SKKU-DM, a comprehensive information platform for dual major options for Sungkyunkwan University students.

Woojin Kwon[1], Jaehyeon Kim[2], Jisoo Park[3], and Chaeeun Lee[4]

[1] Sungkyunkwan University, wojin57@g.skku.edu
[2] Sungkyunkwan University, kjhkk1020@g.skku.edu
[3] Sungkyunkwan University, idpjs2000@g.skku.edu
[4] Sungkyunkwan University, ecloud0313@g.skku.edu

Abstract. This project aims to develop a mobile application-based online platform for students at Sungkyunkwan University who are pursuing or considering a double major. During this process, cloud computing services will be utilized to provide fast service to users, and a serverless deployment model will be implemented. This will allow us to gain experience and knowledge in operating next-generation application environments.

1 Introduction

Our team, composed entirely of double majors in Software Engineering, has experienced firsthand the difficulties in obtaining information related to double majoring in software. With this in mind, we aim to develop a software platform that provides relevant information and resources not only for software double majors but also for students pursuing double majors in other fields. Given the abundance of community-based platforms in the market, we recognized the need to introduce unique features that have not been available before.

Firstly, the required credits vary by department once students begin a double major. Although related files are uploaded to the university's website each year, there is no service that organizes this information in a user-friendly way. To address this, we've included a feature that visualizes the credit requirements for easy viewing. Additionally, we provide a service that recommends other double majors in similar situations, enabling users to connect and communicate. Another feature allows for mock applications, where users can compare their standing against other applicants pursuing the same double major, giving them a better sense of their competitive position.

In Chapter 2, we explain the motivation and need for this project. Chapter 3 introduces existing applications in this domain. Chapter 4 details the key features of our application, while Chapters 5 and 6 outline the project plan and team role distribution.

2 Motivation and Objective

2.1 Motivation

When SKKU students double major, they should investigate various informations. The requirements for graduate might be changed such as writing a paper or doing a project. They should search for joining a new club. They might want to know the roadmap for their new major. Some of them would be interested in the future plan for their career. Unfortunately, those informations are spread out, so it is hard to collect them all for their own. We want to resolve this problem.

2.2 Objective

We are willing to make a mobile application to resolve those problems. We come up with an idea from the LC policy of SKKU. It groups freshmen randomly so that they can easily get used to the campus life. Our goal is to make a community for double majoring students.

3 Related Works

3.1 Everytime [1]

Everytime is one of the most popular community among SKKU students. Unfortunately, that makes more difficult to get useful informations about double majoring. The community is too broad; they deals with lots of topics such as campus life, hobby, getting jobs, etc. But there is no room for double majoring students.

4 Proposed Solution

We proposed an mobile application to make a community of double majoring students. Here are some detailed explanations about its functionalities.

4.1 Tech Stack

For frontend development, we decided to use Flutter as our framework. First of all, both of our frontend development members are familiar to use them. Also, Flutter support cross-platform development, which means that we can develop both Android and IOS applications in a single code base.

For backend development, we decided to user NestJS as our main framework and Typescript as our programming language.

4.2 Proposed features

Requirements for graduation The university offers the file that explains the requirements of graduation, but it might be changed irregularly. For example, department of Mathematics integrated the sections into a single section of credit requirements. We are going to give these requirements to users up to date. Main page would be hard-coded based on the user's information, while sudden changes would be posted via reports from users.

Community(Boardroom) All of the users would share a single boardroom rather than divide under a standard. Since each user has different original/double majors, we are going to give a filtering options. All posts should contain tags that match their content. We will provide some basic tags such as the department and the type of the information. Users can also add their own tags. This allows users to save and search tags that suit their interests easily.

It will also support the ability to save the posts that they like so that they can be easily found later. Comments and likes will help the writer and other uses interact. It will also support the ability to sort the posts according to the criteria such as tags and the number of scraps or likes.

This would help users who are double majoring or hope to doing so easily get the information of their interests.

Square Inspired by the LC system, the sqaure is a function that we intend to provide. To solve the problem of lack of opportunities for interaction among multiple major students, we would like to help create meetings among users. They can create new meetings to gather people with common interests or participate in existing meetings. Based on the information on the profile, we will also support the function of recommending users who seem to match the meeting they created and allowing them to be invited.

Mock apply Students who are not double majoring yet but interested to doing so often curious about the cutline for applying double majoring. For these users, the number and top percentage of applicants for the major will be provided based on the GPA users entered. It is not the purpose of analyzing accurate prediction information, and it will be held as an event for the inflow of double majoring students every semester.

Register and personal information protection The information of users will be used only for service provision purposes. The service can be used only after being authenticated by email from SKKU(g.skku.edu or skku.edu). Information on users' student number, major, and (supposed) double major must be entered. In addition, in order to revitalize the community, all users have to post information related to their original majors. Unless, they will not be able to access the posts from the community.

To ensure anonymity, users can use a nickname. Other information can be viewed through the profile, and we will support setting to choose the range of information sharing.

Proposed schedules and Roles The tables blow show our team's proposed schedules and role distribution.

Table 1: Weekly schedules

Week	Schedule
1 to 4	Project Topic Refinement and Proposal Writing
5 to 11	Application Development
12	Testing
13	Application Deployment And Preparing Final Presentation
14 to 15	Final Presentation

Table 2: Member roles

Kind	Description	Members
Backend	Building an application server	Jaehyeon Kim, Jisoo Park
Frontend	Building a mobile application	Woojin Kwon, Chaeeun Lee
Deployment	Publishing Mobile App and Server	Jisoo Park

5 Implementation

Here we discuss our implementation in detail. The project is divided into two major parts: frontend and backend

5.1 Frontend

Register Page We used our own email authentication API to implement register page. Every API function is implemented on the basis of http requests and responses. To do so, we used 'http' library to communicate between our server and client application. By using this, we can ensure that our user has a valid SKKU email account. In other words, we can assume that the user who complete email authentication is a SKKU student. Also, we created a drop-down menu to distinguish between student who are already double majors and those who are not yet. To the next page, we collected users a picture that represent themselves as a profile image. To implement this feature, we used the 'image picker' library. Finally, to support DM feature, we collected users interests.

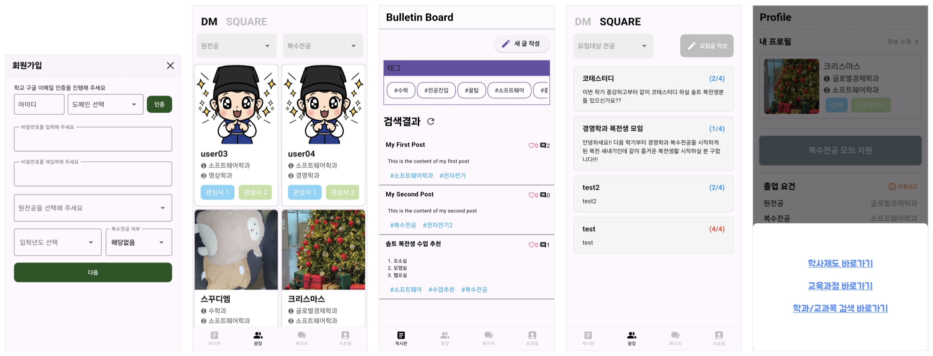
Login Page Throughout the login button, a complex process run over the client and server. First of all, a login request is sent to the server with user's input. Then the server checks if the id/pw information is valid or not. Then if the information is valid, the server make an access token and sent it back to the client. Then the client logs in with the valid token. From this moment, every API request needs valid token to access, prevent other users using our functions invalidly or maliciously. To save the access token internally, we used flutter secure storage library. Then we set a cookie to save the login information using cookie jar and path provider. This helps us sharing login information on all of our pages.

Bulletin Board Page We used multiple selection menu to support tag-based search. To implement this, we used 'multi select flutter'. Users can change the combinations of tags and press refresh button to fetch appropriate search results. In the main page, we showed the basic information of each post: the title, the abbreviated content, the number of likes and comments, and tags. In the post page, users can check the whole contents and comments. If they are the writer of the post, they can either update the post or remove it by pressing the "Update" button. Of course if the user is not a writer, he or she will be refused to do so. At the below of the content, users can check comments. They can post new comments. To change the format of created time of post or comment, we used intl.

DM Page DM stands for both Direct Message and Double Major. We decided to add this feature to support personal connection between users based on their interests. Users can search for other users based on their primary and secondary majors. Then they can find out a list of users who agree to search public. They can send a direct message by tapping the profile of target user. In this manner, we expected that users can interact easily and freely. In order to emphasize the title, we used both Google Fonts to change the fonts.

Sqaure Page Unlike DM, Square supports encouraging users to make small private groups. They can post based on the majors or interests, but if they need more specific conditions, they might write down on contents. While users are gathering others to join, they can communicate on comments. If the gathering is done, a group chatting room will be created and the members can communicate on the Message page.

Message Page In the message page, users can check their all of chatting rooms from both DM and Square features. We decided to separate them because some students and professors felt confused on the difference between DM and Sqaure.



(a) signup page (b) dm page (c) board page (d) square page (e) profile page

Fig. 1: flutter implementations

Profile Page Users can check their informations: profile pictures, majors, and their interests. They can adjust them by clicking the "update information" menu. By implementing the profile page, we figured out that some of majors such as Culture and Technology would be too long to show up in a fixed box. To solve this problem, we used 'auto size text' to automatically adjust the size of text according to the name of majors of users.

Mock apply feature was implemented on backend, ready for API functionalities. But due to the time limit, we couldn't finish integrating them to our application, so it is leaved as unimplemented menus.

We planned to provide graduation requirements to users with an intuitive and easy-to-see UI. However, due to the time limit, we couldn't implemented this feature. Instead, it was replaced with links that could search for academic systems, curriculum, and departments/subjects according to the information of users. We used 'url launcher' to add links.

5.2 Backend

Database Schema The user table serves as the core of the system, storing information related to user accounts. It manages details such as user ID, username, email, password, role, and admission year. Each user's profile information is extended through the profile table, which stores profile image URLs, introductions, interests, and profile visibility settings.

Users are connected to majors through the major table, and their enrolled or interested majors are managed by the user major table. The user major table handles the many-to-many relationship between users and majors and tracks whether a particular major is a user's primary major. The major table stores the name and metadata of each major in JSON format, allowing for the storage of additional information about majors.

Additionally, users can belong to various communities or groups, which are managed by the square table. This table stores information about community names, associated majors, and maximum participant limits. The relationship between users and communities is managed through the user square table, which indicates the specific communities a user belongs to. Within a community, user-created posts and comments are managed through the square post table and the square post comment table, facilitating communication within the community.

General forum activities are managed through the post table and the comment table. The post table stores information such as post titles, content, authors, tags, and likes, with each post connected to multiple comments. The comment table is designed to support nested comments by storing parent comment IDs, enabling a hierarchical comment structure.

Lastly, the functionality for users to simulate applying to specific majors is handled by the mock apply table. This table references user IDs and major IDs and records application scores, providing data for users' selected majors. (The related API implementation has been completed, but due to time constraints, the frontend integration remains unfinished.)

Firebase was used for implementing the chat functionality, and Firestore was utilized for storing chat history. Firebase, as a serverless backend solution, was chosen because it allows for easy implementation of chat features without the need for complex WebSocket configuration within the NestJS framework.

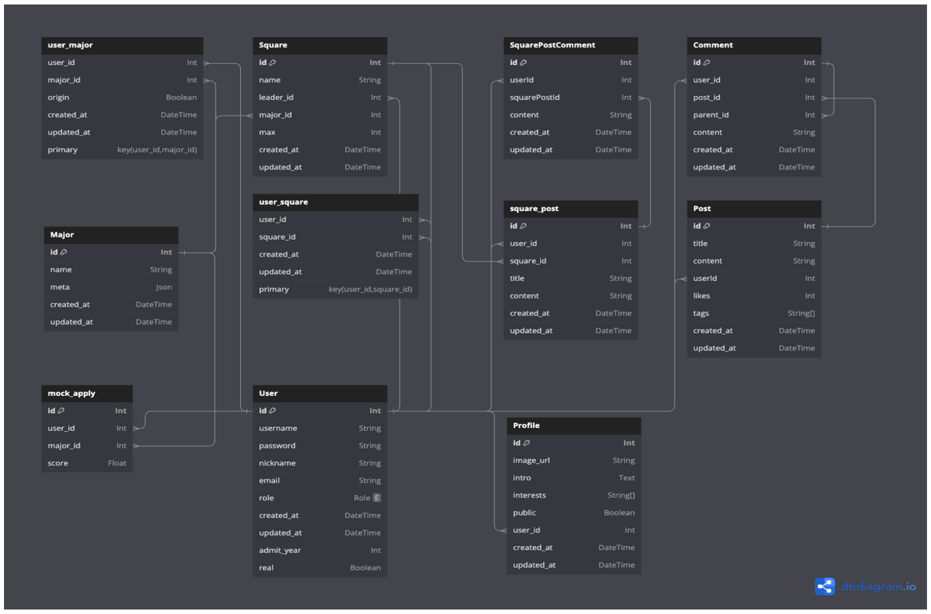


Fig. 2: Database schema

Infrastructures For the backend deployment environment, we configured the infrastructure environment using Terraform code and deployed it to AWS. Terraform, known as Infrastructure as Code (IaC), has the advantage of being able to describe infrastructure architecture through declarative programming. Instead of accessing the AWS console and dealing with complex environment configurations, having Terraform code allows you to easily replicate the same infrastructure structure.

More specifically, we purchased the domain skku_dm.site and registered it with Route53. We also configured VPC and Security Groups to ensure that the EC2 container hosting the database cannot be accessed by unauthorized users. Static files such as user profile images were stored in S3, and we attached CloudFront in front to enable access to static resources through the domain address. Finally, we used Firebase to implement the chat functionality.

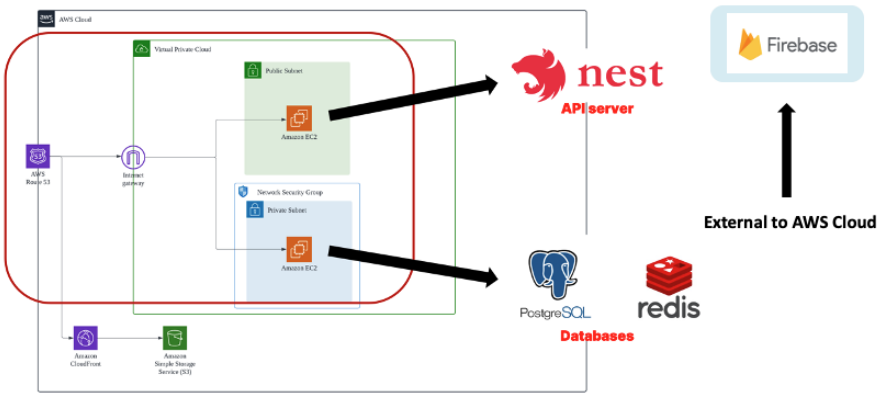
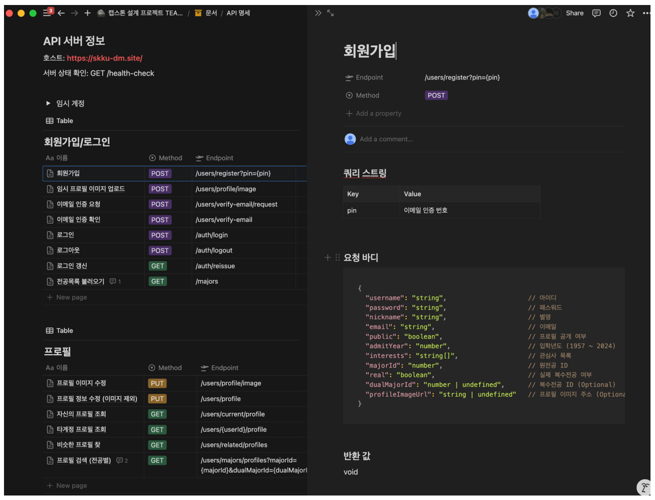


Fig. 3: infrastructure overview

API Implementations and API Documentations To facilitate communication between frontend and backend team members, API documentation was created using Notion after API development. Each API document specifies the required HTTP method, URL query strings, and URL parameters. Additionally, the documentation includes examples of API requests and responses, aiding frontend developers in better understanding the APIs when connecting the frontend and backend. When necessary, the required HTTP request body is also detailed in the documentation to ensure clarity during development.



API 서버 정보

호스트: https://skku-dm.site/

서버 상태 확인: GET /health-check

▶ 임시 계정

Table

회원가입/로그인

Aa 이름	Method	Endpoint
회원가입	POST	/users/register?pin={pin}
임시 프로필 이미지 업로드	POST	/users/profile/image
이메일 인증 요청	POST	/users/verify-email/request
이메일 인증 확인	POST	/users/verify-email
로그인	POST	/auth/login
로그아웃	POST	/auth/logout
로그인 갱신	GET	/auth/reissue
전공목록 불러오기 1	GET	/majors

Table

프로필

Aa 이름	Method	Endpoint
프로필 이미지 수정	PUT	/users/profile/image
프로필 정보 수정 (이미지 제외)	PUT	/users/profile
자신의 프로필 조회	GET	/users/current/profile
타계정 프로필 조회	GET	/users/{userId}/profile
비슷한 프로필 찾	GET	/users/related/profiles
프로필 검색 (전공별) 2	GET	/users/majors/profiles?majorId={majorId}&dualMajorId={dualMajorI

회원가입

Endpoint /users/register?pin={pin}

Method POST

쿼리 스트링

Key	Value
pin	이메일 인증 번호

요청 바디

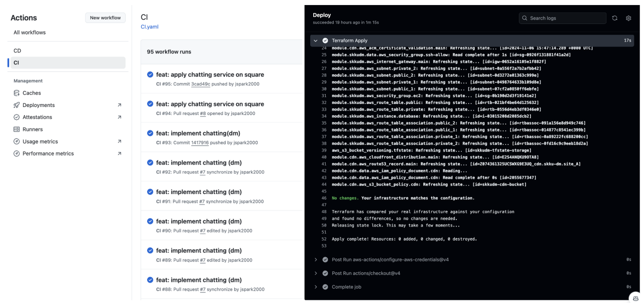
```
{
  "username": "string",                    // 아이디
  "password": "string",                    // 패스워드
  "nickname": "string",                    // 별명
  "email": "string",                       // 이메일
  "public": "boolean",                     // 프로필 공개 여부
  "admitYear": "number",                   // 입학년도 (1957 ~ 2024)
  "interests": "string[]",                 // 관심사 목록
  "majorId": "number",                     // 원전공 ID
  "real": "boolean",                       // 실제 복수전공 여부
  "dualMajorId": "number | undefined",     // 복수전공 ID (Optional)
  "profileImageUrl": "string | undefined"  // 프로필 이미지 주소 (Optiona
}
```

반환 값

void

Fig. 4: API documentation

Continuous Integration and Continuous Deployment The CI/CD functionality was implemented using GitHub Actions. GitHub Actions is a computing runtime provided by GitHub that performs specific processes based on user-predefined YAML file contents. For CI (Continuous Integration), we configured it to include type checking and build testing of the backend NestJS project, execution of backend NestJS test codes, and backend dockerization testing to verify that there are no issues with the backend code currently in the main branch. For CD (Continuous Deployment), it can be executed by users with administrator privileges in the Actions tab of the GitHub project repository. We implemented it to reflect Terraform changes to AWS, upload the backend Docker image to the GitHub image repository ghcr.io, and then update the image of the backend Docker container running on EC2.



CI

CI.yaml

95 workflow runs

feat: apply chatting service on square
CI #95: Commit 3cad49c pushed by jspark2000

feat: apply chatting service on square
CI #94: Pull request #8 opened by jspark2000

feat: implement chatting(dm)
CI #93: Commit 1417916 pushed by jspark2000

feat: implement chatting (dm)
CI #92: Pull request #7 synchronize by jspark2000

feat: implement chatting (dm)
CI #91: Pull request #7 synchronize by jspark2000

feat: implement chatting (dm)
CI #90: Pull request #7 edited by jspark2000

feat: implement chatting (dm)
CI #89: Pull request #7 edited by jspark2000

feat: implement chatting (dm)
CI #88: Pull request #7 synchronize by jspark2000

Deploy

succeeded 19 hours ago in 1m 15s

Terraform Apply 17s

```
module.cdn.aws_acm_certificate_validation.main: Refreshing state... [id=2024-11-06 15:47:14.289 +0000 UTC]
module.skkudm.data.aws_security_group.ssh-allow: Read complete after 1s [id=sg-0926f131881f41a2d]
module.skkudm.aws_internet_gateway.main: Refreshing state... [id=igw-0652a16105e1f882f]
module.skkudm.aws_subnet.private_2: Refreshing state... [id=subnet-0a556f2a7b2afbb42]
module.skkudm.aws_subnet.public_2: Refreshing state... [id=subnet-0d3272e01363c999e]
module.skkudm.aws_subnet.private_1: Refreshing state... [id=subnet-0498764633b109d8e]
module.skkudm.aws_subnet.public_1: Refreshing state... [id=subnet-07cf2a0850ff6ebfe]
module.skkudm.aws_security_group.ec2: Refreshing state... [id=sg-0b190d2d3f19141e2]
module.skkudm.aws_route_table.public: Refreshing state... [id=rtb-021bf4be64d125632]
module.skkudm.aws_route_table.private: Refreshing state... [id=rtb-0556d4eb3df0346e0]
module.skkudm.aws_instance.database: Refreshing state... [id=i-03015208d2085dcb2]
module.skkudm.aws_route_table_association.public_2: Refreshing state... [id=rtbassoc-091a156e8d949c746]
module.skkudm.aws_route_table_association.public_1: Refreshing state... [id=rtbassoc-014877c8541ec399b]
module.skkudm.aws_route_table_association.private_1: Refreshing state... [id=rtbassoc-0a89222fc688200cc]
module.skkudm.aws_route_table_association.private_2: Refreshing state... [id=rtbassoc-0fd16c9c9eeb18d2a]
aws_s3_bucket_versioning.tfstate: Refreshing state... [id=skkudm-tfstate-storage]
module.cdn.aws_cloudfront_distribution.main: Refreshing state... [id=E2S4ANQKU9OTA8]
module.cdn.aws_route53_record.main: Refreshing state... [id=Z074361325UCSWXG9E3UQ_cdn.skku-dm.site_A]
module.cdn.data.aws_iam_policy_document.cdn: Reading...
module.cdn.data.aws_iam_policy_document.cdn: Read complete after 0s [id=2055677347]
module.cdn.aws_s3_bucket_policy.cdn: Refreshing state... [id=skkudm-cdn-bucket]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration
and found no differences, so no changes are needed.
Releasing state lock. This may take a few moments...

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

```

Post Run aws-actions/configure-aws-credentials@v4 0s

Post Run actions/checkout@v4 0s

Complete job 0s

Fig. 5: Github action

6 Limitations and Discussions

There are some limitations on our project.

6.1 Unfinished Implementations of features

There are some features that planned earlier fails to be fully implemented due to the unexpected problems. First of all, conditions of implementation features are changed often during the implementation due to the unexpected situations that we didn't find out earlier. For example, our plan to implement chatting feature was pulling, but after the last progress meetings, we found out that using websockets would be better in terms of server load managing. By far, we changed our code totally. It took extra time for us to deal with those kind of unexpected issues. Also, debugging and testing process took much more time than expected. For example, API connection process was not smooth. We found out that we didn't synchronized the specifications of API functions. To match our requirements, both frontend and backend side need extra works.

7 Conclusion

Although the plan is not perfectly executed, we successfully made a mobile application.

7.1 Synergy of Technology Stack

AWS cloud infrastructure provided a stable backend environment, offering significant advantages in terms of scalability and security NestJS helped build a systematic backend architecture based on TypeScript, making code management efficient through its modular structure Flutter enabled cross-platform development for both iOS and Android, greatly improving development efficiency

7.2 Development Challenges

Managing and optimizing data flow between three different technology stacks was a significant challenge Adapting to Flutter's widget system and implementing responsive UI required new learning Efficiently configuring AWS services and optimizing costs was also an important experience

7.3 Learning Outcomes

Full-stack development experience enhanced understanding of the entire application architecture Gained practical experience in deployment and operations in a production environment Recognized the importance of effective communication and version control through team collaboration

7.4 Future Development Directions

Advanced learning of each technology will enable more efficient architecture design Additional learning may be needed for performance optimization and user experience improvement Can develop further in the direction of strengthening security and test coverage This experience will be a valuable asset in the actual industry. Particularly, the process of creating a complete product by integrating multiple technologies was a great opportunity for growth as a software engineer.

8 Appendix

8.1 Used Flutter Libraries

multi_select_flutter Multi Select Flutter is a package for create multi-select widgets. We used it to implement multiple selection menus for selecting tags. [2]

image_picker Image Picker is a Flutter plugin for picking images from the image library, and taking new pictures with the camera. We used this package to get image from users to register profile image. [3]

intl Intl provides internationalization and localization facilities, including message translation, plurals and genders, date/number formatting and parsing, and bidirectional text. We used this package to parse and reformatting date. [4]

http HTTP is a composable, Future-based library for making HTTP requests. We used this package to integrate our APIs into our application. [5]

cookie_jar Cookie jar is a cookie manager for http requests in Dart, help us to deal with the cookie policies and persistence. We used this package to manage access token as cookies. [6]

path_provider Path provider is a Flutter plugin for finding commonly used locations on the file system. We used this package to save the login information through all of our code bases. [7]

Google Fonts Google Fonts is a library that helps developers use Icons, fonts, etc. We used it to modify icons and fonts in our app. [8]

auto_size_text Auto size text is a Flutter widget that automatically resizes text to fit perfectly within its bounds. We used this package to fit the informations in the profile page. [9]

8.2 Other Frameworks

NestJS(Backend) A progressive Node.js framework for building efficient, scalable server-side applications using TypeScript and object-oriented programming principles.[10]

Terraform(Infra) An infrastructure as code (IaC) tool that enables you to safely and predictably create, change, and improve infrastructure across multiple cloud providers.[11]

References

1. "Everytime," everytime.kr, `https://everytime.kr/`
2. "multi_select_flutter," pub.dev, `https://pub.dev/packages/multi_select_flutter`
3. "image_picker," pub.dev, `https://pub.dev/packages/image_picker`
4. "intl," pub.dev, `https://pub.dev/packages/intl`
5. "http," pub.dev, `https://pub.dev/packages/http`
6. "cookie_jar," pub.dev, `https://pub.dev/packages/cookie_jar`
7. "path_provider," pub.dev, `https://pub.dev/packages/path_provider`
8. "Google Fonts," Google, `https://fonts.google.com/`
9. "auto_size_text," pub.dev, `https://pub.dev/packages/auto_size_text`
10. "NestJS," nestjs.com, `https://nestjs.com text`
11. "Hashicorp Terraform," terraform.io, `https://terraform.io text`