Merge, a team building and cooperation platform for developers and designers

Team F: Myungha Cho, Jaebeom Cho, Joonsun Baeck, Haim Nam

June 9, 2023

Abstract

Merge is a team building and collaboration platform for developers and designers. The main function is to automatically create an optimal team by receiving user information in advance, such as users' experience, investment time, work time, and preferred roles. In addition, there is a team chat function and a function to check projects you have participated in before.

Keywords: team project, collaboration, optimized team building

1 Introduction

Merge is a team building and collaboration platform designed specifically for developers and designers. It aims to simplify the process of team formation and enhance collaboration among users. By leveraging user-provided information, such as experience, investment time, work time, and preferred roles, Merge automatically creates optimal teams for various projects.

Team building is a crucial aspect of any project, and finding the right team members with the required skills and compatibility is often challenging. Merge alleviates this burden by intelligently matching users based on their expertise and availability. The platform facilitates efficient collaboration by providing a team chat function and enabling users to easily access and manage their past project engagements.

With Merge, users can focus on their core tasks, knowing that the team formation process is optimized and tailored to their specific requirements. By bringing together individuals with complementary skills and shared goals, Merge fosters effective collaboration and enhances project outcomes.

This document provides a comprehensive overview of the design, implementation, and evaluation of Merge, highlighting its key features, architecture, and algorithmic approach. It aims to showcase the capabilities of Merge in enabling seamless team building and collaboration among developers and designers.

2 Design

2.1 DB Architecture

MongoDB
MongoDB is a flexible and scalable NoSQL database system that stores data in a JSON-like format called BSON. It is designed to handle large amounts of data and offers high performance and scalability. MongoDB's document-based model allows for flexible and dynamic schemas, making it suitable for projects like Merge where data structures may evolve over time. Additionally, MongoDB's query language and indexing capabilities enable efficient data retrieval and manipulation, enhancing the overall performance of the application.

2.2 Server Architecture

Express.js
Express.js is a popular web application framework for Node.js. It provides a minimalistic and flexible approach to building web applications and APIs. With its simplicity and robust features, Express.js allows developers to create server-side applications efficiently. It provides a routing system, middleware support, and integrates well with other Node.js modules, making it a suitable choice for building the backend of Merge.

The combination of Express.js and MongoDB provides a solid foundation for building the backend of Merge, offering flexibility, scalability, and performance.

2.3 Frontend Architecture

2.3.1 Tech Stack

The frontend architecture of our web application is built using React, a popular JavaScript library for building user interfaces. React is a good choice for several reasons:

1. Component-Based Approach: React follows a component-based architecture, allowing us to break down the UI into reusable and self-contained components. This promotes modularity, code reusability, and easier maintenance.

2. Virtual DOM: React utilizes a virtual DOM, which improves performance by efficiently updating and rendering only the necessary parts of the user interface. This results in a faster and smoother user experience.

3. Declarative Syntax: React uses a declarative syntax, making it easier to understand and reason about the UI. It allows us to describe how the UI should look at any given state, and React takes care of updating the UI accordingly.

2.3.2 Technologies Used

Typescript: TypeScript is a superset of JavaScript that adds static typing to the language. It extends the capabilities of JavaScript by providing type annotations, interfaces, classes, and other features that enhance code quality, maintainability, and developer productivity. With TypeScript, developers can build robust, scalable, and maintainable React.js applications with confidence.

Javascript: We also use JavaScript alongside TypeScript, as it forms the foundation for both languages. JavaScript allows us to leverage the rich ecosystem of libraries and frameworks available.

Functional Components: Instead of class components, we use functional components in React. Functional components are simpler, more concise, and easier to understand. They also facilitate the use of React Hooks, such as useState, useEffect, and useContext, which enable us to manage state and side effects within the functional components themselves.

CSS Module, SASS: We utilize the module.scss approach for styling our components. This allows us to write scoped and modular CSS styles, preventing style conflicts and making it easier to maintain and reuse styles across different components.

2.3.3 Development Tools

Code Editor: We use either WebStorm or VSCode as our code editor, providing a powerful and feature-rich environment for writing code.

Package Manager: We rely on a package manager (such as npm or Yarn) to handle dependencies, build scripts, and other related tasks.

Version Control: We collaborate using Git as our version control system, enabling efficient code collaboration, branching, and merging.

2.4 Algorithm

Our main algorithm is the team formation algorithm, which is primarily based on the answers provided by users regarding their desired class participation. The algorithm aims to form the optimal teams based on user answers, but since the users' responses may vary, there might not always be a perfect optimal solution. However, the algorithm strives to provide sub-optimal results that are as close to the correct answer as possible. The user's answers are based on five questions: coding experience, weekly time commitment, preferred dates and times, preferred role, and supported positions.

Our algorithm can be divided into the following steps:

Step 1: The algorithm first focuses on assigning individuals to teams. Based on the "positionComposition" variable specified by the project host and the positions indicated by the participating candidates, it determines how many teams will be formed and how many members with specific positions will be assigned to each team.

Step 2: Once the team composition is determined, the next step is to assign leaders to each team based on their preferred roles. Those who prefer to be leaders are assigned to teams in the order they applied. If the number of candidates who prefer to be leaders is less than the required number of leaders for the teams, individuals who don't prefer leadership but have more coding experience are randomly assigned as leaders.

Cases for team matching iteration

- **Case 0: Match all** This case aims to match all conditions specified by users' answers.
- **Case 1: Match all with lower condition** Similar to Case 0, but with relaxed conditions.
- **Case 2: Match preferred time and experience** This case focuses on matching users based on their preferred time availability and experience.
- **Case 3: Match preferred time and experience with lower condition** Similar to Case 2, but with relaxed conditions.
- **Case 4: Match preferred time and time spend** This case considers users' preferred time availability and the amount of time they are willing to invest.
- **Case 5: Match preferred time and time spend with lower condition** Similar to Case 4, but with relaxed conditions.
- **Case 6: Match experience and time spend** This case aims to match users based on their experience and the amount of time they are willing to invest.
- **Case 7: Match experience and time spend with lower condition** Similar to Case 6, but with relaxed conditions.
- **Case 8: Match time spend** This case focuses on matching users based on the amount of time they are willing to invest.
- **Case 9: Match time spend with lower condition** Similar to Case 8, but with relaxed conditions.
- **Case 10: Match preferred time** This case considers users' preferred time availability.
- **Case 11: Match experience** This case focuses on matching users based on their experience.
- **Case 12: Match experience with lower condition** Similar to Case 11, but with relaxed conditions.

Step 3: The remaining individuals are then assigned to teams one by one. Ideally, all individuals should be placed in groups where their answers to all questions align. However, due to variations in user answers, it's unlikely to form such groups. Therefore, we relax or exclude conditions for one of the questions from the list and form groups one by one. If a group reaches the maximum team capacity, it is considered formed, and this process goes through 13 iterations based on different question conditions as shown above.

If after the 13 iterations there are still individuals who haven't been assigned to a team, they are randomly placed together to form a team with the remaining members.

2.5 Challenges and Handling

2.5.1 Backend

Testing Environment: We faced difficulties when conducting tests on the backend. To test our algorithm, we needed to create random classes and generate users with random answers to participate in the classes, followed by running the algorithm to form teams. After the tests, we had to restore the data to its original state. This sequential process meant that if the preceding functionalities were not implemented correctly, it became challenging to test the subsequent functionalities. Consequently, a significant amount of time was invested in

setting up the testing environment.

To address this issue, we utilized the Mocha testing library to perform the tests. Additionally, we created functions responsible for generating random users and answers, as well as functions for initializing and managing the database, allowing us to reset the data as needed.

Team forming Algorithm implementation: During the formation of the team building algorithm, we encountered a challenge. In summary, we ended up modifying the algorithm.

Initially, we implemented a score-based algorithm where each question had its own weight. Based on the users' answers, we calculated connection scores between all users and formed a graph using these scores. The team formation was then performed using a greedy algorithm on the edges of the graph. However, the problem with this algorithm was that it was difficult to determine if the teams were being formed in the desired manner. It was challenging to verify if the teams were aligned with the users' preferences.

As a result, we decided to abandon the score-based algorithm and instead focus on creating sub-optimal teams. This modification allowed us to verify the formation of teams based on the specific answers provided by the users.

2.5.2 Frontend

Project Management: In the Merge Project Management application, we prioritized creating an intuitive and user-friendly interface. Based on the purpose of our service, we designed a streamlined workflow with clear navigation and organized information. Additionally, we implemented features like progress bars, tooltips, and user-friendly error handling to enhance the overall user experience.

Registration: In the registration process, we aimed to create a smooth and frictionless experience for new users. We implemented a step-by-step approach, breaking down the registration process into manageable sections with clear instructions. We utilized user-friendly form validation techniques to provide real-time feedback and guide users in filling out the required information correctly. By reducing cognitive load and providing clear calls-to-action, we ensured that users could easily complete the registration process.

3 Implementation

3.1 Backend Implementation

3.1.1 DB Schema

The following schemas are used as our main schema for implementing our product:

User Schema
The User schema defines the structure for storing user information. It includes the following fields:

- **email**: The user's email address (required and unique).
- **password**: The user's password (required).
- **verifyCode**: The verification code associated with the user (required).

- **classes**: An array of class references in which the user is participating.
- **positionIndexByClass**: An array containing the position index for each class the user is part of.
- **name**: The user's name (optional).

Answer Schema
The Answer schema represents the answers provided by users for a particular class. It consists of the following fields:

- **class**: The reference to the Class to which the answer belongs.
- **guest**: The reference to the User who provided the answer (required).
- **answer**: An array containing the user's answers (required).

Class Schema
The Class schema defines the structure for storing class-related information. It includes the following fields:

- **host**: The reference to the User who is hosting the class.
- **guest**: An array of guest user references who have joined the class, along with their associated answers.
- **teams**: An array of references to the teams formed within the class.
- **questionIds**: An array of question identifiers for the class.
- **className**: The name of the class (required).
- **classDescription**: The description of the class (required).
- **classType**: The type of the class (e.g., web, app, game) (required).
- **positionTypes**: An array of position types required for the class (required).
- **positionComposition**: An array specifying the number of positions needed for each position type (required).
- **positionCounts**: An array containing the count of positions available for each position type.
- **recruitStartDate**: The start date of the recruitment period for the class (required).
- **recruitEndDate**: The end date of the recruitment period for the class (required).
- **activityStartDate**: The start date of the class activity (required).
- **activityEndDate**: The end date of the class activity (required).
- **isSecret**: Indicates whether the class is secret or not (default is false).
- **isHostParticipating**: Indicates whether the class host is participating in the class or not (required).
- **accessKey**: The access key for joining the class (optional).
- **hasAccess**: An array of user references who have access to the class.

Question Schema
The Question schema represents the structure for storing questions associated with a class. It includes the following fields:

- **id**: The identifier for the question (required).
- **title**: The title or prompt for the question (required).
- **options**: An array of options for the question (default is an empty array).
- **weight**: The weight or importance of the question (required, default is 5).
- **countScore**: Specifies whether all answers to this question should have the same score or different scores (required, values are "same" or "different", default is "same").

3.1.2 API Design

The backend implementation of UniCoop follows the following folder structure:

auth: checkEmail, deleteUser, getUser login, verify

class: checkAccessKey, createClass, formTeam, getAllClasses, getClass, joinClass

question: getQuestions

team: getTeam, postMessage

The API routing is divided into three main sections: auth, class, and team.

The **auth** section consists of APIs related to authorization, such as login, register, and user verification.

The **class** section includes APIs for class management, such as creating a class (*createClass*), joining a class (*joinClass*), and forming a team (*formTeam*). It also provides APIs for retrieving information about all classes (*getAllClasses*) and a specific class (*getClass*).

The **question** section provides an API for retrieving questions (*getQuestions*).

The **team** section includes APIs for managing teams, such as retrieving team information (*getTeam*) and sending messages between teams (*postMessage*).

This folder structure and API design allow for a well-organized and modular backend implementation, making it easier to manage and maintain the UniCoop application.

3.1.3 Testing

Test

- login.test
- createClass.test
- joinClass.test
- formTeam.test

Testing Approach

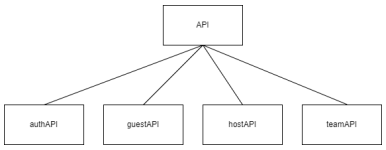

Figure 1: Frontend API Structure

The APIs are tested using the test codes located in the test folder. The focus of the tests is primarily on the POST APIs. Test codes have been written for login, create class, joinClass, and formTeam. Since create class requires a user to be logged in, joinClass requires create class to be completed, and formTeam requires joinClass to be completed, the tests were conducted sequentially to ensure the necessary prerequisites are met.

3.2 Frontend Implementation

3.2.1 API

In the front end, the APIs for communicating with the back end were divided into four categories according to the topic. API related to user authentication is divided into authAPI, API related to process projects as a guest is managed in guestAPI, API related to handle projects as a host is managed in hostAPI, and API related to team activities is divided into teamAPI.

3.2.2 Key Components

- **CodePopUp:** Utilizes a Modal to display code or content in a pop-up window.
- **LabelInput:** A reusable input component that allows users to enter data with a label attached.
- **Layout:** Acts as a wrapper, providing a consistent layout structure for all pages. It accepts different pages as children and ensures a unified visual design and navigation experience.
- **Loader:** Responsible for displaying a loading indicator to enhance the user experience during data fetching or other asynchronous operations.
- **Logo:** Helps reduce code repetition by providing a centralized way to display the application logo.
- **MergeButton:** Designed to make styles extensible, allowing for easy customization and reuse across different button variations.
- **NavBar:** Renders the navigation bar, providing links and options for users to navigate through the application.
- **OptionRadios:** Represents a group of radio buttons and receives an onChange function in a specified format to handle user selections.
- **ProgressBar:** Visualizes the progress of a user's process or task, providing a visual indicator of completion.

Figure 2: Frontend Pages Structure

- **ProjectBox:** Represents a box containing information about collaborative projects that are used repeatedly throughout the application.

3.2.3 Additional Files

Apart from the key components and pages, our frontend architecture includes the following additional files:

- **MergeContext.tsx:** This file manages frequently accessed information such as myInfo and token headers as a context using React's useContext hook.
- **helper.ts:** Trivial but complex and frequently used functions are managed in this file, improving code organization and reusability.
- **interface.ts:** This file contains TypeScript interfaces, defining type information and default values used throughout the application. TypeScript's type sensitivity helps catch errors and provides better development tooling support.

3.2.4 Other Technologies and Libraries

In addition to React and TypeScript, we leverage the following technologies and libraries:

- **useSWR:** We partially use swr to handle data fetching and caching, improving the user experience by reducing unnecessary network requests.
- **dayjs:** dayjs is used to manage and manipulate date-related data in the application.
- **material-ui-icons:** This library provides a collection of icons that enhance the visual representation of various UI elements.
- **toast:** The toast library is utilized to display user-friendly notifications, particularly for error handling and providing clear feedback to users.

3.3 UX/UI Viewpoints

3.3.1 Pages

There are eight major pages that users can access: Activity, Apply, Home, Manage Project, My Page, Project Participation, Project Registration, Sign In, What Is Merge Page

- **Home:** The landing page of the application, providing an introduction and overview of the service.
- **MyPage:** Allows users to view and edit their personal information.
- **SignIn:** Provides a login and sign-up interface for users.

- **WhatIsMerge:** This page offers an introduction to the service and allows users to provide feedback.
- **RegisterProject:** A two-step process where users can input project information and answer questions to register a project.
- **ManageProject:** This page enables users to manage their projects, including editing project details and performing related actions.
- **ParticipateProject:** Users can view and explore all available projects to participate in.
- **Apply:** Users can apply for a project by answering the provided questions.

4 Evaluation

4.1 Algorithm Evaluation

To evaluate the effectiveness of our algorithm, we conducted multiple tests to assess its performance. The results of these tests are summarized in the following table:

User Counts	10	20	30	40	50
Unformed Teams	1.3	1	1	1	1
Matching more than two	0.72	0.78	0.75	0.81	0.78

Table 1: Algorithm analysis

As shown in the table, the majority of teams were successfully formed, regardless of the number of users. The number of unformed teams remained consistently low across different user counts. Additionally, the percentage of teams that satisfied more than two conditions was consistently high, with all values exceeding 70percent. These results indicate that our algorithm has achieved a significant level of success in providing suboptimal solutions. Furthermore, it was able to effectively form teams even in cases where participants supported only one position or provided the same answers.

Overall, the evaluation of our algorithm demonstrates its capability to form teams and satisfy multiple conditions, thereby contributing to successful team building.

4.2 Client server interaction

Overall, the communication between the client and the server was smooth and error-free. We did not encounter any major issues during the interaction between the two. All APIs functioned properly and performed as expected. The client was able to send requests to the server, and the server responded correctly with the requested data or performed the required operations. This seamless client-server interaction ensured a seamless user experience and efficient data exchange between the front-end and the back-end.

5 Limitations

5.1 Imperfect solution

In the end, the sub-optimal solution had its limitations, and it could not guarantee a 100 percent (at max 80 percent) perfect solution or prevent extreme edge cases. Our algorithm goes through 13 cases, which allows for team formation in most scenarios. However, if the teams cannot be formed even after going through all the cases, the remaining individuals are randomly assigned to teams. We concluded that further relaxing the conditions would not significantly improve the situation.

To address this limitation, we propose improving the user experience by providing users with better choices for team formation. This could involve suggesting more favorable options to users or expanding the algorithm's possibilities to handle more extreme cases. By implementing such improvements, we can enhance the team formation process and provide users with a better understanding of how teams are being formed.

5.2 Not Host centered team formation

Furthermore, our product currently forms teams based on the predefined questions and conditions provided by the project host. However, it is essential for the functionality of our product to allow hosts to customize the team formation process according to their preferences. To achieve this, we plan to implement a feature that allows hosts to define their desired team formation criteria. Implementing this feature would require rethinking and modifying the algorithm.

By incorporating this customization feature, hosts will have the flexibility to tailor the team formation process to their specific requirements. To accommodate this functionality, the algorithm would need to be redesigned and updated to align with the host's desired team formation approach.

6 Related Works

1. Asana and Trello: Project management tools for task collaboration and progress tracking. These tools do not have team formation functions.
2. Google Docs and Microsoft Teams: Real-time document collaboration and team communication tools. However, they do not include team formation functions.
3. iCampus: A platform suitable for university classes with team building features. It allows setting a duration for projects. However, it does not consider user preference, lacks feedback features, and does not have project management or progress check capabilities.
4. Beside: Provides good team matching, but lacks project management and progress tracking features. It is not suitable for university classes, does not consider user preference (only experience and skills), and requires manual team building.
5. BeginMate: Offers a team building system based on user preference, but is not suitable for university classes and requires manual intervention. It also lacks feedback, project management, progress check, and duration features.

6. Google Docs and Microsoft Teams: Real-time document collaboration and team communication tools. These tools do not include team formation functions.

Despite the existence of these collaboration services, there is still a gap in the market for a platform that can provide all necessary features for university team projects. Merge aims to fill this gap by offering a comprehensive platform that includes team matching, project management, progress tracking, and peer evaluation features, tailored specifically for academic team projects.

7 Conclusion

In conclusion, Merge is a team building and collaboration platform designed to address the challenges of team formation and cooperation. Our algorithmic approach aims to form optimal teams based on user-provided information, such as experience, time availability, preferred roles, and more. Through rigorous testing, we have evaluated the effectiveness of our algorithm and observed successful team formations in various scenarios.

However, we acknowledge the limitations of our sub-optimal solution. Although our algorithm strives to provide the best possible teams, it cannot guarantee a perfect solution in all cases. We have identified the need for continuous improvement, such as enhancing the user experience by suggesting favorable team options and expanding the algorithm's capabilities to handle more challenging scenarios. Furthermore, we recognize the importance of customization in team formation. To address this, we plan to implement a feature that allows project hosts to define their desired team formation criteria. This customization will provide flexibility and alignment with the host's preferences, requiring modifications to the algorithm to accommodate the changes. Overall, Merge aims to provide a comprehensive platform for team building and collaboration, supporting users throughout their entire project lifecycle. By addressing the limitations and incorporating user feedback, we strive to enhance the team formation process and deliver a more tailored and effective collaboration experience for developers and designers.

8 References

[1] Interview. https://www.notion.so/PICK-LE-5e3cc6a527304bcb85092468506fc7af. Year 2023. Accessed on March 14, 2023.

[2] Article. Author WomanEconomy Title Don't know even know names in team Project Url https://www.womaneconomy.co.kr/news/articleView.html?idxno=210026. Year 2023. Accessed on March 14, 2023.

[3] Cloudwards. (2023, March 29). Trello vs Asana: The Best Project Management App for Your Team. Retrieved March 31, 2023, from https://www.cloudwards.net/trello-vs-asana/

[4] Business 2 Community. (2023, March 29). 15 Best Project Management Apps in 2023. Retrieved March 31, 2023, from https://www.business2community.com/workflow/best-project-management-apps.

[5] Workable. "Collaboration Tools: A Tutorial for Busy Teams." Workable Resources. Workable, 27 Jan. 2023, https://resources.workable.com/tutorial/collaboration-tools.