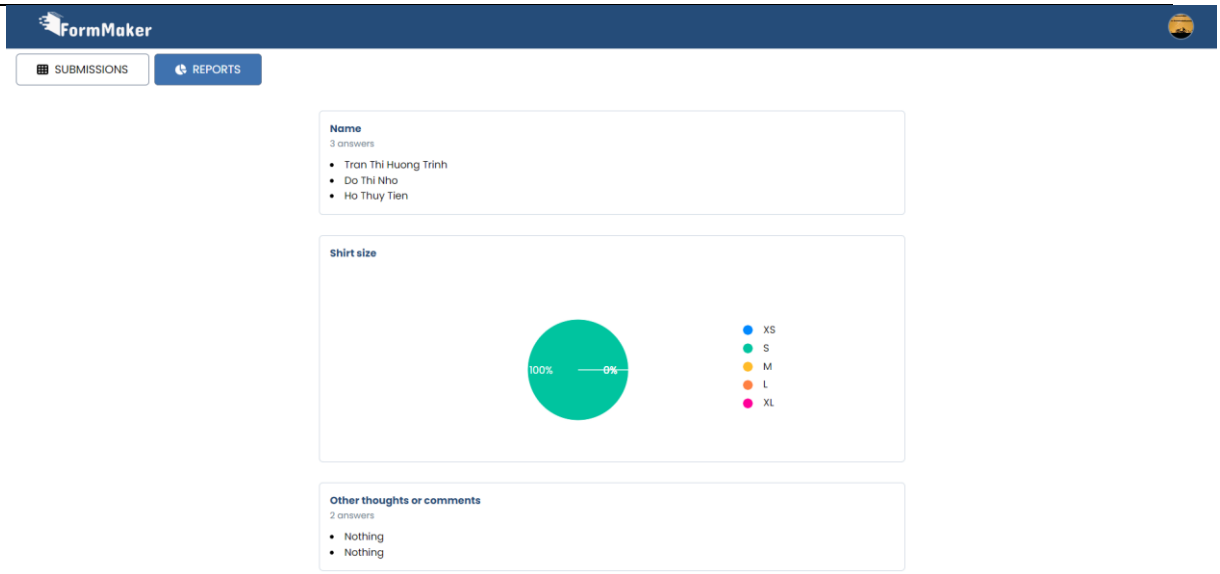

Figure 55. Report response screen

Users select the "Reports" tab to view a summary of the responses.

Advantages: Users can get an overview of the total number of responses, view pie charts for single-choice and dropdown responses, and view bar charts for multiple-choice and scale rating responses. This helps users analyze responses more effectively, especially for forms with a large number of responses.

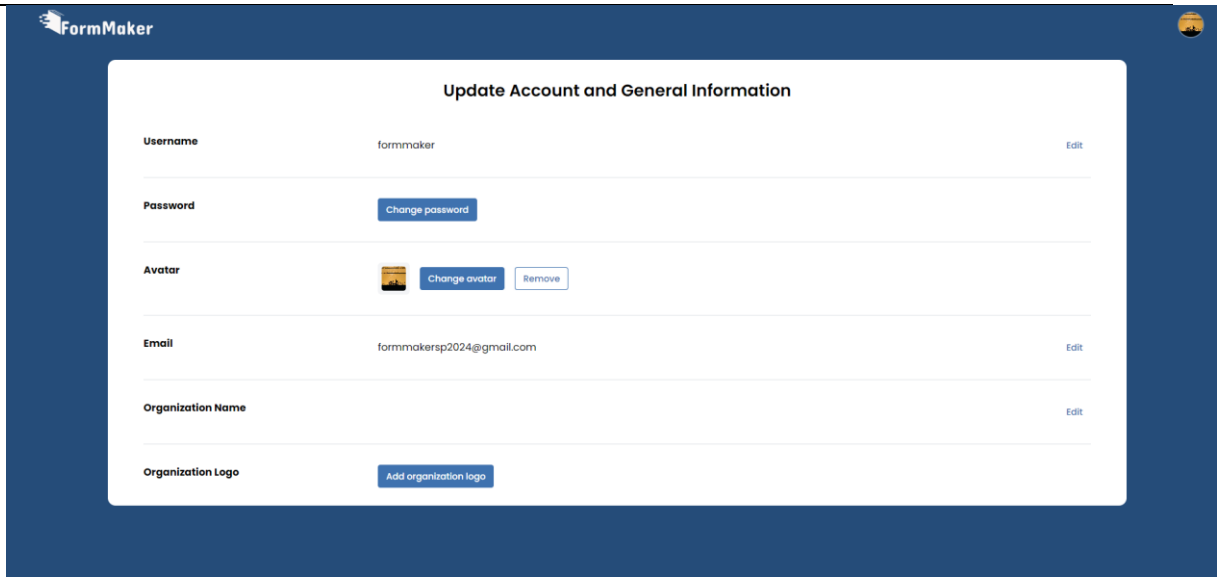

Figure 56. Manage account screen

Users can update their account details such as changing username, password, avatar, email, organization name, and organization logo.

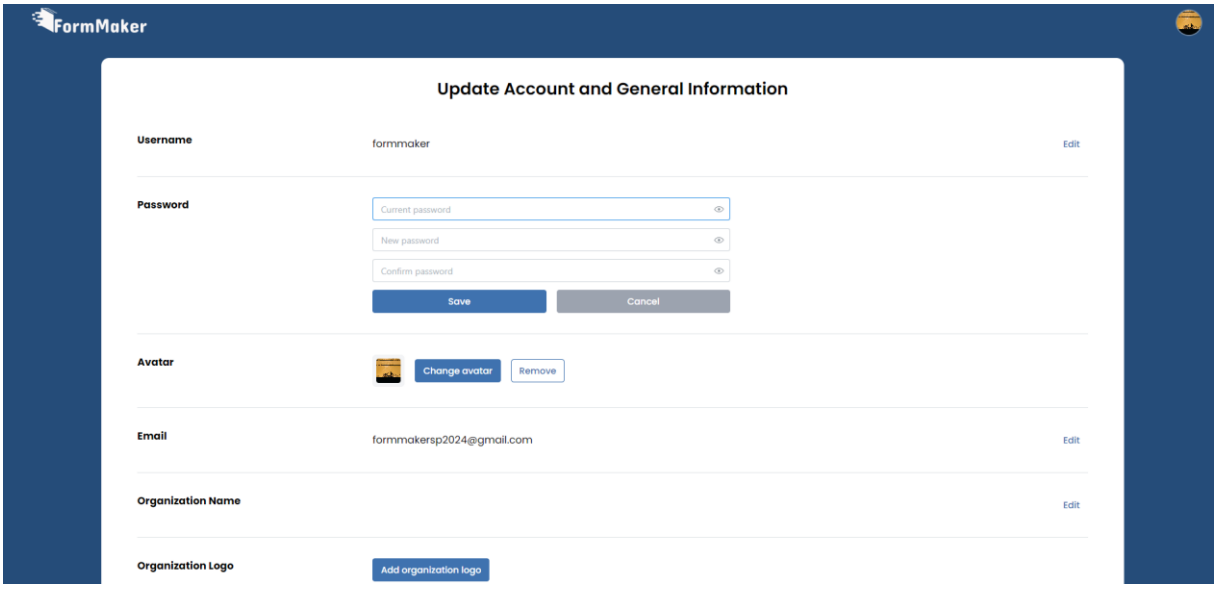

Figure 57. Change password screen

User clicks on the “Change password” button. Then, entering current password, new passwrod, and confirm the new password.

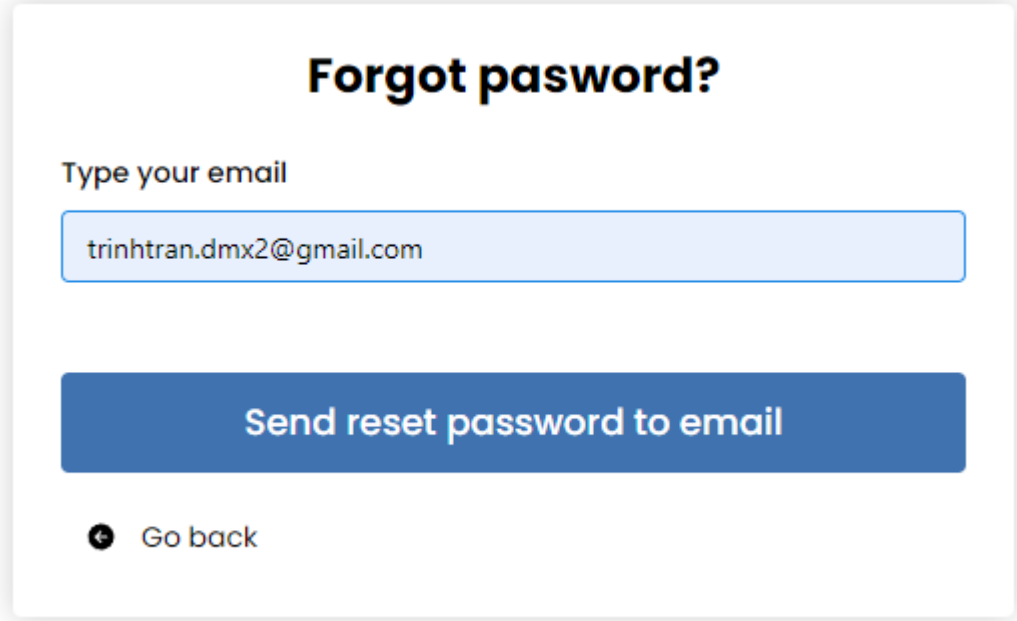

Figure 58. Forgot password screen

Users click on "Forgot password," and the system redirects them to the forgot password page. They enter their email account and click on "Send reset password to email."

The system then sends a link to that email for password reset.

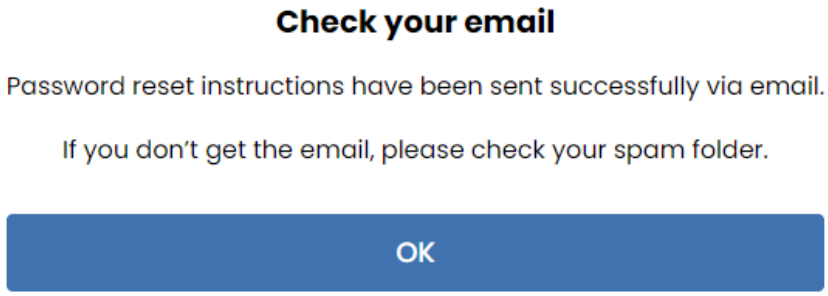

Figure 59. Notificate email has been sent screen

After receiving this notification, users access their Gmail account that they provided and check the email from formmakersp2024@gmail.com.

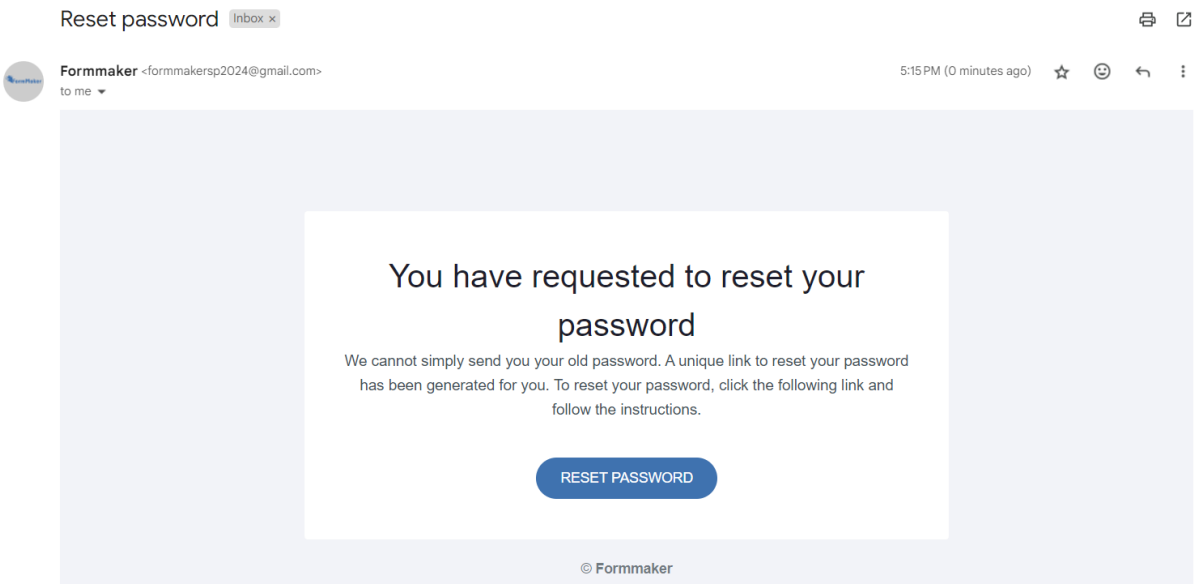

Figure 60. System sends email screen

Users click on the “Reset password” button. The system then redirects them to the reset password page.

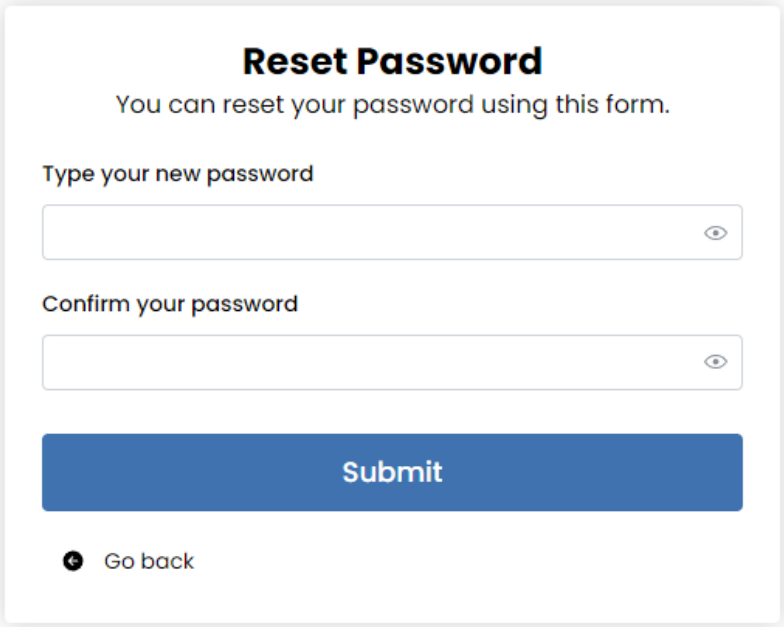

Figure 61. Reset password screen

Users enter a new password, confirm the password, and click the "Submit" button. The system then redirects them to the home page. This indicates that the user has successfully reset their password.

CONCLUSION

1. Achievement

During the time of researching, researching the theoretical basis and deploying technology application, the project has achieved the following results:

Theoretical side: After building this system, I could understand how to work with some frameworks and libraries such as NodeJS Framework, ReactJS with tailwind for CSS, typescript and apply libraries. I learned about the structure of the frameworks and how to apply them to my project. Besides, it also helps me to update technology trends and my future career.

Application: Developed an app enabling users to easily generate forms with chatbot assistance, enhancing online form creation.

Users could import an existing form from Google to the app.

They could handle forms by creating, reading, updating, deleting, searching, previewing, viewing all submissions, moving them to another folder or team, or inviting collaborators to manage jointly.

Throughout the project, I enhanced various skills including English proficiency, self-study, research, planning, and presentation abilities, ...

2. Future work

With some disadvantage I have already mentioned in evaluation demo main features and evaluation part, the future work of this project is as follow:

- Import forms sequentially, no need to enter links
- Chatbot can automatically edit forms as required, increase accuracy and can link requests together for smarter responses.
- Add more properties to flexibly edit each element
- Allows the installation system to receive notifications when the form has a large number of submissions
- Allows viewing the edit history of each member of the team, members can leave the team
- Allows many ways to choose color for folders
- Transfer team ownership rights to another member
- Allows you to select fields to export to excel file
- Render the correct location where the user entered the error to validate the answer

REFERENCES

[1] Many authors, “About Node.js?”, nodejs.org. [Online]. Available at: https://nodejs.org/en/about

[2] Many authors, “What is Javascript?”, aws.amazon.com. [Online]. Available at: https://aws.amazon.com/vi/what-is/javascript

[3] Nguyễn Hưng, "What is TypeScript? The complete basic knowledge about TypeScript for beginners," vietnix.vn. [Online]. Available at: https://vietnix.vn/typescript-la-gi

[4] Many authors, “HTML: HyperText Markup Language”, developer.mozilla.org. [Online]. Available at: https://developer.mozilla.org/en-US/docs/Web/HTML.

[5] Many authors, “CSS: Cascading Style Sheets”, developer.mozilla.org. [Online]. Available at: https://developer.mozilla.org/en-US/docs/Web/CSS

[6] Many authors “Express.js”, wikipedia.org. [Online]. Available at: https://en.wikipedia.org/wiki/Express.js

[7] Many authors, “Getting started with React”, developer.mozilla.org. [Online]. Available at: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started

[8] Tailwind CSS authors, “Documentation overview. Tailwind CSS”. [Online]. Available at: https://tailwindui.com/documentation

[9] Mantine authors, “About Mantine – How to implement in React”. [Online]. Available at: https://mantine.dev/about/

[10] React Grid Layout contributors, “React Grid Layout - Usage”. Available at: https://github.com/react-grid-layout/react-grid-layout

[11] Nguyễn Hưng, “What is PostgreSQL? Overview of knowledge about the PostgreSQL data management system?”, vietnix.vn. [Online]. Available at: https://vietnix.vn/postgresql-la-gi

[12] Prisma authors, “Build with Prisma ORM”. [Online]. Available at: https://www.prisma.io/docs

[13] Many authors, “What is API RESTful?”, aws.amazon.com. [Online]. Available at: https://aws.amazon.com/vi/what-is/restful-api

[14] openAI authors, “OpenAI documentation”, platform.openai.com, [Online]. Available at: https://platform.openai.com/docs/overview

How to implement into the project: https://www.npmjs.com/package/openai