BloomMate

House Gardening with AI Chat using LG Tiiun

An Soonho
dept. Information Systems
Hanyang Univ.
Seoul, Republic of Korea
ash1823@hanyang.ac.kr

Kim Donghyun
dept. Information Systems
Hanyang Univ.
Seoul, Republic of Korea
akainoo@hanyang.ac.kr

Shin Hyunah
dept. Information Systems
Hanyang Univ.
Seoul, Republic of Korea
hyunah0923@hanyang.ac.kr

Yoon Yongsung
dept. Information Systems
Hanyang Univ.
Seoul, Republic of Korea
mirr001003@hanyang.ac.kr

***Abstract*—BloomMate is a user-friendly gardening application tailored for beginners or individuals with limited gardening experience, particularly those who have acquired the LG Smart Cottage and intend to optimize their gardening with the LG Tiiun. This app simplifies the gardening process by allowing users to register their plants and access a range of personalized guidance to enhance their gardening skills.**

The standout feature of BloomMate is the unique capability to communicate with your plants. Through generative AI technology, this app facilitates direct interaction with plants, creating a personalized and engaging experience for users. Moreover, it exploits AI to identify and diagnose plant diseases, offering valuable insights to maintain the health of your pet plants.

***Index Terms*—gardening, generative AI, Matter, AI**

I. Role Assignment

Below is the table of role descriptions. Due to the small members in the team, some roles are distributed to the same person in order to harmoniously progress the project.

Roles	Name	Task description & etc.
User	Shin Hyunah	She plays the role of a plant grower. She has little experience growing plants and doesn't know how to water and nourish them. Additionally, she often kills her plants because she doesn't have time to take care of them, so she needs a service that takes care of her plants automatically.
Customer	Yoon Yongsung	He is a person who owns a house, and in particular, he has purchased an LG Smart Cottage or has an LG Tiiun. He goes to his house on the weekends, enjoys country life, and takes care of plants. He wants software that automatically manages his plants even on weekdays when he is away or on weekends when he cannot be there.
Software developer	An Soonho	As a software developer for our project, he designs and creates APIs and databases, and builds the AI that can implement the requirements of the service we plan.
Development manager	Kim Donghyun	He is responsible for the overall management of our project, distributing each task, setting and managing task deadlines. Additionally, he helps resolve any issues between the backend and frontend.

II. Introduction

A. *motivation*

a. Customer Trend Analysis

1) positive effect of pet plant
Pet plant is known for positive effect to give psychological/emotional stability and inspiration about life satisfaction. Especially, it is showing clear tangible results in several areas like overcome stress and depression, therapy various physical illnesses, air purification.[1]

2) The appearance of Green Hobby
As interest in planterior increases, sales of flower gardens/flowers increased compared to the previous year (2019). The search volume of 'planterior' on Naver, a search portal site, also increased significantly during the pandemic compared to pre-COVID-19, especially during the April 2020 pandemic. Also, attention in urban farmer, self-gardening, weekend farm is growth. An indicator that can confirm this is the number of Instagram hashtag posts, with the home gardening tag exceeding 520,000 and the planterior tag exceeding 163,000. [2]

3) Impact of COVID-19
Pet plant has long received attention to relieve loneliness in single-person households or the elderly, and the home gardening market, including pet plant, has grown rapidly as restrictions have arisen on outdoor activities since the COVID-19. Compared to before COVID-19, sales of flowerpots increased by 20%, gardening supplies increased by 48%, and seedlings increased by 92%. March to April 2020, SSG.com saw a 97% increase in home gardening category sales, and Enuri.com saw a 491% increase in garden set sales. Accordingly, the Rural Development Administration predicted that domestic home gardening sales will increase rapidly after COVID-19, reaching 500 billion won in 2023, an eight-fold increase from 60 billion won in 2020. [3]

4) Expanding the market size of plant cultivation machines
The market for plant growers that allow you to grow flowers and vegetables at home becomes larger. The market size, which was 10 billion won in 2019, increased to 60 billion won in 2020 and is expected to reach 500 billion won in 2023. [4]

5) Urban agriculture
The number of 'urban farmers' in Seoul, who farm on rooftops, verandas, and weekend farms, increased 14-fold from 45,000 in 2011 to 640,000 in 2019. During the same period, Seoul's urban agricultural space also increased 6.9 times from 29ha to 202ha. [5]

6) Desired residential space after retirement
In 2021, the real estate platform 'Zigbang' conducted a survey on the residential space desired after retirement, and 'country house, townhouse' ranked first with 38%, beating apartments (35.4%). When comparing housing transactions in the first half of the year by housing type, apartment transactions decreased in June compared to January, while house transactions increased. [6]

Considering these data, interest in horticulture such as home gardening and planterior is steadily increasing, and along with this, interest in weekend farms, vacation homes, country houses, etc. is also steadily growth. Therefore, it can be expected that the demand for gardening or growing plants in country houses will gradually get larger. At the same time, sales of plant-related home appliances will also become greater.

b. Company Analysis

1) Regulations related to the companion plant industry
As interest in pet plants grew, the Seoul Metropolitan Council passed the 'Ordinance on Fostering and Supporting the Pet Plant Industry' at the plenary session in May, laying the foundation for the growth of the pet plant industry and focusing on expanding supporting projects.[7]

2) LG Electronics: Innovation for a Better Life
LG Electronics is currently trying to change from being a company that only sells simple home appliances to a company that provides a lifestyle that can change customers' lives to be smarter through applications such as ThinQ.

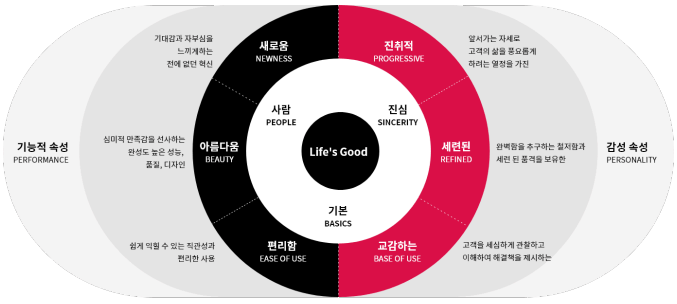


Fig. 1. LG electronics Brand Goal

Customers are always at the center of the LG brand story. LG makes an incredible contribution to the lives of its customers and always strives to make their lives more valuable.

i. PEOPLE
Just as important as a good product is the customer experience using that product. Customers' lives are improved with new products through creativity

and innovation, and a richer life is completed by experiencing products with excellent aesthetic satisfaction. Every product we create must be able to play that role in our customers' lives. Creating a better life is how we do our best for our customers.

ii. SINCERITY
How to make life new and more convenient, our technology always starts with sincerity toward customers and their lives. It is an attitude of constantly making passionate efforts and pursuing sophistication through thorough perfection rather than flashiness, and the most important factor when creating products is sincerity.

iii. BASICS
Rather than being different for the sake of being different or flashy for show, we focus on the core values that customers who use the product expect. The core value refers to offering important value to customers' lives beyond the product's functions. [8]

c. Product Analysis

1) LG Smart Cottage
It is a small modular house in the form of a second house that combines LG Electronics' advanced energy, heating, and cooling technology, and differentiated premium home appliances, with a two-story one-room structure measuring 31.4m^2. The 4kW solar panel installed on the roof can cover a significant amount of the electricity used per day by two adults, and the heat pump cooling and heating system 'Therma V Monobloc' is installed to significantly reduce energy consumption. A home energy solution was implemented that stores used power in a home ESS system. There is a charger for electric vehicles (EV) outside the Smart Cottage, and the inside is equipped with premium home appliances equipped with various technologies to increase energy efficiency, such as the Objet Collection Wash Tower Compact, a dishwasher, an induction electric range, and a water purifier. Through the 'LG ThinQ' application, you can check home appliances, heating, cooling, and air conditioning (HVAC) systems, and energy storage and consumption. It has a premium interior with a warm atmosphere, with a sophisticated design based on the concept of walnut wood tone and an exterior finished in light beige.

Fig. 2. LG Smart Cottage

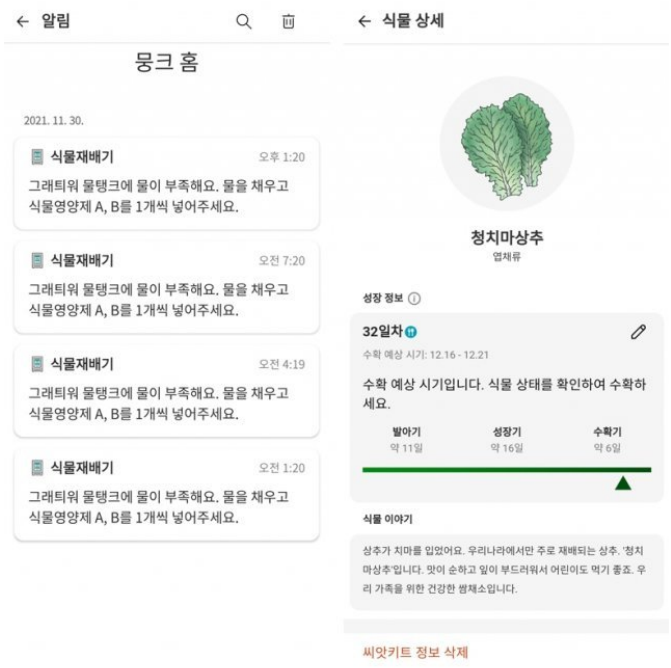
2) LG Tiiun

숙박업소의 소형 냉장고 만한 크기다 (출처=LG전자 홈페이지)

Fig. 3. LG Tiiun

LG Tiiun is a home plant grower designed to target the home gardening market, which is expanding due to the impact of COVID-19. It can be said to be the culmination of LG's technologies, combining the temperature control technology of the Dios refrigerator, the air conditioning technology of the Whisen air conditioner, the inverter compressor technology of the Tromm dryer, the water supply control technology of the PuriCare water purifier, and the LED lights of LG Display. You can easily and comfortably grow plants by filling them with water 6 times and adding nutrients 6 times in a month. The power consumption for one year is 42.5kWh, which is similar to that of a small refrigerator. LG Tiiun consists of two shelves (upper and lower compartments), and a total of six seed kits can be installed, three each. Seed kits limited to 2 types of herbs, 4 types of leafy vegetables, and 1 type of flower are being sold. Through the LG ThinQ app, you can

check the operation/growing status on your smartphone and receive a notification when nutrients or water are insufficient.



LG ThinQ 앱을 통해 스마트폰으로 동작/재배 상황을 확인할 수 있다 (출처=IT동아)

Fig. 4. Example of using ThinQ for LG Tiiun

B. Problem Statement

a. LG Smart Cottage: absence of home turf
People live in house almost enjoy gardening. However, LG Smart Cottage that LG Electronics launched doesn't have home turf. So, we think LG Tiiun for outside to help with LG's brand goal is necessary. Especially in the case of smart cottages, many people buy them as a second home and are unlikely to be there every day. In this case, they may struggle to get information about the condition of the plants, unless they have restrictions except for gardening with Tiiun.

b. Brown Thumb
In abroad, the term, brown thumb, means the people who always kill the plant because of lack of skills. Like this, there are more things to worry about in growing plants than you think, so if you don't pay attention, you often kill the plants, and even if you care a lot, there are many cases where you kill the plants because you don't know how to grow them properly. Therefore, Software, to help all of the plant growers can grow them easily, is essential.

c. Rasing Plants: absence of communication
Although raising pet plants is already known to help with various emotional stability, it was judged that there was less of a feeling of communication with each other compared to animals with which one could actually communicate. In order to grow plants well, constant attention is needed, and for this to happen, it is important to have affection for the plants. Therefore, in order to increase the fun of growing plants, it is necessary to be able to communicate with plants.

C. Solution

a. Meeting the needs of more customers
Considering the aforementioned, the number of people interested in gardening, cottages, etc. will increase. To contribute to increasing the value of life for the growing number of caterers' customers who grow their own gardens in cottages, we have adopted LG Tiiun which can be used in the outside garden of LG Smart Cottage as a target appliance.

b. Brown thumb turns into green thumb with LG
We have designed a software that allows people who are not talented in plant care to easily grow plants. To briefly introduce the features, customers can register their plants with the app and it will adjust the appropriate temperature, humidity, light, ventilation, etc. according to the plant species and automatically spray water and nutrients. Through the camera, the current health status of the plant (pest infestation, wilting, etc.) can be identified using AI, so that the customer can come and check the plant before it dies.

c. Build a bond with plants
One of the disadvantages of growing plants is that customer can only get an idea of their condition by looking at the wilting of their leaves or the appearance of fruits or flowers, and can't communicate with them directly like animals. These disadvantages reduce the fun of growing plants. To overcome this, we devised a feature that allows customer to communicate with plants using generative AI. If customers can communicate with their plants, they will be able to build a bond with them and give them more affection.

D. Research on Any Relative Software

a. Groo
Groo is an application for plant care scheduling, community, plant books, and plant-related purchases. You can enter your experience with plants and where you grow them, and it will recommend the best plants for your space. You can also take photos of your plants to learn about their types and care, and use the plant guide to find out what to look out for. You can also communicate with other people who are interested in

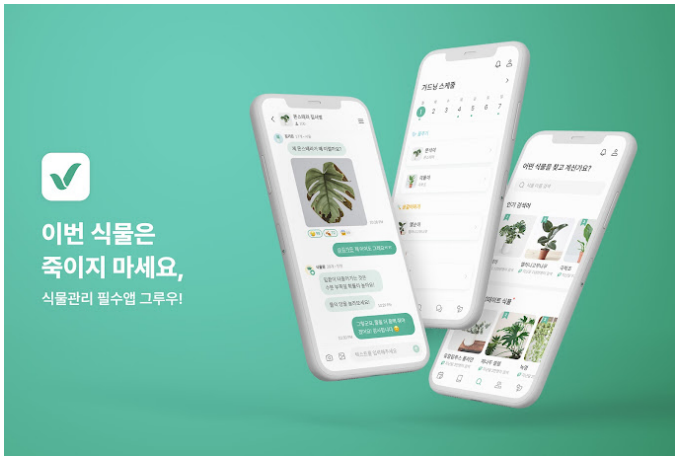


Fig. 5. Groo

growing plants through the community and buy gardening supplies. What makes it different from other plant growing applications is that it uses vision AI to check the health and disease of plants.

Fig. 6. Diagnosis of plant health using vision AI

b. Planta
Planta is an application that helps you grow plants, similar to Groo. You can add information about your plants to the app and it will automatically remind you when to water, fertilize, spray, clean, etc. based on an algorithm. You can even take a photo of your plant and get the information you need by identifying the plant's variety. You can also get plant advice, keep a diary to track your plants' growth, and more. One of the unique features is that the app also recognizes where you live and recommends plants that are suitable for your hardiness zone and adjusts the watering schedule according to the local weather.

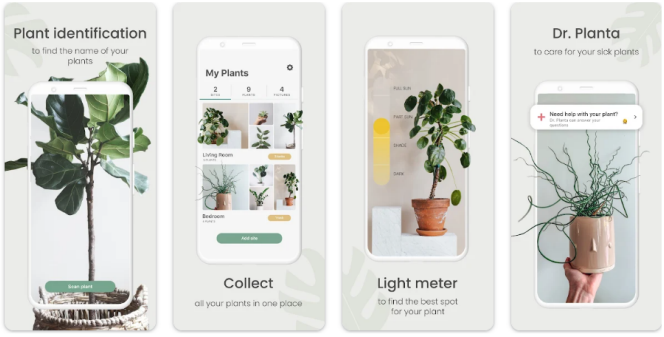


Fig. 7. Planta

c. PlantLink
PlantLink is an application that allows you to manage plants by connecting the Soil Module, an IoT device. Through the IoT device, you can record changes in soil humidity, illumination temperature, etc. to create a growth report. It is possible to talk to plants through chatbots and conduct psychological analysis through these conversations to provide mental health reports and receive plant management advice, but it is currently not working smoothly.

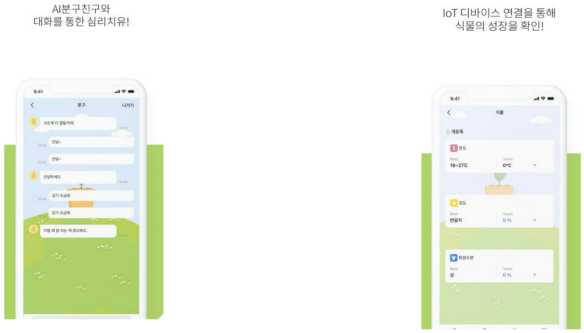


Fig. 8. PlantLink

d. Greg
Greg is the zero-guesswork plant care app & community. It makes growing indoor plants super easy and fun. If you scan any plant, Greg's plant vision will identify the species, tell you all about it, measure the pot, and even the distance to the nearest window. And it provides highly-personalized plant care, a custom watering plan based on each plant's species, size, plus the actual details of your home environment. Moreover, you can communicate with other veteran growers for answers to any plant questions you have.

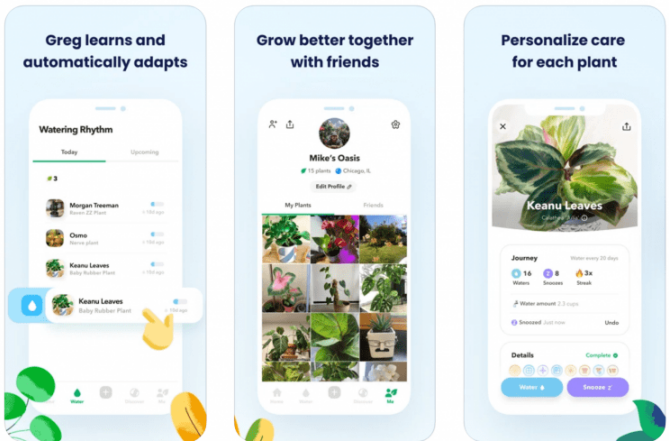


Fig. 9. Greg

III. REQUIREMENT ANALYSIS

A. Create an account

When you first install and run the application, you'll see a page where you can sign up or log in. If you click Sign Up, we'll ask you a few questions to get your information and create your account. The following information is required from users when signing up.

- Name
- Picture
- Location of home
- The size of the garden
- ID and Password
- The serial number of users' LG Tiiun

After signing up, you'll be taken to the login page. You can edit all the information you received during the signup process on your My Page.

B. Log in

Logging in is an essential step for every member to access and manage their own garden on BloomMate, and it's the gateway to utilizing BloomMate's full range of features. The login process involves entering the ID and password you provided during registration. Once successfully logged in, you'll be directed to a screen that allows you to either add new plants or view the plants you have previously registered, depending on whether you have any plants registered. The login requirement is a fundamental aspect of the BloomMate platform. It empowers users to have a personalized gardening experience, enabling them to manage their garden with ease. The authentication through the use of an ID and password is crucial for maintaining account security and ensuring the protection of sensitive information.

C. Add own plant

Logged-in users who don't have any registered plants will be taken to the plant registration screen. The plant registration process is essential for managing plants and chatting with them. To register a plant, users need to enter some information. The following is a list of information required to register a plant.

- Name of plant
- Plant variety
- Photo of plant
- Planted date

At this stage, you have the following constraints.

- Limited plant types
  - Provide application services within a limited plant to provide optimal personalized information.
- Limited number of plants based on the size of your garden - Warns that planting more plants than the size of your garden can accommodate may result in poor gardening.

After entering the above information, the user can view their plants on the registered plants screen.

D. See the registered plants

Your registered plants will appear in a list, showing the name of the plant you entered when you registered it and how many days it has been with you based on the date you planted it. Clicking on each plant will take you to a page where you can view the plant's details. If there are no plants, the user will see a screen with a button to add a plant.

E. View details of a plant

You can click on each plant to view its details, and the first thing you'll see is a photo and name of the plant you've registered. Also, you can start chatting with the plant by clicking the chat button. The plant details page contains the following information.

- Name of plant
- Plant variety
- Planted date
- Basic information based on plant variety:
  - Appropriate temperature
  - Appropriate illuminance
  - Soil moisture
  - Cultivation difficulty
  - Flowering season
  - Watering cycle
  - Caution

    etc.

You can also take a photo to detect the current status of a plant by clicking checkup button.

F. Chat with your plant

Users can converse with their plants using ChatGPT, a popular generative chat AI. This AI assumes the role of the

plant, providing regular updates on its condition and growth. In addition to standard plant status and growth records, users can engage in daily conversations with their green companions. ChatGPT shares insights into the plant's current state, addresses questions and concerns, and offers care advice. Critical information, such as plant status and growth records, is automatically delivered daily and accessible on the chat screen. Users can easily search for past information by date in the chat history. This feature fosters a stronger connection between users and their plants, enhancing the gardening experience.

G. Examine plants through photos

Users have a convenient way to assess the health of their plants by simply taking a photo. When users capture an image of their plant, leaves and all, an AI model steps in to evaluate the plant's current health status and check for any potential diseases. This feature not only informs users about the plant's overall condition but also provides guidance on how to manage any issues that may arise. If there are any health concerns, the AI not only identifies the issue but also offers detailed information about the specific disease, if present. Furthermore, it goes the extra mile by suggesting effective methods for addressing the problem, whether it's a common issue that can be easily resolved or a more complex condition requiring special care.

H. My page

My page is where you can view your membership information. You can take the following actions from here.

- View user's name and picture
- Modify account information
- View about BloomMate

IV. DEVELOPMENT ENVIRONMENT

A. Choices of Software Development Platform

a. Development Platform

1. Windows
   Windows is an extremely popular and widely used operating system for software development, known for its versatility and user-friendly interface. Currently, Windows is the pre-installed OS on most computers in South Korea, far outpacing other OSes in terms of share. It offers a multitude of features and supports a vast array of programming languages and tools, making it the go-to choice for developers worldwide. With Windows, developers have the flexibility to utilize various development environments and frameworks, empowering them to create applications efficiently and effectively. Whether it's desktop applications, web development, or mobile app development, Windows provides the necessary tools and resources to meet the demands of any project. Additionally, Windows continually evolves and updates its platform, ensuring that developers have access to the latest technologies and advancements in the software development field. In conclusion, Windows is an indispensable platform for software development, offering a comprehensive ecosystem and empowering developers to bring their ideas to life.

2. MacOS
   MacOS is a popular operating system favored by many developers, especially those in the creative and design fields. Known for its sleek and intuitive user interface, MacOS offers a seamless development experience. It provides a wide range of powerful tools and frameworks, making it an excellent choice for building applications for various platforms. With MacOS, developers can take advantage of tools like Xcode, which is the primary integrated development environment (IDE) for Apple's platforms. This is especially important when the application we want to develop must also support IOS. To develop for iOS, Xcode is essential, and Xcode can only run on MacOS. Xcode offers a comprehensive set of tools and resources for developing iOS, macOS, watchOS, and tvOS applications. Additionally, MacOS has a strong integration with other Apple products and services, allowing developers to create a cohesive ecosystem across different devices. Overall, MacOS provides a robust and efficient development environment for developers looking to create software for Apple.

b. Tools and Language

1. JavaScript(JS)
   JavaScript is a widely-used programming language known for its versatility and its ability to create dynamic and interactive web applications. It serves as the backbone of modern web development and is supported by all major web browsers. With JavaScript, developers can add functionality to websites, handle user interactions, and manipulate web page content. It is a high-level, interpreted language that allows for rapid development and prototyping. After the introduction of Google's V8 engine-based Node.js, JavaScript can now be executed in a non-browser environment as well. JavaScript also benefits from a vast ecosystem of libraries and frameworks, which enhance its capabilities and simplify the development process. These libraries and frameworks are published by npm, the Node Package Manager, and developers can easily download and utilize them in their projects. Some notable features of JavaScript

include its single-threaded nature, prototype-based language structure, and its status as a script language. JavaScript is regularly updated by ECMA (European Computer Manufacturers Association), with the updated version being referred to as ECMAScript. Recent ECMAScript versions support object-oriented programming and functional programming. In summary, JavaScript is an essential language for web development, empowering developers to create engaging and interactive user experiences on the web.

2. TypeScript(TS) [9]
   TypeScript is a superset of JavaScript that provides static typing, making it a powerful tool for building robust and scalable applications. It was developed by Microsoft and first published to the public in 2012. With TypeScript, developers can catch errors during compile-time, resulting in fewer bugs and improved code quality. It offers advanced features such as interfaces, generics, and decorators, which enhance code organization and maintainability. TypeScript is fully compatible with JavaScript, allowing developers to gradually migrate their existing JavaScript codebases to TypeScript. It also seamlessly integrates with popular JavaScript libraries and frameworks, making it a versatile choice for web development. Nowadays, companies are compelled to adopt TypeScript instead of JavaScript, regardless of project size, due to its numerous benefits. Overall, TypeScript is a valuable addition to the development environment, offering enhanced type safety and productivity for developers.

3. yarn [10]
   Yarn is a JavaScript package manager that helps manage project package dependencies and facilitates package sharing among developers. Due to security and performance issues experienced with npm as their projects grew larger, Facebook developed Yarn as a replacement. Yarn offers improvements in performance and security compared to npm. One key advantage of Yarn is that it installs packages in parallel, unlike npm which installs them sequentially, resulting in faster installation times. Yarn also utilizes caching, further speeding up package installation. Additionally, Yarn utilizes semantic versioning to specify package versions. For this project, we have chosen to use Yarn due to its fast performance. This is especially important as we will be installing various frameworks and libraries for frontend development.

4. Eslint/Prettier [11]
   ESLint performs automated scans of your JavaScript and TypeScript files to detect common syntax and style errors. Prettier scans your files for style issues and automatically reformats your code to ensure consistent rules for indentation, spacing, semicolons, and quotes (single vs double). We use these tools on our teams for the following reasons:

   - They promote consistency by enforcing the same rules for everyone.
   - They save time in code reviews by allowing us to focus on code structure and semantics instead of style issues.
   - They catch errors, with ESLint being particularly effective at detecting syntax errors and basic type errors like undefined variables. Although Prettier doesn't catch errors, it still contributes to code quality.
   - Changes made by these tools are automatically applied to code when the file is saved in Visual Studio Code.

   While setting up these tools requires an initial time investment, the time-saving benefits accumulate over time.

5. React-Native [12]
   React Native is an open-source UI software framework developed by Meta Platform, Inc. It enables developers to use the React framework along with native platform capabilities to build applications for Android, Android TV, iOS, macOS, tvOS, Web, and Windows. Unlike web apps, React Native communicates with the Native Thread through native bridges, resulting in optimized performance. In terms of hybrid frameworks, Flutter has an advantage in making it easier for web developers to write React Native code. Previously, there was a perception that Flutter was much faster and lighter. However, with the introduction of Fabric in React Native 0.68, which supports faster communication between Android and iOS, that perception has become a thing of the past. That being said, React Native does heavily rely on other libraries, even for core features like the camera function in mobile applications, which requires the use of third-party libraries. Nevertheless, due to the aforementioned advantages, it is difficult to find a better option than React Native for creating fast and cross-platform applications that support both Android and iOS.

6. React Query [13]
   React Query is a library that simplifies the process of fetching, caching, and keeping your

React application's server state synchronized and updated. Unlike other data fetching methods that require complex and verbose code, React Query provides a simple and intuitive API that can be used within React Components. Previously, managing state in React applications lacked clear distinction, leading to confusion when writing code. Specifically, determining which state should be continuously fetched from the network and which state is solely dependent on the client required a thorough understanding. However, with the introduction of React Query, you can easily differentiate between server state management and client state management in your React application. This distinction enables efficient collaboration and simplifies cache operations, loading operations, and the implementation of difficult asynchronous operations in network state management.

7. Recoil [14]
   In React, it can be challenging to directly pass data between two different components unless they have a parent-child relationship. In such cases, you need to send the data to the parent component and then pass it back to the component that requires it. This process, known as props drilling, can make it difficult to keep track of the props being passed around. That's why having a global state management library is crucial for modern React applications. React's own Context API requires implementing it from scratch and requires significant effort to prevent unnecessary re-rendering. On the other hand, Redux, which was the most popular choice until 2020, has a steep learning curve and a complex architecture that can make it challenging to understand the code flow quickly. In contrast, Recoil is relatively easy to use if you understand the basic syntax of React. It is also free from re-rendering issues and provides various optional features for global state management. Additionally, Recoil is lightweight and does not significantly impact application performance.

8. Python [15]
   Developed in 1991 by programmer Guido van Rossum, Python is known for its readability and easy syntax, making it quicker to learn compared to other programming languages. As a result, it has gained popularity among non-programmers and is utilized in various fields such as statistics, data analysis, modeling, deep learning, and artificial intelligence. Python's intuitive and user-friendly syntax makes it highly recommended for beginners in programming. Additionally, it is the most widely adopted language, meaning that many libraries necessary for application development are readily available. Therefore, we have chosen Python for building the backend and training machine learning models.

9. Django [16]
   Django is a popular and widely used web application framework written in Python. It offers robust features and components for efficient website and web application development. One of its key advantages is simplifying and accelerating the development process. Django provides pre-built components for user authentication, an admin panel, forms, and file uploads. User authentication is made simple with a few lines of code, ensuring secure sign up, log in, and log out. The admin panel allows easy content and data management. Django's form handling system simplifies the implementation of complex forms with validation and error handling. Handling file uploads is effortless with Django's built-in capabilities. Additionally, Django has a vast ecosystem of libraries and extensions, such as djangorestframework for creating RESTful APIs. Overall, Django is a versatile and powerful framework for building robust and scalable web applications, suitable for both beginners and experienced developers.

10. SQlite [17]
    Since SQLite doesn't run as a server process, it's a ready-to-use database. It's lightweight but has all the necessary features. It supports transactions and is platform-agnostic. Since it's a locally run DB, it's also highly portable. However, it is not an isolated environment, so the server must also handle the burden of the DB. Additionally, you can't authorize users, only assign basic access rights based on the operating system. We decided to use SQLite because it is supported by Django and our application doesn't have a complex database structure or require various features of a database management system.

11. AWS EC2 [18]
    Amazon Elastic Compute Cloud (Amazon EC2) offers scalable computing capacity on-demand in the Amazon Web Services (AWS) cloud. By using Amazon EC2, you can accelerate application development and deployment while reducing hardware costs. Instead of purchasing physical servers, our team has decided to utilize a cloud system for its economic benefits and flexibility for future expansion. With Amazon EC2, you can deploy multiple virtual servers, configure security and networking, and manage storage.

It allows you to increase capacity (scale up) to handle compute-intensive tasks such as monthly or annual processes or spikes in website traffic. Conversely, when usage decreases, you can decrease capacity (scale down). One of the great advantages of Amazon EC2 is that it is free for the first year. Since our application needs to be built urgently and will not run for more than a year, Amazon EC2 is an excellent choice.

12. Jupyter NoteBook [19]
    Jupyter Notebook is a popular open-source web application that provides an interactive environment for writing code, visualizing data, and documenting. It supports various programming languages, including Python, R, and Julia. Jupyter Notebook is organized into code cells and markdown cells. In code cells, you can write and run Python code, while in markdown cells, you can write documents, formulas, or insert images. This combination of code and documentation is especially useful for projects like data analytics and machine learning. One of the advantages of Jupyter Notebook is the ability to run Python code line by line and view the results, which is helpful for debugging and testing. The output is immediately displayed below the code, allowing you to see visualized graphs, tables, and more. Additionally, Jupyter Notebook can be easily used online. Google Colab, provided by Google, is a cloud-based Jupyter Notebook environment. For our project, we will utilize Jupyter Notebook to run machine learning tasks using Python. Machine learning often involves writing lengthy code, and it is crucial to check and debug the results along the way. Jupyter Notebook is the ideal tool for this purpose.

13. Tensorflow / Tensorflow light [20]
    TensorFlow is a library created by Google that provides features for implementing deep learning programs. It is primarily implemented in C++, but supports multiple languages including Python, Java, and Go. However, Python is prioritized as it is most suitable for our application. Machine learning is a complex field, but with machine learning frameworks like TensorFlow, implementing a machine learning model has become more accessible and less complicated. These frameworks simplify tasks such as data acquisition, model training, making predictions, and refining results. Additionally, TensorFlow Lite is a mobile library designed for deploying models on mobile devices, microcontrollers, and other edge devices. When running a machine learning or deep learning model through TensorFlow Lite, it produces a small file of less than 3MB. This allows us to use trained models on mobile without any capacity constraints. In our application, we will use TensorFlow Lite to ensure immediate model availability on mobile devices.

14. Material Design [21]
    UI/UX is crucial in modern mobile and web applications, and a unified UI can greatly enhance the user experience. This unified UI is achieved through the use of a design system. However, implementing a design system on your own can be challenging. Therefore, it is common practice to utilize publicly available design systems, with Google's Material Design being the most prominent one. Material Design is a design approach that combines the benefits of flat design with the use of shadows to create a sense of depth. It also provides a comprehensive design system that includes typography and primary-secondary color schemes. This design can be easily implemented in the Figma library and seamlessly integrated into code using react-native-paper.

15. Git [22]
    Git is an effective version control system that allows you to manage versions and incorporate changes and updates seamlessly. But before delving into Git, let's first discuss what a "version control system" is. It is a system that records changes to a file over time and enables you to retrieve that file later when needed. When working on a document, there are usually multiple revisions and updates from the initial draft to the final version. Along the way, we often rename the file to "final," "_final," "finalized," and so on, thus overwriting the previous versions. This can make it difficult to go back to a specific point in time and understand the changes that were made. However, with a version control system, this becomes possible. By using a version control system, you can manage multiple versions of the same information. This allows you to track changes over time and identify the individuals who made them. You can easily revert back to previous versions or the original version, and quickly identify the person responsible for any issues that may arise.

16. Cloudinary
    Cloudinary is a cloud-based media management platform that enables you to efficiently manage, optimize, and distribute images and videos. This service is particularly useful for effectively managing and optimizing the media assets used

in your web and mobile applications. While AWS's image server could have been an option, we considered the fact that our machine learning models run within Django. Using AWS's static server would have compromised server capacity and performance. Therefore, we made the decision to utilize Cloudinary as a completely new static server for images. Now, the AWS server only needs to store the image download addresses from Cloudinary in its database.

B. *Software in use*

1. Visual Studio Code

Fig. 10. Visual Studio Code

Visual Studio Code is a lightweight text editor developed by Microsoft, which is excellent for web development. Nowadays, it offers a wide variety of powerful plugins that make it as lightweight and powerful as other IDEs. Our team exclusively uses Visual Studio Code for all of our code development. For front-end development, there are plugins available for React-Native and for seamlessly transferring work from Figma to code. Furthermore, there are plugins for Eslint and Prettier that automatically apply code formatting when you save a file, greatly enhancing the overall developer experience. In addition to front-end development, Visual Studio Code is also used for backend and machine learning tasks. The Python and Jupyter Notebook plugins are highly regarded, and the debugging capabilities are excellent.

2. Xcode

Fig. 11. Xcode

XCode is an integrated development environment (IDE) developed by Apple. It is essential for developing an iOS application using React Native. While we write the code in Visual Studio Code, we need to compile it as an iOS application using XCode.

3. Github

Fig. 12. Github

Github is a program that supports projects using Git, acting as a remote control center for Git. It serves as a platform for version control and collaboration among developers, offering a cloud-managed version control system. Git and Github are often used interchangeably for collaboration in modern software development. However, the capabilities of Github go beyond that. Firstly, Github is the preferred platform for open-source software, providing access to the source code of various tools used by our team. Additionally, any issues or bugs with open libraries can typically be found on Github. Github also offers other collaborative features. Pull requests allow us to review work in different Git branches before merging them. Furthermore, Github actions simplify the process of implementing continuous integration and continuous deployment (CI/CD). In our team, we utilize Github actions to check for errors in TypeScript on the frontend and to instantly reflect code changes to AWS EC2 on the backend.

4. Figma

Fig. 13. Figma

Figma is an application for UI/UX design, available as a cloud-based SaaS program. One of its main advantages is its real-time nature, allowing users to see immediate modifications to the UI within Figma. This eliminates the traditional process of a planner or developer reviewing a designer's file, requesting changes, waiting for updates, and then reviewing and modifying it again. Now, anyone can receive instant confirmation and immediate feedback, regardless of their job title. Another great aspect of Figma is the extensive collection of plugins and communities available. Many people have created useful design components and shared

them with the Figma community. This allows users to leverage high-quality UIs without having to create them from scratch. Lastly, Figma offers the functionality to create prototypes. These prototypes can be used to demonstrate button interactions within Figma, providing developers and planners with a clearer understanding of the application flow.

5. Slack

Fig. 14. Slack

Slack is a communication tool that offers several advantages. One of the biggest advantages is the ability to create separate channels for different teams and topics. This helps to avoid crowded conversations and keep discussions organized. Additionally, within a channel, you can use the thread feature to have focused discussions on specific topics. Threads allow users to send messages within a message they've sent, creating a threaded conversation. Another powerful feature of Slack is huddles. Huddles enable instant online meetings with just a push of a button. During a huddle, users can collaborate by writing on a shared computer screen using their mouse, making it easier to work together. Lastly, Slack offers a variety of plugins. One notable plugin is the Github plugin, which allows users to automatically post messages to Slack using tags, such as code review requests.

6. Notion

Fig. 15. Notion

Notion is a web-based SaaS application that functions as a wiki. One of its advantages is the ability to create articles in the form of MD files and see real-time changes. Recently, with various updates, it has become a valuable tool for managing meeting minutes and projects.

7. Overleaf

Fig. 16. Overleaf

Overleaf is an online platform that facilitates collaborative writing and editing of LaTeX documents. It offers a user-friendly interface for creating scientific and technical documents such as research papers, reports, and thesis papers. With Overleaf, multiple team members can work on the same document simultaneously, enabling seamless collaboration and change tracking. Additionally, it provides built-in features for managing references, equations, tables, and figures, making it a preferred choice for researchers and academics. This document was written using Overleaf's IEEE specification.

8. Postman

Fig. 17. Postman

Postman is a SaaS (Software as a Service) tool that facilitates faster and easier API development. It is a platform that enables you to test, document, and share your developed APIs. It is particularly useful for testing RESTful APIs. When it comes to testing backend functionalities like authorization, headers, and caching, it can be challenging. However, Postman automates these tasks and allows for easy customization, making testing a breeze. Additionally, Postman allows you to share tested URLs, simplifying collaboration between front-end and back-end developers.

9. ChatGPT

Fig. 18. ChatGPT

ChatGPT is an AI-powered tool that enables real-time conversations with an AI. For GPT-3.5, it is trained on data up to 2021, while GPT-4 is trained on more recent data. ChatGPT has revolutionized generative AI,

offering enhanced capabilities for tasks such as report generation, article summarization, problem-solving, and even coding. In our team, we will utilize ChatGPT to simulate plant conditions, allowing the AI to engage with users as if it were a plant.

10. DBML & DBDocs

Fig. 19. DBML& DBDocs

DBML is a Domain Specific Language (DSL) for defining database structures. This text-based language allows you to explicitly describe tables, columns, indexes, foreign key relationships, and more. DBML is designed to create visual and intuitive database structures. It allows developers to define database schemas without having to write complex SQL syntax. dbdocs is a web-based tool that generates documentation based on database schemas written in DBML. dbdocs allows you to create beautifully documented websites directly from DBML files. This tool can help you visualize your database design, share it with team members, and help non-technical stakeholders understand the structure. When a DBML file is uploaded through the dbdocs.io service, then dbdocs processes it to provide detailed documentation of your database schema, including tables, relationships, indexes, and more. This document includes search capabilities, relationship diagrams, table details, and more, making it easy to navigate and understand your database structure.

11. Android Studio

Fig. 20. Android Studio

Android Studio is an integrated development environment (IDE) for Android, developed by Google. Similar to Xcode, Android Studio is essential software as it handles the final compilation of an Android application.

12. Flipper

Fig. 21. Flipper

In React Native, Flipper is a platform used for debugging and development purposes. It is an open-source tool developed by Facebook and offers various features for debugging and performance profiling of mobile applications. Flipper itself provides three types of debugging: checking the DOM tree structure, monitoring network communication, and inspecting image information. One of the powerful aspects of Flipper is its plugins. It offers a wide range of plugins that allow you to debug libraries essential for developing BloomMate, such as react-navigation, recoil, and react-query. You might wonder if React Native already provides its own Chrome debugging. However, some libraries, especially those related to react-native-reanimated, may not work with Chrome debugging enabled due to compatibility issues with UI Threads. In such cases, Flipper can still be used as it debugs at the JavaScript level.

C. Team's Development Environment

TABLE I
TEAM'S DEVELOPMENT ENVIRONMENT

Name	Environment
Kim Dong Hyun	MacOS Monterey 12.5 react-native 0.72.4 python 3.9.6
Shin Hyun Ah	Windows 11.22.2 react-native 0.72.4
Yoon Yong Sung	Windows 11.22.2 react-native 0.72.4
An Soonho	Windows 11.22.2 python 3.10.7

D. Cost Estimation

To create BloomMate, we take advantage of a diverse range of cost-effective programs that are available for free. This helps us minimize the expenses related to the development process. However, it's important to mention that using the ChatGPT API does come with a cost. Nevertheless, there is a provision that allows you to enjoy a free quota of approximately $18, which means you can ask around 300,000 questions without any charges. Therefore, you can be confident that the expenses associated with running the

application will be relatively low.

E. Task Distribution

TABLE II
TASK DISTRIBUTION

Tasks	Name	Descriptions
Frontend Developer	Shin Hyunah, Yoon Yongsung, Kim Donghyun	Front-end developers utilize languages like React Native and TypeScript to create applications. They are responsible for designing the interfaces that users interact with, such as tapping buttons and swiping through screens. Their primary objective is to create a user experience that is both accessible and engaging, while adhering to the specified design. Additionally, front-end developers are responsible for transferring user-entered information to the backend developers. The reason for having three front-end developers is that two of them write code for each screen, while the third developer reviews and optimizes the code for the screens that users see. This organizational structure requires effective teamwork, clear role allocation, excellent communication skills, and collaborative synergy.
Backend Developer	An Soonho	Backend developers are responsible for designing the database and application architecture, as well as writing the APIs used by frontend developers. When working with APIs, backend developers need to be able to receive information from application users through the frontend and provide the correct return value to the API. They also need to design APIs that interact with the backend to leverage generative AI and machine learning features, and make them accessible to application users. This role requires a strong understanding of the central database and software structure, and ensuring that software development aligns with that structure.
UI-UX Designer	Yoon Yongsung	The UI-UX designer, using Figma, is responsible for determining how the application screens are presented to users. This role involves deciding which screens will be more engaging and comfortable for users to use. As a UI-UX designer, the goal is to create a design that keeps users engaged and encourages them to return to the application. Once the UI-UX decisions are finalized, they can be communicated to the front-end developers. It is important for the UI-UX decisions to not only look good, but also consider the needs of both the application user and the front-end developer who will be building the screens. The design should also allow the back-end developer to design the database. This role relies on effective collaboration with developers.
Product Designer	Shin Hyunah	A product designer is someone who designs products with a user-centered perspective. They must have the ability to put themselves in the shoes of the user, find the pain points within the service as it currently exists, explore what they would like to see added, and come up with ideas. They are responsible for creating the overall framework of the product or service. Product designers are also responsible for communicating with the rest of the team to find out how they can take their ideas further and finalize their direction. Product designers should focus on the user usability of the product or service, exploring features that users really need and modifying the design of the product.
AI Developer	Kim Donghyun	A machine learning software developer works with algorithms, data, and artificial intelligence. Their role involves researching, building, and designing artificial intelligence software specifically for machine learning purposes. They primarily focus on applying artificial intelligence systems to various applications. The responsibilities of this role include collecting, cleaning, and preprocessing data to extract meaningful value. They then use this data to train models and deploy them in software. Additionally, the machine learning software developer must appropriately implement machine learning algorithms into software functions, conduct experiments and tests of AI systems, and determine the most suitable models for the application's functions. They are also responsible for designing and developing machine learning systems, as well as performing statistical analysis.

V. SPECIFICATIONS

A. Landing Page

The Landing Page is the first screen that users see when they open the application. It prominently displays the BloomMate logo and a condensed representation of the application's identity. Additionally, it includes two buttons that allow users to navigate to the SignUp Page and the Login Page. It is important that the design of each button aligns with our established design system and clearly indicates their respective functions.

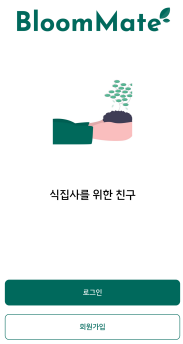


Fig. 22. Landing Page

B. SignUp Page

Users can begin using BloomMate via the SignUp Page. This is a crucial step as BloomMate offers personalized plant management and chat services. During the signup process, users are required to fill in six fields: name, ID, password, Tiiun product key, garden size, and address. Each field is presented on a separate page, and none can be skipped.

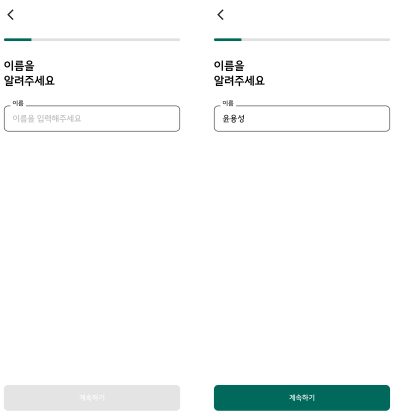


Fig. 23. Sign Up Page: Example of Next button enabled

An arrow icon in the top left corner of each page indicates ”back.” Clicking this icon will either return you to the Landing Page (if you’re on the first step, entering your name), or to the previous step. A Call-To-Action (CTA) Button is located at the bottom of each page. This button only activates once all fields are correctly filled in. The button will read ”Sign up” on the last step and ”Continue” on all others. Upon pressing the activated Register button on the final step, a Register Post API request is sent to the server. If received successfully, the user is directed to the Login Page.

The constraints and peculiarities for each input item are as follows: (1) The name should be between 2 to 5 characters. Input is accepted directly from the user via the keyboard. (2) The username should contain 5 to 20 characters, allowing only English letters and numbers. Input is accepted directly from the user via the keyboard. (3) The password should contain 8 to 20 characters, allowing only English letters and numbers. A password re-entry is required for confirmation. Input is accepted directly from the user via the keyboard. (4) The Tiiun product key should begin with ’tiuun’ and be exactly 8 characters long. Input is accepted directly from the user via the keyboard. (5) For the garden size, use the toggle button to select either Large, Medium, or Small. Selecting another size will deselect the current one. Each garden size has a plant limit indicated by the number of pots in the image - Large has 7, Medium has 5, and Small has 3 pots. (6) For the address, we use the address input API provided by Kakao. It only provides the street address and displays the smart cottage photo when the address is entered.

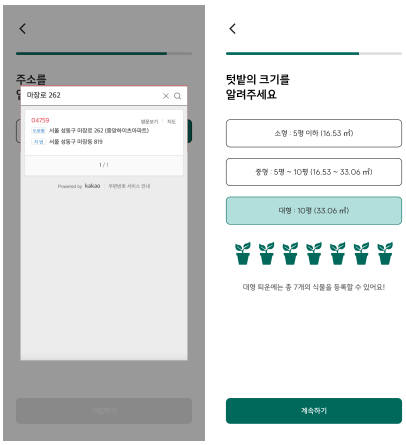


Fig. 24. Sign Up Page: Input Field Requirements

C. Login Page

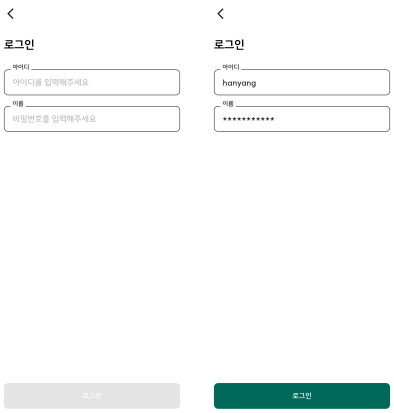


Fig. 25. Login Page

The Login Page of BloomMate allows users to personalize their experiences and utilize core features. Login is essential because it enables the server to identify the user making the request. The login process requires a username and password, each on a separate page, and both must be filled in. The

Login button at the bottom is initially disabled, but it becomes enabled when the username and password are correctly entered without errors. If you press the enabled login button, a login POST API request is sent to the server. If successful, the server provides a token, stored via the Set-Cookie command in the header, which identifies the user. You are then directed to the Plant List Page. On the top left of this page, there's an arrow icon pointing left. Pressing this icon returns you to the Landing Page.

The constraints and peculiarities for each input item are as follows: (1) For the ID, you are allowed to enter a minimum of 5 and a maximum of 20 characters, using only English letters and numbers. User input is accepted directly through the TextInput via the keyboard. (2) For the password, you must enter between 8 and 20 characters, again using only English letters and numbers. User input is similarly accepted directly through the TextInput via the keyboard.

D. Plant List Page

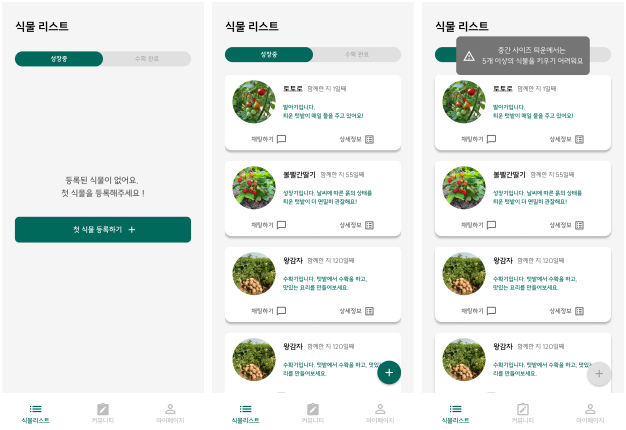


Fig. 26. Plant List Page: Growing Tab

The Plant List Page, which appears immediately after a user logs in, displays a list of plants currently growing in Tiiun on SmartCottage or those previously grown and successfully harvested. The plant list is retrieved from the server via a GET request. This page, found on the first tab, is one of three screens where the bottom tab bar is visible. The icon on the first tab should be in the primary color, with the other two icons in the disabled color. The page should be titled "Plant List" at the top. Below that, there are two top tabs: 'Growing' and 'Harvested'.

For the "Growing" Tab: (1) If no plants are currently growing, an 'Add Plant' button should be displayed in the center of the screen, accompanied by a text prompt to add a plant. Clicking the 'Add Plant' button will navigate to the 'Plant Add Page'. (2) If plants are growing, the screen should display a scrollable list of plants, each presented as a card. These cards contain six elements: a plant picture, a nickname, the number of days spent with the plant, a growth stage guide, a 'Chat with Plant' button, and a 'Plant Details' button. Clicking on a plant card navigates to the 'Plant Detail Page'. The 'Chat with Plant' button opens the chat feature, while the 'Plant Details' button leads to the 'Plant Detail Page'. A floating button at the bottom of the scroll directs to the 'Plant Add Page' when clicked. (3) If the maximum number of plants has been reached, a message should appear stating that no more plants can be added. This message may be overlooked if the list of plants is full, requiring the user to scroll to see the bottom. If the 'Add Plant' floating button is clicked in this case, the button should change to a disabled color and a toast message should display, indicating that adding more plants is not permitted.

Fig. 27. Plant List Page: Completed Tab

For the "Completed" Tab: (1) If you don't currently have any harvested plants, write a text in the center of the screen stating that you don't have any harvested plants and encouraging you to grow your plants to harvest them. (2) If you do have any harvested plants, the body of the screen should be a scrolling page. Within the scrolling screen, show a list of harvested plants in card format, one by one. Nothing happens when the plant card button is pressed. Inside the plant card, there are three pieces of information: a picture of the plant, the plant's nickname, and the date it was grown.

E. Community Page

The Community Page is a resource for BloomMate users. It provides answers to questions, solutions to problems, and necessary information. The aim is to reduce user bounce rates and enhance farming experiences with accurate information from smartCottage. Plant list items are fetched from the server using GET requests. The bottom tab bar, one of three screens displayed, is located on the first tab. Icons on the second tab should be in the primary color, while the remaining two icons should be in the disabled color. The Community Page should feature the title 'Plant Buttler Community' at the top. Directly underneath, there should be two main tabs: 'Questions and Answers' and 'Expert Articles'.

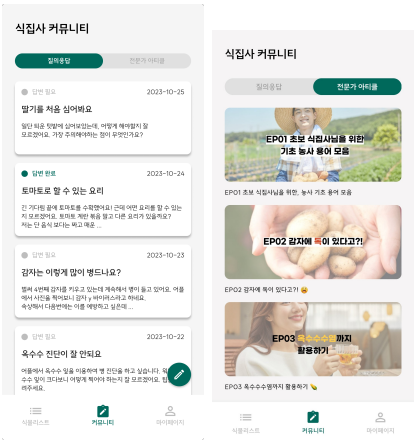


Fig. 28. Community Page

For the "Questions and Answers" Tab: The main part of the Q&A screen is a scrolling page, with questions displayed in a card format, sorted by the most recently asked. Tapping on a question card directs you to the Question Detail Page. Each question card includes four details: answer completeness, question date, question title, and question body. The question body has an 80-character limit; if it exceeds this, it's truncated with ellipses. At the bottom of the scroll screen, there's a floating button. Pressing this button navigates to the Add Question page.

For the "Expert Articles" Tab: The article screen should be a scrollable page where articles are displayed in a card format. Each card contains two pieces of information: a thumbnail image and a title. Tapping on an article will allow you to view it in WebView.

F. Question Add Page

Fig. 29. Question Add Page

The Question Add Page is where users can post their inquiries. Users can ask about their BloomMate experience, report issues, or pose questions about their plants. Each question includes a title and content, both provided by the user as TextInput. However, the content is entered in a multiline TextInput. Each TextInput features an appropriate placeholder. The Ask Question button is located at the bottom. Initially disabled, it becomes enabled when both title and content contain at least one character. An arrow icon on the top left of the page serves as a back button, leading back to the Community Page.

G. Question Detail Page

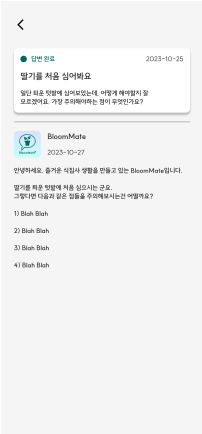


Fig. 30. Question Detail Page

The Question Detail Page is where you can see the details of your question. You'll find professional answers from the team behind BloomMate. First, the question is formatted like a question card in the question list. If the answer is complete, it will be shown along with the date of the answer. If the answer is long, it might be a scrolling screen. In the top left corner of the page, you'll see an arrow icon that indicates left. Pressing the arrow icon will take you back to the Community Page.

H. MyPage

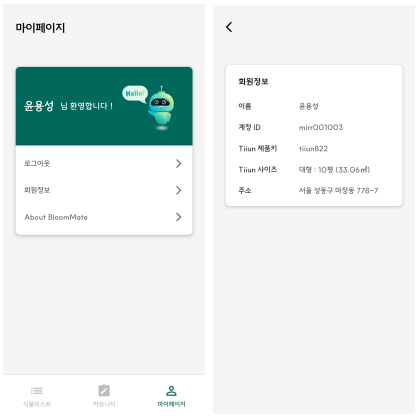


Fig. 31. MyPage

MyPage, a feature of BloomMate, provides information tailored to the user. It is found on the third tab of the bottom

bar, which is visible on three screens. The icon on the second tab should be in the primary color, while the other two icons should be in the disabled color. At the top, MyPage should be titled 'MyPage', followed by a welcome message in the form '*Welcome userName*'. MyPage offers three functions: Logout, View Membership, and About BloomMate.

The Logout function, when clicked, deletes the user's cookie information and redirects the user to the landing page.

The View Membership function, upon selection, navigates the user to a new screen where they can review their membership details. This screen displays five types of information: name, ID, activated product key, garden size, and address.

Lastly, the About BloomMate function directs the user to a Notion webview that provides an introduction to BloomMate.

I. Plant Add Page

BloomMate is an app designed to assist you in growing plants in Tiiun. To use it, you need to register your plants on the Plant Add Page. The plant addition process requires input in four fields: photo, variety, nickname, and planting date. Each field is located on a separate page, with the photo field split across two additional pages. All fields are mandatory and cannot be skipped. All pages feature an arrow icon in the top left corner, indicating a return function. Clicking this icon will either take you back to the Plant List Page if you're at the first step, or to the previous step. Each page also includes a Call-To-Action (CTA) button at the bottom. This button will only be activated once all fields on the page are filled out correctly. The button text reads "Add Plant" on the final step and "Continue" on all other pages. On the final step, pressing the activated Register button will trigger a server request to the Add Plant Post API. Upon successful receipt, you will be redirected to the Login Page.

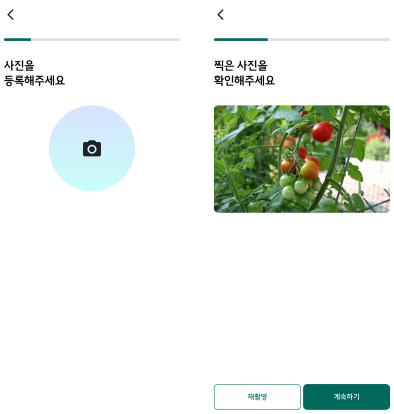


Fig. 32. Plant Add Page: add Picture

Now let's explain each input: (1) For photos, once the user takes a photo, it is instantly registered with the photo server using a POST request. The URL received from the photo server is then used in the final POST request. (2) Regarding plant varieties, users can choose one of the four options provided in the bottom-sheet via a toggle button. The options are strawberry, tomato, potato, and corn. These varieties are used in the AI model. They were chosen for their suitability in home gardens due to their reasonable growth heights. The toggle button allows you to switch between varieties. If another variety is selected while one is already active, the first one will be deselected. (3) For nicknames, the user should enter a minimum of 1 character and a maximum of 5 characters. TextInput allows direct input from the user through the keyboard. (4) To choose the planting date, users can select a date from the calendar provided in the bottom-sheet. Dates later than the present are not selectable. The selected and pressed states should be clearly marked.

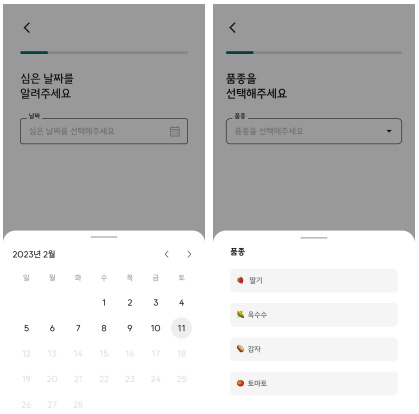


Fig. 33. Plant Add Page: Fields

J. Plant Detail Page

Fig. 34. Plant Detail Page

Users want to access information about the plants they are growing. To fulfill this need, they should be able to view plant details on the Plant Detail Page. All plant details are obtained from the server through a GET request. Plant information can be categorized into two types: details and growth information. Details remain the same for each plant variety. The Plant Detail Page displays eight types of details: variety, temperature, humidity, light level, flowering time, watering frequency, difficulty, and precautions. The growth information indicates the current stage of the plant based on the planting date: germination, growing, or harvesting. We have included a graph to provide a quick overview of the growth information. At the bottom of the page, there are buttons for different scenarios: a 'Harvest' button during the harvest season and a 'Diagnose' button for other situations. Clicking the Harvest button triggers a PATCH request to the server and redirects you to the Plant List Page. On the Plant List Page, the plant is no longer considered a growing plant as it has been harvested. Alternatively, you can click the Diagnose button, which will take you to a page where you can diagnose the plant. In the top-right corner, there is a button that navigates to the Plant Edit Page. The top-left corner of the page contains a left arrow icon. Clicking the arrow icon will return you to the Plant List Page.

K. Plant Edit Page

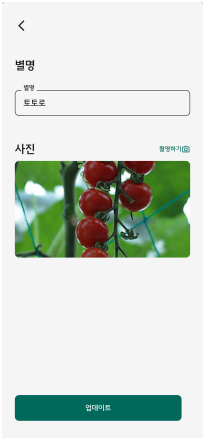


Fig. 35. Plant Edit Page

Users may want to update information about their plants, especially if they're growing them and need to take pictures of them frequently as they grow. The Plant Edit Page is designed for this purpose. There are two things that can be updated: a nickname and a photo. The nickname is accepted as a TextInput, and if it is empty, an error will be thrown. The photo can be updated by taking a picture. At the bottom is the Update button, which is activated if at least one item exists that is different from the existing one and there are no errors. If you press an active update button, it will send a PATCH request to the server, and if it is successful, you will be taken to the Plant Detail Page. In the top left corner of the page, you will see an arrow icon indicating left. Pressing the arrow icon will take you back to the Plant Detail Page.

L. Plant Diagnosis

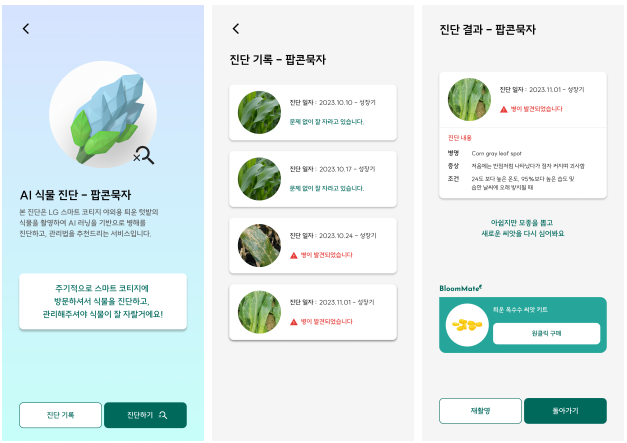


Fig. 36. Plant Diagnosis

One of BloomMate's main features is AI plant diagnostics, which serves two purposes. Firstly, it helps users, especially those with limited plant-growing experience, determine if their plants have any diseases by diagnosing their health. The AI diagnosis simplifies the decision-making process. Secondly, by encouraging users to regularly diagnose their plants when visiting SmartCottage, it promotes a sense of care and nurtures a loving relationship between users and their plants.

The plant diagnosis process begins on the information page, where users can learn about the benefits of using AI to diagnose their plants. It also reminds them to periodically visit SmartCottage to monitor their plants. From the information page, users can navigate to the Make a Diagnosis page and the Diagnosis History page.

The Make a Diagnosis page functions as a camera app. Users take a picture of their plant using the app, and the diagnosis is performed within their phone using the TensorFlow Lite model. After the diagnosis is complete, users are directed to the diagnosis results page. The diagnosis result will indicate whether the plant is (1) sick or (2) healthy. For each diagnosis, the following information is provided: the photo taken, the date of diagnosis, and the plant's growth stage (germination/growing). If the plant is diagnosed as sick, users will receive guidance on the disease, symptoms, and conditions. Unfortunately, if the plant is diseased, it will need to be replanted. Additionally, the generative AI will automatically provide a link to purchase customized Tiiun seeds that are most suitable for the user's current plant. By tapping on the link, users can make the purchase, and a confirmation message will be displayed. At the bottom of the page, there are options to reshoot the picture or return to the diagnosis guide page.

The Diagnostic History page displays a scrolling screen where users can view their previous diagnosis records. Each

Fig. 37. Plant Diagnosis - One Click Purchase

diagnosis history is presented in a card format, showing the photo taken, the date of diagnosis, and the plant's growth stage. By tapping on a diagnosis history card, users can view the diagnosis log page in the same format as the diagnosis results page.

M. Chat With Plant

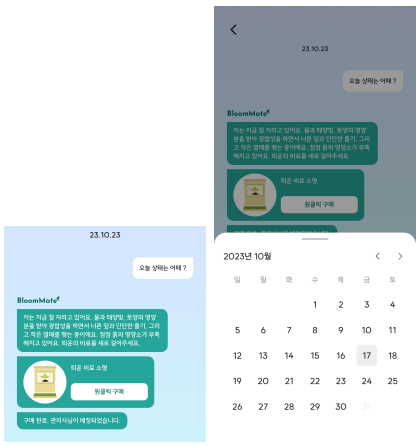


Fig. 38. Chat With Plant

The chat feature is the second core component of BloomMate. Each day, when a user requests to start a chat, the server synthesizes all available information and generates a script to send to the generative AI. This script essentially prompts the generative AI to assume the role of a plant. The server utilizes six types of information to create the script: plant details, plant nickname, plant planting date, today's weather, Tiiun soil information, and diagnostic information. Today's weather is obtained from the weather API based on the address provided during the signup process. As for Tiiun soil information, it is currently not accessible, so we will randomly generate a good or bad status. Regarding diagnostic information, we only retrieve the date of the last diagnosis, not the actual content. We assume that this date represents the last time the user visited SmartCottage. If it has been more than a week, we will include a situation in the script where the plant expresses a desire to see the user.

Let's recap the chat flow with the plant:

1. The user clicks the "Start today's chat" button.
2. The server generates a script to obtain today's report from the generative AI and sends it to the client.
3. The client checks the report and, if the soil condition is bad, generates a link for fertilizer recommendations from the generative AI.
4. The user is then free to chat with the generative AI, which has been trained with scripts.

Additional specifications:

1. The above flow applies when there are no ongoing chats for the day. If there is already an active chat, users can continue without the need for the "Start" button.
2. Today's chat ends at midnight every day.
3. There is a floating button located at the bottom left of the screen. Tapping on the previous report will open a bottom-sheet with a calendar. By selecting a past date from the calendar, users can view the chat history for that specific day.

VI. ARCHITECTURE DESIGN & IMPLEMENTATION

A. Overall Architecture

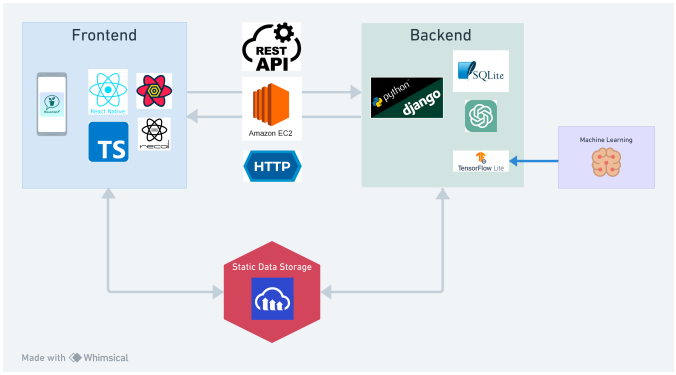


Fig. 39. Application Architecture

BloomMate consists of four main modules: Frontend, Backend, Static Data Storage, and Machine Learning. Let's take a closer look at each module.

The Frontend module refers to the part of the mobile application that is used by the end user. We prioritize meticulous code writing, good UI/UX design, and bug-free functionality in the frontend. BloomMate's frontend is built using react-native with Typescript. We utilize react-query for server status management and recoil for internal global status management. Additionally, we use react-native-calendar for the calendar feature and react-hook-form for input form management. With the BloomMate mobile application, users can perform various functions such as adding plants, checking plant information, diagnosing plants, chatting with plants, accessing expert articles, and asking questions.

The Static Data Storage module is accessible from both the frontend and backend. This module serves as an externalization of certain database functions, specifically for storing and loading static data, images, videos, etc. BloomMate utilizes cloudinary for this purpose. Despite being a free service, cloudinary offers a generous 25GB

of storage and simplifies the process of uploading and downloading images. Here's how it works: the frontend uploads a photo to cloudinary and receives a downloadable URL as a response. The frontend then passes this URL to the backend, which stores it in its own SQLite database. When the frontend receives a request using the URL, it fetches the URL from the database and retrieves the corresponding image from cloudinary. In BloomMate, the frontend uploads images in two cases: when adding a plant and when diagnosing a plant.

Please note that the third module, which will be discussed in more detail, covers the process of storing and loading static data in the backend's SQLite database.

The third module is the Backend module, which serves as the server handling user requests. What sets BloomMate apart is that the database is built within the Backend module itself. This is made possible by using Django, a technology that includes its own DB called SQLite. Looking at the SQLite part, depicted in the ERD above, it consists of 9 databases. Among these, the article table is an independent table not connected to any other table. Within Django, there is code that also interacts with chatGPT. We provide chatGPT with information about the plant and the user, assigning the plant's role. It then sends the phrases you write to chatGPT, retrieves the answers, and sends them back to you. The TensorFlow Lite model created in the fourth module is also stored in the backend, along with the necessary code to execute it. After the user takes a photo, the photo URL is stored in the second module, cloudinary, and this URL is used for diagnosis.

The fourth module is the Machine Learning module. Its purpose is to create a model capable of image classification using TensorFlow's ResNet50. When designing BloomMate, our team focused on whether or not the plant's owner is present in the SmartCottage. For each scenario, the core functionality is as follows: if the owner is not in the SmartCottage, generative AI is used to interact with the plant; if the owner is in the SmartCottage, a picture of the plant is taken and AI is used to diagnose it. This Machine Learning Repository is intended for the AI used in the second scenario. It involves training the AI using a Custom Dataset and converting it to TensorFlow Lite so that it can be utilized in real projects.

B. Directory Organization

TABLE III
DIRECTORY ORGANIZATION - FRONTEND 1

Directory	***File / Folder Name***	***Library***
BloomMate/ BloomMate-FE	android ios node_modules patches src .eslintrc.js .gitignore .prettierrc.js app.json babel.config.js index.js package.json react-native.config.js tsconfig.json	react react-native patch-package postinstall-postinstall
BloomMate-FE /patches	react-native-action-button+2.8.5.patch react-native-paper+5.10.4.patch	patch-packages
BloomMate-FE /src/atoms	button.atom.tsx divider.atom.tsx icon.atom.tsx image.atom.tsx modal.atom.tsx point-linear-gradient.atom.tsx skeleton.atom.tsx suspender.atom.tsx text.atom.tsx text.util.ts text-input.atom.tsx toast.atom.tsx	react react-native @mobily/stacks color lodash react-native-paper react-native-vector.icons react-native-fast-image react-native-modal react-native-linear-gradient react-native-shimmer-placeholder react-native-toast-message
BloomMate-FE /src/assets	img.asset.ts lottie.asset.ts	lottie-react-native

TABLE IV
DIRECTORY ORGANIZATION - FRONTEND 2

Directory	*File / Folder Name*	*Library*
BloomMate-FE /src/screen	root.navigator.tsx about-bloom-mate.screen.tsx article-webview.screen.tsx community-qna-detail.screen.tsx community-qna-post.screen.tsx landing.screen.tsx login.screen.tsx plant-add.screen.tsx plant-cht.screen.tsx plant-detail.screen.tsx plant-detail-edit.screen.tsx plant-diagonsis-intro.screen.tsx plant-diangosis-list.screen.tsx plant-diagnosis-log.screen.tsx plant-diagnosis-result.screen.tsx primary/ primary-community.screen.tsx primary-my-page.screen.tsx primary-plant-list.screen.tsx primary-plant-harvesed-list.screen.tsx primary-plant-current-list.screen.tsx primary-community-article.screen.tsx primary-community-qna.screen.tsx signup.screen.tsx user-info.screen.tsx	@actbase/react-daum-postcode @gorhom/bottom-sheet @hookform/resolvers @mobily/stacks @react-native-community/hooks @react-navigation/bottom-tabs @react-navigation/material-top-tabs @react-navigation/native @react-navigation/stack axios color dayjs eslint-config-prettier eslint-plugin-import eslint-plugin-prettier eslint-plugin-react eslint-plugin-unused-imports lodash lottie-react-native moti react react-hook-form react-native react-native-action-button react-native-calendars react-native-error-boundary react-native-gesture-handler react-native-image-picker react-native-linear-gradient react-native-mmkv react-native-pager-view react-native-paper react-native-safe-area-context react-native-screens react-native-shimmer-placeholder react-native-tab-view react-native-toast-message react-native-webview react-query recoil rooks yup

TABLE V
DIRECTORY ORGANIZATION - FRONTEND 3

Directory	*File / Folder Name*	*Library*
BloomMate-FE /src/providers	mutation-indicator query-client recoil ui	react react-native @mobily/stacks react-native-flipper react-query-native-devtools react-native-reanimated react-native-mmkv-flipper-plugin
BloomMate-FE /src/layouts	basic-layout CTA-section dialog loading-pae modal-header scroll-view	react react-native @mobily/stacks react-native-keyboard-aware-scroll-view @react-native-community/hooks
BloomMate-FE /src/hooks	get-account-info-query.hook.ts get-article-list-query.hook.ts get-plant-chatting-query.hook.ts get-plant-detail-query.hook.ts get-plant-diagnosis-record-detail-query.hook.ts get-plant-diagnosis-record-list-query.hook.ts get-plant-list-query.hook.ts get-question-detail-query.hook.ts upload-image-library.hook.ts upload-photo.hook.ts	react react-query react-native-image-picker axios

TABLE VI
DIRECTORY ORGANIZATION - BACKEND 1

Directory	*File / Folder Name*	*Library*
BloomMate	db.sqlite3 manage.py requirements.txt test2.jpg	
BloomMate /BloomMate_ backend	__init__.py asgi.py settings.py urls.py wsgi.py	os django.core.asgi pathlib datetime dotenv django.contrib django.urls

TABLE VII
DIRECTORY ORGANIZATION - BACKEND 2

Directory	*File / Folder Name*	*Library*
BloomMate /accounts	__init__.py admin.py apps.py models.py serializer.py tests.py urls.py views.py	django.contrib django.apps django.db django.utils rest_framework django.urls rest_framework_simplejwt
BloomMate /articles	__init__.py admin.py apps.py models.py serializer.py tests.py urls.py views.py	django.contrib django.apps django.db rest_framework django.urls rest_framework_simplejwt
BloomMate /chatting	__init__.py admin.py apps.py models.py serializer.py tests.py urls.py utils.py views.py	django.contrib django.apps django.db django.utils rest_framework django.urls rest_framework_simplejwt openai requests random django.utils geopy.geocoders django.conf django.shortcuts django.utils
BloomMate /community	__init__.py admin.py apps.py models.py serializer.py tests.py urls.py utils.py views.py pagination.py permissions.py	django.contrib django.apps django.db django.urls rest_framework django.https rest_framework_simplejwt
BloomMate /plants	__init__.py admin.py apps.py models.py serializer.py tests.py urls.py views.py model.tflite	django.contrib django.utils django.apps django.db rest_framework django.urls tensorflow numpy os keras

TABLE VIII
DIRECTORY ORGANIZATION - MACHINE LEARNING

Directory	*File / Folder Name*	*Library*
BloomMate /BloomMate-ML	.vscode dataset tensorflow-lite-models test-case .gitignore model.h5 model.tflte README.md resnet-tflite-test.ipynb resnet-all-datasets.ipynb training-result.png	tensorflow numpy keras pathlib matplotlib
BloomMate-ML /dataset	Corn Common rust Corn Gray leaf spot Corn Healthy Corn Northern leaf blight Potato Early blight Potato healthy Potato Late blight Strawberry healthy Strawberry leaf scorch Tomato Early blight Tomato healthy Tomato Late blight Tomato Target Spot Tomato Yellow Leaf Curl Virus	

C. Module1 - Frontend

1) Purpose
We chose to use TypeScript-based React Native for the development of BloomMate. Let's first discuss the reasons behind this decision. React Native is a cross-platform framework developed by Facebook that allows for simultaneous development of iOS and Android applications. One of our team members had prior experience with React Native at a previous company, and the rest of the team had JavaScript development experience, making it a logical choice for us. The decision to adopt TypeScript was motivated by the need for more stable development. JavaScript, being dynamically typed, has a higher probability of unexpected bugs at runtime. This becomes particularly problematic when working with react-navigation, an essential component for React Native, as it becomes unclear what parameters and navigation the screen inherits when TypeScript is not used. To mitigate these drawbacks, we developed BloomMate using React Native with TypeScript. In the frontend, we perform the following key functions: (1) Receiving input from users (2) Passing the input to the server (3) Fetching data from the server.

2) Functionality
All features available on BloomMate can be accessed on the frontend. To get started, simply sign up and log in by providing the necessary information (all user

inputs are handled with error handling implemented). Once logged in, you can create your own nickname and attach a photo to add a plant. From there, you can freely interact with your plant and even diagnose its condition by examining its leaves if it appears to be in poor health. Additionally, you have the opportunity to ask questions to experts and read high-quality articles for further knowledge and guidance.

3) Location of source code:
https://github.com/BloomMate/BloomMate-FE

4) Class Component
   a) BloomMate/BloomMate-FE
   When you first enter the Frontend Repository, you will come across many files that were not created by the developers themselves. These files are mostly automatically generated during the installation of react-native. They primarily consist of configuration files and files optimized for the Android and iOS environments. However, this does not mean that the BloomMate developers have not made any modifications. Let's proceed to explain each file or folder, providing a detailed explanation of the various class components within the folders.

   - android [folder] :
   The android folder is crucial for ensuring the smooth functioning of a react-native project on Android. Developers typically modify the files inside this folder for the following reasons:
     i) Making changes related to the application's build, such as modifying the build version, name, or permissions.
     ii) In earlier versions of the react-native library, modifications to the native part of Android were required. However, this is becoming less common now, as such modifications are either handled in JavaScript or not required at all.
     iii) Inserting static elements such as fonts and icons.

   Therefore, it is common to modify files like android/build.gradle, android/app/build.gradle, and AndroidManifest.xml for these purposes.

   - ios [folder]:
   ios folders, like android folders, ensure that react-native projects run smoothly on iOS. The purpose of modifying the ios folder is similar to that of the android folder, but there is a difference. This difference is related to the Podfile. In an iOS project that uses CocoaPods, the Podfile is a configuration file used to define external libraries and dependencies to be added to the project. It allows CocoaPods to install and manage the required libraries for the project. Most react-native libraries require modifications to the Podfile. At BloomMate, we wanted to simplify the process for developers by avoiding the need to enter commands manually each time. Instead, we have included the command "npx pod install" in the post-install section of the package.json file. This command automatically updates the Podfile after downloading new libraries.

   - package.json & node_modules [folder] :
   The package.json file manages information about the current project and its dependencies, which are modules installed through the package manager (npm, yarn). It is formatted in JSON and contains the following information:
     i) Basic information section: This includes the project name and version.
     ii) Script section: This defines various commands that can be used in the project, such as building, running, testing, or automating tasks. In BloomMate, additional commands are included, such as type-checking for TypeScript errors, postinstall-related automatic command settings, and automatic cache value deletion when starting react-native.
     iii) Dependencies and devDependencies section: These manage the versions and dependencies of the libraries installed in BloomMate. The library versions are managed using semantic versioning. Dependencies are packages needed in the production environment, while devDependencies are packages needed only during development.

   All installed libraries are stored in the node_modules folder. To use a library anywhere in the project, you will refer to the installed library in the node_modules folder.

   - patches [folder]:
   This folder is where the patch-package library operates. By using patch-package, you can make custom changes to the libraries within the node_modules, and these modifications will persist even when the changes are deployed. In other words, the modifications made to the node_modules are managed by Git and applied in any execution environment. This allows for safely sharing modifications when quick fixes are needed for library bugs or when additional functionality is desired that is not supported by the library. The libraries that have been patched and the reasons for patching them will

be discussed later.

- src [folder] :
  The src folder stands for ”source” and is where all the code written by the frontend developer of BloomMate is stored. If files are managed at the top level instead of within the src folder, it has the disadvantage of disrupting the hierarchical order of the folder structure. Therefore, it is common practice to create a single folder and manage everything inside it, and BloomMate has chosen src as that folder. The various folders within the src folder will be examined in detail later in this document.

- .eslintrc.js & .prettierrc.js :
  ESLint and Prettier are primarily used in JavaScript codebases to enhance code quality and ensure consistent style. ESLint detects syntax and common errors, providing developers with guidelines through static analysis. On the other hand, Prettier focuses on code formatting, ensuring consistent style and improving code readability. When used together, these tools facilitate efficient collaboration and maintain consistent code quality. The two files in question configure ESLint and Prettier, respectively. In particular, BloomMate installs additional plugins for ESLint to manage unused import statements and the order of import statements.

- app.json & react-native.config.js :
  These two files configure React Native itself. They manage the displayName of the app and the font location. This allows you to install the font in the appropriate locations for both Android and iOS using commands.

- babel.config.js :
  The babel.config.js file is used to specify Babel settings in a React Native project. Babel is a tool that transforms JavaScript code, primarily converting it to be compatible with browsers or environments that do not support the latest ECMAScript standards. It also converts JSX code into React.createElement function calls, enabling it to be understood at runtime. In BloomMate, we utilize the default Babel preset and include the module-resolver plugin to define module aliases. This allows us to import modules using aliases instead of using relative paths.

- index.js :
  The index.js file is one of the entry point files in a React Native project. A React Native app is composed of JavaScript code, and the index.js file is the first file executed when the app starts. Its main responsibility is the initial setup and rendering of components. Typically, in this file, the main component of the app is registered and rendered using AppRegistry. As the starting point of a React Native app, you can add necessary initial configurations or logic here and, if needed, modularize them into different files for better management.

- tsconfig.json :
  The tsconfig.json file is used to configure the behavior of the TypeScript compiler in a TypeScript project. This file provides information to the compiler about the project’s configuration, compilation options, path settings, and other specific configurations. In BloomMate, TypeScript plays a crucial role, and tsconfig.json is an important file for configuring it. It is not enough to define aliases for relative paths only in babel.config.js; they must also be defined in tsconfig.json as TypeScript cannot recognize them otherwise. Additionally, to use files like lottie and png as modules, their file extensions should be added as types in tsconfig.json.

b) BloomMate-FE/patches
   I will introduce a special folder called the ”patches” folder in the root directory. This folder will be encountered before we reach the ”src” folder. Let’s examine which libraries require patches.

   - react-native-action-button+2.8.5.patch :
     First, the react-native-action-button is a library that implements a floating button. However, we encountered a warning message ”Animated: useNativeDriver was not specified” when using this library. To address this issue, we referred to the GitHub issue and found a suggestion to modify the library directly. Unfortunately, it has not been actively maintained for the past three years. As a result, we made the necessary modifications ourselves and proceeded with the patch.//
   - react-native-paper+5.10.4.patch :
     This library, which I will explain in more detail in the Atom folder, is the best implementation of Google’s Material Design System. It excels at implementing the Material Design System, but it has a limitation regarding the padding value of the label, specifically in TextInput. Unfortunately, this issue cannot be resolved through props. As a result, I had to fix this problem by directly modifying the library.

c) BloomMate-FE/src/atoms
Each folder within the src directory has a specific assigned role. The role can be determined based on the words between the dots in the module name. Let me first explain the atom module. The term ”atom” is an abbreviation for ”atomic” and is inspired by the atomic design system. While it cannot independently create functionality, it plays a crucial role in the UI by representing the most fundamental UI components. Although not all of the atom components follow Google's Material Design system, I have made an effort to align with it for most of them, although there have been practical compromises.

- button.atom.tsx :
  Buttons are components that allow you to trigger specific actions when they are pressed. The specific action is received through onPress. Buttons follow the material design guidelines faithfully and can be implemented in three modes (outlined, contained, text). The text inside the button can be obtained through children, and icons can also be displayed. When the button is pressed, a layer of color covers the entire button, indicating that the button is in a pressed state.

- divider.atom.tsx :
  A divider serves as a UI element that separates other elements. It is a thin line in a light gray color. If you want a thicker divider, you can set the ”heavy” parameter to true.

- icon.atom.tsx :
  Icons are a valuable tool for conveying information through visual symbols. In react-native, you can use icons by utilizing the react-native-vector-icons library. This library supports various icon sets, and we have decided to use the material icon set. To use material icons, simply import the package from react-native-vector-icons without any additional steps. When using an icon on a typical screen, you only need to provide three values: name, size, and color. It is worth noting that modern icons are treated as fonts, allowing you to use the Icon component within Text. This simplifies styling for many publishing tasks.

- image.atom.tsx :
  Images play a crucial role in the Frontend, as they effectively convey meaning to users. In BloomMate, users can add plants by taking pictures within the application, and images are also essential when viewing plant information and diagnosing plant photos. Additionally, images are necessary for various illustrations, logos, and examining plant lists and diagnostic logs. In BloomMate's overall architecture, all images are downloaded from cloudinary URLs. When using the native Image component of react-native, two problems arise. Firstly, a blank image is displayed while the photo is being loaded from the URL. Secondly, even if the image has been loaded once, it will reload when leaving and re-entering the screen. To address these issues, BloomMate utilizes react-native-fast-image. This library allows a loading component (Skeleton) to be displayed while the URL is being loaded. Once the image has been loaded, it is cached and does not need to be reloaded again. Certain images in the application, such as logos and seed-related illustrations, are accessed through fixed URLs. To improve the user experience (UX) and enhance the overall quality of the application, the preLoad function of FastImage is used to create cache values in advance when the application starts.

- modal.atom.tsx :
  A modal is a floating screen that appears above the existing screen. It allows users to display important information without navigating to a new page. Modals can provide a clearer workflow by showing the relationship between the floating screen and the underlying screen. In BloomMate, we use the react-native-modal library instead of the native modal provided by react-native for additional effects. The use of nativeDriver in Android and iOS for modal animations is particularly attractive as it helps optimize performance in large applications. BloomMate offers two types of modals: 1) a modal with elements positioned in the center of the screen and 2) a BottomSheet modal positioned at the bottom. Each modal is distinguished by the isBottomSheet prop. There is also the onBackDrop prop, which determines what happens when the user taps on the empty part of the modal. In BloomMate, it is mostly used to close the modal.

- point-linear-gradient.atom.tsx :
  LinearGradient is used in React Native to create gradient effects using the react-native-linear-gradient library. In BloomMate, gradient effects are utilized to enhance the design of specific UI elements. It is worth noting that all gradients share the same colors. To prevent redundant code, we have implemented this as a separate

component, adhering to the DRY (Do Not Repeat Yourself) principle.

- skeleton.atom.tsx :
  This is a loading animation that displays an approximate layout of the screen before the actual data is rendered. It helps to reduce user boredom during loading and decreases the chance of users leaving the page. BloomMate has chosen to use the react-native-shimmer-placeholder library among various Skeleton libraries because we believe that the shimmer effect provided by this library is suitable for the loading component. When using this component, it is important not to pass the width and height values as props. Instead, they should be declared within the style and passed as props.

- text.atom.tsx :
  Implementing the Text component in react-native can be challenging, especially for beginners. This is because:
  i) Unlike React, text must always be placed inside the Text component, which can lead to frequent mistakes.
  ii) There are numerous props and styles related to fonts, which require a significant amount of time and effort to write in code.

  To address these challenges, BloomMate has adopted the Typography approach from the Material System. Typography combines the word ”Type,” meaning ”print,” with the suffix ”-graphy,” meaning ”art or technique.” In modern typography, it not only conveys information but also communicates emotions and brand presence. Therefore, it is important to utilize typography as a design element that goes beyond just the concept of fonts. The Material 3 Design System’s Typography introduces three elements that control the variants:
  i) fontSize
  ii) lineHeight
  iii) letterSpacing

  (Material 2 also controlled fontWeight, but it has been removed.) There are a total of 15 variants that combine prefixes like ’display,’ ’headline,’ ’title,’ ’label,’ and ’body’ with suffixes like ’Small,’ ’Medium,’ and ’Large.’ fontWeight and color are limited to a few predefined options to ensure consistency. This approach results in consistent text and provides a sense of unity in terms of UI and UX.

- text-input.atom.tsx :
  TextInput is a component that allows users to input text. It follows the Material Design System and only has one outlined mode, unlike Button. Showing TextInput in various modes would disrupt consistency and cause user confusion. For this component, we used react-native-paper, which implements the material design system effectively.When using this component, the label and error are the most important aspects. In the case of TextInput provided by react-native itself, there is no label. Therefore, the only way to provide information about the text input was through the placeholder. However, in BloomMate, we provide both label and placeholder, allowing us to provide more diverse information through TextInput. For example, the label could say ”ID” and the placeholder could say ”Please enter within 10 characters.” The second important aspect is the error. This is especially useful when inputting information according to specifications, such as the Tiiun product key, which must be within 8 characters. The error works well with the validation of react-hook-form. By passing the error message through fieldState.error.message, TextInput displays the error message at the bottom when an error occurs.

- toast.atom.tsx :
  A toast message is a popup that displays a short message to the user and disappears after a certain period of time. An example of a toast message is the popup that appears when pressing the back button, displaying the message ”Press back button again to exit the app”. However, the toast messages provided by Android or iOS have limitations such as message length and position. In BloomMate, we have built a custom toast component using react-native-toast-message to overcome these limitations. To use this custom toast component, we declare it below the top-level App (in the case of BloomMate, it is declared below ¡RootNavigator/¿ in App.tsx). Then, within the screen, we can call Toast anytime using Toast.show to display a desired message at the desired position. As an example, in BloomMate, we used a toast message to inform users that they cannot add more plants when all the plant slots are already filled.

d) BloomMate-FE/src/assets
   The second folder in the src directory is named assets. Assets is where static data is stored. By pre-building the static data, you can ensure a smoother user experience without any loading during runtime

when users are using the application.

- img.asset.ts :
  This file contains the URLs of images. As explained in image.atom.ts, these URLs are predetermined and will be used directly in the application. They are not URLs that will be fetched from the server through GET requests. Managing these URLs in a single file helps maintain code structure consistency.

- lottie.asset.ts :
  Lottie is an open-source library created by Airbnb for displaying vector animations in web and mobile applications. It allows for the rendering of animations designed in After Effects, which are saved as JSON files, within applications. By using Lottie, designers can seamlessly integrate their After Effects designs into code, fostering collaboration between designers and developers and reducing the effort needed to implement animations. At BloomMate, we utilize lottie-react-native to display multiple Lottie animations. This file imports and converts the JSON data from different Lottie files into variables.

e) BloomMate-FE/src/screen
The third folder in the src directory is called ”screen”. The files within the screen folder can be divided into two types: 1) Navigator files - These files contain the navigator of react navigation, which groups multiple screens together. 2) Screen files - Each individual screen displayed in the application corresponds to one of these files. They can communicate with other screens using the navigation and route received from the navigator. The screens themselves follow a hierarchy of screen-module-components. This is done to implement the MVC structure in the frontend. The ”V” represents components, while the ”C” represents modules. In other words, the module level manages data logic based on hooks, while the component level handles all UI rendering. Finally, certain screens need to retrieve data from the server by making a GET request. In these cases, BloomMate uses react-query with the suspense option set to true. As a result, Suspense is added at the screen level for some screens, with fallback functions directly provided. Meanwhile, for other screens, Suspense is managed at the module level using Higher-Order Components (HOC). In the descriptions of the screens below, the following structure is generally followed: (1) Explanation of the overall logic of the screen using representative hooks. (2) Description of the types of modules. (3) Explanation of other peculiarities (UI-related peculiarities, logic-related peculiarities, etc.).

- root.navigator.tsx :
  This is the main file that serves as the starting point for all screens in BloomMate. The main file uses the Stack Navigator, which is the simplest navigator and provides easy methods for navigator reset and more. There are three things to consider regarding the main file. First, it manages the route parameters for each screen. By using types in the main file, each screen can receive the appropriate parameters. Second, the header of the navigator itself is not displayed. After several trials and errors, it was concluded that it is better to set the headerShown property to false instead of customizing the header of the navigator itself, and create headers in each screen. Lastly, the Bottom-Tab navigator is positioned within the Stack navigator. This means that the Stack navigator is the parent and the Bottom-Tab navigator is the child structure. This is because the bottom tab bar should only be exposed within the screens inside the Bottom-Tab. The Bottom-Tab navigator is declared in the primary navigator, which will be mentioned later.

- about-bloom-mate.screen.tsx :
  This screen consists of a WebView that connects to the BloomMate team’s Notion. In the team’s Notion, you can access information about what BloomMate is, why it was created, and who the creators are.

- article-webview.screen.tsx :
  This screen contains a WebView component where you can access articles written by experts about plants.

- community-qna-detail.screen.tsx :
  This screen displays the question and answer transcripts for the QA. It can have two states: 1) State with only a question - If there is only a question, the module responsible for managing the answer will not be shown. On the question side, a status message ”Answer Required” will be displayed. 2) State with an answer - If the module managing the answer is also shown, on the question side, a status message ”Answer Completed” will be displayed. In BloomMate, the administrator manages the answering process. Therefore, the profile for the answer will consist of BloomMate icons. Data is obtained through the useGetQuestionDetailQuery, and the necessary ID value for the GET request params is passed via the screen router. To ensure that the answer module is not shown if there is only a question,

the screen handles the query loading using an early return structure. (This is a rare structure in BloomMate.)

- community-qna-post.screen.tsx :
  This screen provides a space for users to ask questions in the Q&A section. There are two main parts on this screen. 1) The useQnAForm component, which uses react-hook-form, handles the user input when asking a question. It ensures that the specified requirements (e.g., entering a question title and content) are met. The form consists of four sections: a header, a title field, a content field, and a footer. 2) The usePostCommunityQnaMutation component, powered by react-query, handles the submission of the form data to the server. The types required in the body of the post request perfectly match the types provided by useQnAForm. As the post request is being processed, a MutationIndicator is displayed on the screen. Once all the requests are completed, the screen reverts back to its original state using the navigation.replace function. One notable feature in terms of user interface is the TextInput component, which allows users to input the question content. By supporting multiline input, users can write the question content across multiple lines.

- landing.screen.tsx :
  This screen is the first thing users see when they arrive on BloomMate. It shows the BloomMate logo and an image that symbolizes what BloomMate represents. At the bottom, there are two buttons for logging in and registering. The login button has a solid style, while the registration button has an outlined style. This distinction is made because the login function is usually more important for regular users, as they are less likely to register every time. The module is divided into sections for easy content management and footer display.

- login.screen.tsx :
  This screen is designed for users to log in. It has two main components that serve specific purposes: 1) The useLoginForm hook, which utilizes react-hook-form, is responsible for managing user input data during the login process. It enforces various constraints, such as requiring a minimum of 5 characters for the ID and allowing only English letters and numbers. All of these conditions can be handled within useLoginForm. 2) The usePostLoginMutation hook, which utilizes react-query, is responsible for sending the form data received from the user to the server. The required types for the body of the post request are perfectly matched with the types provided by useLoginForm. While the post request is loading, the screen displays a MutationIndicator. Once all the requests are completed, the screen navigates to the primary screen using the navigation.reset function. This ensures that the landing screen is not shown when navigating back, as the navigator stack is managed accordingly. Additionally, before performing the reset action, usePostLoginMutation sets the access token in the header of defaultAxios. This step is crucial for authenticated communication with the server when using most BloomMate features after logging in, as the token needs to be included in the header for authentication purposes. In terms of UI, this screen follows a standard design with a header, an ID input field, a password input field, and a footer.

- plant-add.screen.tsx :
  This screen allows users to add plants. The primary hooks used in this screen are: 1) usePlantAddForm with react-hook-form. 2) usePostPlantAddMutation with react-query. Since these hooks have been already described in detail above, we will skip the explanation as they have the same logic. The important thing to note is: The management of steps in the plant addition screen using recoil. As specified, the plant addition process consists of multiple steps. Although React is a Single Page Application (SPA), it will appear to users as multiple screens composed of steps. To handle this, we have introduced recoil as a global state. In BloomMate, the prefix ”$” is used for atom values in recoil. The atom used in the plant addition screen is $plantAddState. In summary, the logic of the plant addition screen is as follows: 1) Receive user input and manage it using react-hook-form. 2) If valid user input is received, enable the buttons in the footer. 3) By pressing the button, move to the next step using recoil. 4) Render the corresponding module for the next step. 5) Repeat this process until the last screen, where usePostPlantAddMutation is called. 6) If the plant is successfully added to the server, reset the recoil state and switch screens (this process is resolved using useWillUnmount). The modules consist of three parts: header, contents, and footer. In the header, if the back icon is pressed and it is the first screen, it goes back; otherwise, it goes to the previous step. The logic described

in the plant addition screen is also applied identically to the sign-up screen. Finally, there is a special logic specific to the plant addition screen, which is photo registration. This is implemented in the following order: 1) First, press the photo add button to open a modal. 2) In the modal, choose whether to take a photo directly or choose a photo from the gallery. This is implemented using useUploadPhotoMutation and useUploadImageLibraryMutation, respectively. 3) Upload the selected photo to Cloudinary. 4) Insert the uploaded URL using usePlantAddForm. In other words, the data passed to the BloomMate server is a Cloudinary URL string. This logic is also used in plant diagnosis.

- plant-chat.screen.tsx :
  This screen enables users to have conversations with plants. The main features on this screen are: 1) useGetChattingContentsByDate - a function that retrieves all chat history on a specific date. 2) usePostPlantChat - a function that sends the user's message to the server. 3) $plantChatState using recoil - a state management library. These functions are interconnected, and their logic is not separated using useEffect. Therefore, I will write the logic in the order of events that occur when entering the screen. 1) When entering the screen, the useGetChattingContentsByDate function is executed. Let's assume a scenario where there are no chats for today (empty data received from the server). 2) In the center of the screen, there is a button with the text "Check Today's Report". When this button is pressed, the user's message "How is today?" is added to the $plantChatState. 3) By using useEffect, if the last message in the $plantChatState is the user's message, it is sent to the server using the usePostPlantChat function. 1) At the same time as sending it to the server, the server-loading value is added to the $plantChatState. This allows showing a message to the user indicating that the server is loading. 2) If a value is received from the server, the server-loading message, which was the last message in the $plantChatState, is changed to the server message and shown to the user. 3) In the scenario where there are chats for today, as in case 1, the received value from the server is separated into user-server values and added to the $plantChatState. The subsequent process involves receiving messages from the user through TextInput. Additionally, there is a feature to view past messages through a floating button. In this case, the useGetChattingContentsByDate function can be used to retrieve past messages. Once you go back to a past point in time, chatting is no longer possible. This screen is unique because Suspense is created as an HOC (Higher Order Component) within the module. It had to be created within the module to preserve PointLinearGradient. The recoil state is automatically reset with useWillUnmount.

- plant-detail.screen.tsx :
  This screen allows users to access detailed information about planted plants. Users can review the plant's details and view photos taken during the plant's addition. A bar graph has been included to show growth information, and it utilizes MotiView for animated effects. MotiView is a library that enables the use of animations in react-native, similar to CSS animations. Upon entering the screen, there is an animation where the screen slides in from left to right using translateX. From this screen, users have the option to navigate to the plant-detail-edit screen and plant-diagnosis-intro screen. However, during the harvesting season for a plant, the bottom button changes to "Harvest." Pressing this button will initiate the harvesting process. (Harvested plants can be viewed in the primary-plant-harvested-list.)

- plant-detail-edit.screen.tsx :
  This screen allows users to edit the information of the plants they have planted. The plant information itself, such as the planting date and crop type, cannot be modified. However, the name and photo of the plant can be changed. Two important functions are used on this screen: 1) usePlantDetailEditForm using react-hook-form, and 2) useUpdatePlantMutation using react-query. These functions have been explained in detail in the previous section. I will only explain the different parts of this screen, excluding the parts that have the same logic. The main distinction lies in the defaultValue. While other screens start with empty input values, this screen is for modifying existing data. Therefore, there are input values that the user has already entered, and these values need to be set as defaultValue. These input values can only be obtained from the server. In such cases, react-hook-form provides a method called reset. By using useEffect, the retrieved data is set as the default value using reset when the GET request is successful. After that, normal modification can be done. Once the modification is made, the navigation is reset

using navigation reset. This is necessary to avoid various bugs that may occur when using goBack, navigate, replace, etc. (such as the existing screen not being updated and having two plant detail screens).

- plant-diagonsis-intro.screen.tsx :
  This screen provides information and allows users to diagnose plants. The contents module includes helpful images and phrases regarding plant diagnosis. The footer module is significant and contains a CTA-Section with buttons that direct users to the ”Diagnosis List” screen and the ”Diagnose” screen. When users tap the ”Diagnose” button, they will be prompted to either capture a photo or select one from the gallery, similar to the screen for adding a plant. The chosen or captured photo will be uploaded to Cloudinary, and the corresponding URL will be sent to the server. Once the server sends the ID value for the diagnosis result, users can proceed to the ”Plant Diagnosis Result” screen.

- plant-diangosis-list.screen.tsx :
  This screen acts as a list screen for plant diagnosis results. It allows you to view the list of diagnoses that have been performed so far. The main feature of this screen is the use of FlatList, a component in React Native that improves the performance of rendering lists. FlatList is specifically designed to handle long lists with many items efficiently and optimize scrolling performance. It also takes care of item spacing and top/bottom components when there are no items. In BloomMate, FlatList is not only used on this screen but also on various other screens, which will be discussed later. Another important aspect of this screen is the utilization of FastImage. During the development of BloomMate, Images were replaced with FastImage to manage caching. It was found that when using the react-native Image component on this screen, images were not rendered correctly when there were more than five lists. This was because there was no caching, which caused the images to be rendered from scratch each time and put too much load on the UI thread. By replacing it with FastImage, we were able to successfully render the images without any issues.

- plant-diagnosis-log.screen.tsx :
  This screen functions as a log for plant diagnosis. It is similar to the screen displaying the results of a plant diagnosis, but it does not include any components or logic related to GPT. In simpler terms, once a plant has been diagnosed with a disease, it cannot be removed from the log screen. It will require another diagnosis.

- plant-diagnosis-result.screen.tsx:
  This screen appears immediately after diagnosing a plant and provides detailed explanations of the diagnosis results. If the plant is diseased, additional detailed explanations related to the disease are also included. A button is provided to purchase new seeds recommended by GPT automatically, and clicking this button will delete the corresponding plant.

- primary.navigator.tsx:
  This navigator is a bottom-tab navigator that manages the screens to be included, similar to the root navigator. It is named ”primary” because screens with bottom tabs are perceived as the most important from the user’s perspective. The settings of the navigator are similar to the root navigator overall, with the addition of managing the style of the bottom tabs.

- primary-plant-list.screen.tsx:
  This screen is displayed when the first tab of the bottom tabs is pressed. It includes two additional screens configured with a material-top navigator: the current plant list screen and the harvested plant list screen. One important point to note is that the tabs of the top navigator are custom implemented to match the design prototype, requiring the code to be written from scratch to implement the tabs.

- primary-plant-harvested-list.screen.tsx:
  This screen appears when the first tab at the bottom is pressed, and the first tab at the top is pressed. It displays a list of currently grown plants. Similar to other lists, it is implemented using FlatList. Additionally, it includes a floating button and toast message implementation.

- primary-plant-current-list.screen.tsx:
  This screen appears when the first tab at the bottom is pressed, and the first tab at the top is pressed. It displays a list of plants that have already been harvested. Similar to other lists, it is implemented using FlatList.

- primary-community.screen.tsx:
  This screen appears when the second tab at the bottom is pressed. Within this screen,

two additional screens are constructed using material-top-navigator (Q&A and article screens). It also implements a custom tab for the top navigator.

- primary-community-qna.screen.tsx:
  This screen appears when the first tab at the top is pressed within the second tab at the bottom. It displays a list of Q&A. Similar to other lists, it is implemented using FlatList.
- primary-community-article.screen.tsx:
  This screen appears when the second tab at the top is pressed within the second tab at the bottom. It displays a list of expert articles. Similar to other lists, it is implemented using FlatList.

- primary-my-page.screen.tsx:
  This screen appears when the third tab at the bottom is pressed. It consists of buttons for logging out, checking user information, and accessing information about BloomMate.

- signup.screen.tsx:
  This screen is used for signing up. The logic is the same as the plant-add screen. Two different aspects are: 1) receiving input with toggle buttons and 2) using the daum-postcode library to input addresses. After completing the sign-up, the user is directed to the login screen to log in.

- user-info.screen.tsx:
  This screen can be accessed from the primary-my-page screen. It allows users to check the information provided during sign-up.

f) BloomMate-FE/src/provider
The provider is located in the 4th folder of src. In React, the provider refers to components that need to be positioned higher in the DOM tree. Developers can create their own providers using the Context API or use libraries that manage global states. These libraries often provide a specific component with ”provider” as a suffix. While it may seem possible to directly include such libraries in the App without a provider folder, in most cases, these libraries require detailed configurations. That’s why the provider folder was created to handle these configurations.

- mutation-indicator:
  The name ”mutation-indicator” was inspired by the useMutation function in react-query. Its usage is simple. The useMutationIndicator hook takes an array as its first parameter. You can include the loading variables of requests (POST, DELETE, PATCH, etc.) that send data from the client to the server in this array. If any element in the array is true, a modal will appear, covering the entire screen. This modal is used to indicate that data is being sent, and it shows an animation using Lottie.

- query-client:
  This provider is used for working with react-query. Some settings have been implemented to make it more convenient when using useQuery, and additional code has been written to capture react-query in Flipper.

- recoil:
  This provider is used for working with recoil. Note that it must be positioned at the top of the DOM tree. Additional code has been written to capture recoil in Flipper.

- ui:
  This provider contains various libraries and components related to UI. Firstly, there are providers for react-native-paper and mobily/stacks. Secondly, there are providers and components related to SafeAreaView. SafeAreaView ensures that the screen is not covered by the notch at the top and the physical buttons at the bottom on iOS.

g) BloomMate-FE/src/layout
The 5th folder in src is called ”layout”. It contains larger components compared to Atom and is designed to receive children or equivalent objects as props. The purpose of the layout is to create the structure of the screen rather than provide functionality.

- basic-layout:
  This layout is the most fundamental one. It manages padding values on the left, right, top, and bottom sides, and sets the background color to white. Unless a screen uses gradients, the basic-layout is included as the top-level component in all screens by default.

- CTA-section:
  CTA stands for Call-To-Action, which aims to encourage customers to perform desired actions. CTA-section is created for buttons located at the bottom of the screen, and can accommodate up to 2 buttons. These buttons can be arranged horizontally or vertically, and typically consist of one outlined button and one contained button to create contrast.

- dialog:
  Dialog is used to place text and buttons in a pre-agreed specific location within modals. It consists of four elements: 1) title - the title of the entire dialog content, 2) content - additional explanation about the dialog content, usually describing what happens when the button is clicked, 3) okayButton - a button that usually contains the text ”OK” and triggers the intended action, and 4) cancelButton - a button that usually contains the text ”Cancel” and simply closes the dialog without any further actions. By placing these four elements in a pre-agreed location, overall UI consistency can be achieved.

- loading-page:
  This layout is typically used with Suspense. Ideally, a Skeleton should be created with Suspense for each component that receives data. However, due to development schedule and UI irregularities (too many skeletons can be visually overwhelming), a loading-page can be applied to the entire screen. By placing an ActivityIndicator in the center of the screen, it clearly indicates the loading state.

- modal-header:
  This layout is commonly used at the top of most screens. It manages commonly used icons, types of title text, and more. The important point of modal-header is the use of useBackHandler. This hook determines how the physical back button should work on Android. Typically, the back button in the modal-header should have the same logic as when the back icon is pressed. Therefore, useBackHandler is used within the modal-header to manage this.

- scroll-view:
  At first glance, this layout may seem similar to the scrollView provided by react-native itself. However, BloomMate has many screens that receive input through TextInput. To detect when the keyboard is raised on such screens, the modified version of KeyboardAwareScrollView provided by react-native-keyboard-aware-scroll-view is used in this layout.

h) BloomMate-FE/src/hook
   The 6th folder of src is called ”hook”. Hooks in this folder do not provide a detailed explanation for each file because their logic is almost the same. They were placed directly under src because they can be considered as top-level hooks, meaning they can be used in multiple screens. Firstly, due to the nature of REST API, GET requests should be used in multiple screens. This aligns with the design of REST API, where a large amount of information is fetched at once and then repeatedly used in various locations through caching. Secondly, if a library is related to images, it is currently being used repeatedly in various parts of the screen. Therefore, it was positioned as a top-level hook. Hooks are responsible for the ”C” (Controller) in the MVC (Model-View-Controller) structure. They receive specific inputs or declare them in a declarative programming style and fetch data from the model. Then, they play a role in passing the data to the view (screens) through the return statement.

D. Module2 - Backend

1) Purpose
   The role of the backend is to build and manage back parts such as applications and databases. It builds server logic to return appropriate outputs according to user requests, and manages databases to store, retrieve, update, and manage input and output data. Backend designs the architecture of the data flow in the back of the application, helping to organize the structure of it. Backend also develops and manages APIs to allow the application to communicate with users. It Maintains security to prevent unauthorized access to user information and to prevent direct hacking or errors that may occur. We chose Django for our backend database management for the following reasons Django is written in Python, so a lot of functionality is built in. This allows developers to focus more on the core functionality of the application and not waste time on the basics. Django also simplifies working with databases through its object-relational mapping (ORM), which allows developers to easily manipulate databases without having to write SQL themselves. In fact, its support for sqlite as a database allows code written in Django to store data in the correct format for sqlite. Django has an administrator interface that makes it easy for developers to see what’s actually being stored and manage the site when they deploy the backend part.

2) Functionality
   The BloomMate application allows users to perform various activities related to plants. BloomMate processes the information which user enters into the application and stores it in a database. When the user needs information, it returns useful data to the user on demand. Users can register, manage, diagnose and chat with their plants. All information generated when using these features is automatically stored in SQLITE. In the case of plant diagnostics, the user can use an function of AI running on the backend to determine the presence of a disease. User can also use Chat GPT-4 to feel the dialog with the plant based on the plant information registered by the user, weather information based on

the address location information, etc. All these user inputs help to make chatting with the plant feel natural.

3) Location of source code:
https://github.com/BloomMate/BloomMate-BE

4) Class component
   - accounts
     - admin.py : Administrators can check the information of users who have registered through the application by accessing the administration page. When he accesses the page, he can see the user's ID, name, TIIUN number, garden size and address. In addition, he can use filters to categorize users based on the characteristics added according to the type of user set in the user model. He can search for users by ID and username, and view users in order of the date they joined. The Admin page also allows administrators to edit user information and add users. At this point, we specify which parts can be changed and which parts must be filled in when a user is added. This is the same information that regular users enter when they sign up or edit their information.

     - models.py : This is the default setting for processing the user's information that is set when the user registers and logs in.

       * Create User : This helper class is used when a regular user logs in through the application. If the user does not enter the required information (ID, password, name, TIIUN number, garden size, and address), we set it up to display an error saying "This field is required. We also made sure to store this information in the database.
       * Create Super user : This helper class is used to create administrators. You can create an administrator using django's create superuser, not your application. The information that you have to enter is the same as what a regular user has to enter when they sign up. The difference between a regular user and an administrator is that they have additional privileges. Administrators are the only ones who can access the Admin site and have full access to view, modify, and delete information stored on the Admin site. Administrators are also stored in the database in the same way as regular users.
       * User : When a user or administrator signs up, the parent helper class can generate information, and this is the model for handling that information. We specified the type and length for each piece of information. We also included the joined date and the date the information was modified. We've also included partial settings for permissions to differentiate between administrators and regular users. In the case of ID, we set it to unique to avoid duplication. We also specified the fields that must be filled in during the sign up process.

     - serializer.py : When passing the information set in the user model to the API, set the information to pass. When a user registers or logs in, we set it to use all the information the user has entered.

     - views.py : The View role in Accounts folder represents server-side management of the functionality available to users when they first connect to the BloomMate application.

       * Sign up : Users can register if they have never used the BloomMate application before. The BloomMate application uses a REST API and also uses JWT tokens. Before using any feature of the application, the user must verify that he is an authorized user using a JWT token, but in the case of registration, this is not necessary because the user has not yet registered. Therefore, all users are free to use the function and once all the required information has been entered, the backend will return the information entered by the user, the message "Registration Success" and a status of 200.

       * Log in : Users can log in by sending a POST request with the ID and password they created when they registered. BloomMate issues a new access token to the user for each login. The validity of the access token is set to one day. The token allows the user to authenticate to use other features in the application. Upon login, the issued token is stored in the user's device cookie. The return value of the POST request is the information the user entered when logging in. If the username or password is incorrect or does not exist, status 400 is returned with the message "The entered username and password do not exist.

       * User Info : Users can verify their information through GET requests. To check the user's information, the user must authenticate with the JWT access token issued during the login. Once the token is verified, the user can check their ID, name, TIIUN number,

garden size, and address information. The user can also change their information by making a PATCH request. The user can change the rest of the information except the ID. However, you cannot set the information to be modified to empty. If you try to force the ID to be changed, the message ”This information cannot be changed” is returned. If the modification condition is met, all of the user’s information is returned, including the modified information. We have implemented logout using the DELETE request. The logout works by deleting the access token stored in a cookie on the user’s device. We made sure to return the message ”Successfully logged out” so that the user can verify that the logout was successful.

    * Token Verify : Here the user can check the validity of the issued token with a GET request. The user can determine if the JWT token stored in the cookie is available. This function is not available after logout or before the first login because the token is not stored in the cookie. In this case, it returns the message ”The token does not exist in the cookie”. If the token is present in the cookie, decode the token to determine if it is valid. If it is valid, return the message ”Token is valid” along with the token information; otherwise, return the message ”Token is invalid”.

- articles
  - admin.py : For articles, we adopted a format that allowed administrators to add them directly from the backend and make them visible to regular users. To accomplish this, we wrote code to allow admins to add, edit, and delete articles directly from the admin page.

  - models.py : We set the model to the article. We set the ID value of the article as the primary key. For the article, we need the title, content and thumbnail image. For the image, instead of storing the file directly in the database, we processed the image in the frontend and stored the received URL to reduce the database storage capacity. For the content, we chose to connect to an external notion site and display it in the web view. We also minimized the amount of data actually stored in the database by connecting to the URL instead of adding the content directly.

  - serializer.py : When passing the information set in the article model to the API, set the information to pass. We decided to use all the data we set as input from the article model.

  - views.py :
    * Article List : The user can view the article via a GET request. As with other functions, the user must authenticate with a JWT token. All of the article’s data set in the model can be retrieved from the database.
    * Create Article : The administrator adds articles directly in the backend. You can add an article using a POST request, or you can go to the admin page and add it using a POST request. You must enter and submit the data specified in the article model. The permission is set to isAdminUser to specify that only administrators can do this. The JWT token is used to determine if the user is an administrator. General users cannot register items because they are different from administrators.
- chatting
  - admin.py : The Admin page allows administrators to review and manage the history of users’ chats when they use the chat service. In this case, they can see a list of chats on the screen, and the parts of the list that they can see are the date of the chat, the content of the chat, whether the content was sent by the user, and the plant ID of the chat. By checking the content of the chat, administartors can make sure that there are no errors in the service.

  - models.py : When setting up the chat model, the biggest question was how to store the conversations between the user and Chat GPT, and in what format. Originally, we planned to store the input as user chat and the output as Chat GPT’s response, but we decided to treat all chat content as ’chatting content’ and use the ’is user chat’ variable to distinguish whether it is a user chat or not. Since the idea of chatting is to have a conversation with a plant, we made sure to get the plant’s ID value as a foreign key. Since a chat can be with more than one plant, we set up a one-to-many relationship.

  - serializer.py : When we send the values set in the chatting model to the API, we specified the values to use. In the model, we imported the plant ID as a foreign key, and we made sure that the serialized values included not only the plant ID, but also the plant information associated with the ID, and the plant details. In the case of the plant details, we set ’read only’ to ’true’ so that it cannot be changed because it is a fixed value. When fetching the plant information, we

used a separate serializer created by plant to return the plant information.

- utils.py : For chatting with plants, we have implemented several functions that are separate from the view. The information needed to chat with a plant is the plant information, the weather at the address you entered, the soil information provided by Tune, and the date of the last visit.

  * Get City Address : We need the user's address as chat information to get weather information for that address. We set to get the address information that the user entered when signing up for membership. However, to get the exact location of the address, we used 'split' and 'join' to get only the street address without the street number or apartment number. To get weather information, we need to use the latitude and longitude values of the address, so we used 'geopy'. We set it to return the latitude and longitude of the address, and if there is an error in the address, it will return 0 and 0 respectively.

  * Get Weather Data : We have used the OpenWeather API to get weather information. The API key is stored separately in the '.env' file along with the Chat GPT API key for chat for security purposes.If you enter the latitude and longitude values from the address along with the API key into the URL where you can get the weather information, you can get it for that location. We set the required information to "temperature", "humidity", "weather information (sunny, cloudy, rain, fog, etc.)", and "region" among other information that can be obtained from the API. If any other error occurs, we want to return the 'weather information not found' message.

  * Get Soil Condition : As for the soil condition, the original idea was to use the 'LG TIIUN' to get the soil condition using TIIUN's API. We decided that one of the most important information for growing plants, besides the weather, is the condition of the soil in which the plants are planted. So we categorized the soil into 'good' and 'bad' and set the soil condition to be attached to the information in the chat.

  * Get Last Vsit Date : An important part of growing a plant is being able to see it in person to make sure it's okay. To do this, we realized that users would need to know when they last visited while chatting, so we set the most recent date of "Diagnose a Plant" to be the date of the last visit, knowing that when users diagnose a plant, they take a picture of the plant's leaves. To do this, we set up the Plant Diagnosis Record database to get the most recent diagnosis date. The date format is 'year-month-day'. If the plant has never been visited, we want it to return a message saying "The plant hasn't been visited since planting.

  * Generate Chatgpt Response : This function connects to Chat GPT-4 and receives chat responses. We chose to use Chat GPT version 4 because it has been trained with more recent information than its predecessor and has the ability to better understand and generate more advanced languages spoken by humans. In keeping with the concept of chatting with plants, the default system prompt in Chat GPT is set to the plant itself, based on the plant information provided by the user. The function brings the history of the user's conversations with the plant by date and enters it into the assistant prompt, allowing the conversation to continue naturally. It also pulls in weather information, soil conditions, and the name of the user who registered the plant. The user only needs to type a phrase like "How is the plant today?" but in the backend, the server enters other information, such as above, to get the desired answer. The backend returns the answer along with the soil condition, whether the user is chatting, and the plant ID.

- views.py : Users can get a list of plants they've chatted with by making a GET request. When making a GET request, it is important to note that each user will have different plants that they have chatted with, so you will need to include the plant ID in the query string. You will also need to include the date in the query string in the form of 'year-month-day', as the number of chats will increase the number of chats displayed on the screen, which may cause delays in loading the chat history. Use a JWT token to verify that the asset was added by the user. If all three conditions are met, the chat history will be retrieved from the chat database based on the plant ID, the date, and the ID of the user who chatted with the plant. The API values returned to the user are 'chat ID', 'chat content', 'check user chat', and 'plant name'. The user can initiate a chat using a POST request. As with the GET request,

the chat will receive the plant ID and date as a query string. In this case, the date is used so that if the current date and the date entered in the query string are different, the message 'not today's date' is returned. We also set the 'is user chat' variable to 'true' because the user enters the content they want to send at the same time as the POST request. Then we call the 'generate chatgpt response' function to get the chat GPT response. In this case, the 'is user chat' variable is set to 'false'. After the above process, we save it to the database.

- community
  - admin.py : On the admin page, admins should be able to see the questions users have asked and the corresponding answers, so we've created separate pages for questions and answers.
    - Question : For questions, we have made it possible to see the "question ID", "user asked", "question title", and "question creation date". When searching for questions on the admin page, admin can search by username or question title, and the most recent questions will be displayed at the top based on the question creation date.
    - Comment : For answers, the administrator can see the answer ID, the question to which the answer was attached, the content of the answer, and the date the answer was created. When searching for answers, the administrator can search by the content of the question to which the answer is attached or by the content of the answer. As with questions, the most recent answers are displayed at the top based on their date of creation.
  - models.py : We set up a model to store in the database. There are questions and answers in the community, and by implementing the model separately, we set the relationship between questions and answers.
    - Question : For questions, we linked the user to the foreign key. Since a user can ask multiple questions, we set up a one-to-many relationship. The question model contains the following information: question ID, question title, question content, question creation date, and question poser.
    - Comment : Answers have a one-to-one relationship with questions. A question can only have one answer, because the answer only deals with the part that the administrator directly answers in the backend. So it is a one-to-one relationship. The comment model includes the comment ID, the question content of the commented question, the comment date, and the comment content.
  - pagination.py : If the community posts a lot of questions and it becomes difficult to view them on one page, we split the page into many pages. We set the page size so that only 10 questions can be displayed on one screen. When users enter the page they want to check in the query string with the word 'page_size', the community page is divided into several parts so that you can check the 10 questions that belong to that page.
  - permissions.py : For a community, we want everyone to be able to see the question, but we don't want anyone to be able to edit or delete the question unless they are the author. We use permissions to differentiate between users, so if the user is not the author of the question, we allow the read permission request, but return false to deny editing and deleting since they are different from the author of the question.
  - serializer.py : This is the part that decides how to serialize the dataset from the model. Since we separated the model into questions and answers, we should do the same for the serializer.
    - Question : For questions, we decided to serialize all data created by the model. To get information about the person who created the question, we use the user's ID in the serializer. We don't want the user information to be modifiable, so we set it to ReadOnlyField.
    - Comment : For the comments, we use all the information received from the model, but set the question as a part that is not required to be entered when using the POST request from the view. Since we will be using a query string to categorize the questions in the answer, we set the requirement to 'False' so that only the answer content can be written.
  - utils.py : Since the relationship between a question and an answer is a one-to-one relationship, we need to check if the question has an answer before adding the answer to the question. In this case, we return 'True' if the answer to the question exists and 'False' if the answer does not exist.
  - views.py : It is responsible for the creation of an API that can be used by users on pages that form

a community around questions and answers.

* Question List : A user can use a GET request to get a list of all the questions in the community. The user must use a JWT token to prove that they are a logged-in user. This is because the list of questions should only be visible to the logged-in user. When viewing the information in the list of questions, an additional piece of information that the user can see is ”check if a response has been made”. This variable is not handled by the model or the serializer, so we need to handle it separately. We have added this information to the serialized information for the question so that it can be checked. Also, we don’t need the answer content or username in the serialized information, so we removed it from the API that can validate the list of questions. We return the organized information with a status of 200.

* Create Question : Users can register new questions using POST information. As with other features, this requires user authentication via a JWT token. The user sends a POST request with a question title and question content in the body. The question ID and question registration date are generated at the same time as the POST request, and the user information is set to automatically add the information of the user who submitted the question when the POST request is sent.

* Question Detail : This API provides a detailed view of a single registered question. Users can retrieve the information using a GET request. However, they can decide which question they want to get details about by specifying the question ID in the query string. They should also be able to verify that they are an authenticated user with a JWT token. In the details, users should also be able to see if the question has been answered. In the details, they should be able to see not only the question, but also the comments to the question, so you should use a question serializer and an answer serializer to retrieve information about both the question and the answer. You can also use the JWT token to verify that the question was created by a user, which allows you to edit and delete registered questions. To edit a question, you can put the part you want to edit (title or content) in the body and use the PATCH request. To delete a question, you can also use the DELETE request to delete the question associated with the question ID. If the question was not created by you, both editing and deleting will return the message ”This question was not created by you. You can’t edit or delete this question”.

* Create Comment : For registering comments, we used a method where the administrator adds them directly from the backend using POST requests. Therefore, we used ’IsAdminUser’ from Permissions in this case. To determine if the person trying to add a comment is an admin, we use a JWT token. Before adding a comment, we get the question by entering the question ID via the query string. If the question doesn’t exist, we get the message ”Question does not exist”. Also, since a question can only have a single answer, we must first determine if a comment exists. If a comment exists, it will return the message ”You can’t add a new comment because it already exists”. If these conditions do not conflict, the administrator can add a comment which will set the ’is_answered’ variable to ’true’ and return with the new comment information stored in the database.

- plants
  - admin.py : The information that can be found on the Admin page related to plants is categorized and displayed on each page. The information is categorized into plant information registered by the user, specific details about the plant, information about the plant’s disease, and records of the user’s disease confirmation.

    * Plant : The information related to the plants is the same as the data set in the model. In addition, the Admin page allows you to determine germination, growth, and harvest periods based on the date and time each plant was planted. On the Admin page, the administrator can review the plant information added by each user and add, edit, and delete plants directly.

    * Plant Type : The type of plant, i.e. the details of the plant, can be added directly by the administrator via a POST request from the admin page. Users can only grow the added plants directly in TIIUN. Users can see this information when viewing their personal plant information, but cannot edit the details.

    * Plant Disease Type : For the Plant Disease type, the Disease ID and the name of the

plant disease are displayed directly on the Admin page. Administrators can add each disease information via a POST request from the admin page. Disease types also include ”healthy”. For each disease, the symptoms and causes of the disease can be found on the page that appears when you click on the disease information.

* Plant Disease Record : The Admin page allows each user to manage their personal plant diagnosis records. On the Admin page, the Admin can see the ID of the diagnosed plant, the ID of the disease, and the date the diagnosis was made. When he clicks on each disease record, he can see only the Plant ID and Disease ID. The other information is available on the previous page, so he can review only the necessary information on the following page.

– models.py : This is the part of the model configuration related to plants. We have divided the model into 4 categories to avoid duplication of information in the database.

  * Plant : To add a plant, we need the following information: ’plant nickname’, ’plant photo’, ’plant type ID’, ’plant date’, ’harvest date’, and ’added by’. For the plant photo, we chose the same format as the other images, processed it on the front end to get the URL, and stored the URL in the database. Also, the plant type and plant need to be related, so we linked them with a foreign key. The harvest date was set to a null value as initialization since it was for plants that were still growing. A single user can manage multiple plants, so we also linked this with a foreign key.

  * Plant Type : Regarding the details of the plant, we included a number of pieces of information, including the variety of plant the user planted. The data included ”ideal temperature,” ”ideal humidity,” ”ideal light,” ”bloom season,” ”watering frequency,” ”ease of care,” and ”precautions”. The names of the variables set in the model may be different from those given, and the reason for not using the names of the variables in the model is that we have chosen different terms for quicker understanding in the docs. We also included the start and end dates of each cycle to calculate germination, growth, and harvest times.

  * Plant Disease Type : For plant disease types, the administrator stores the name of the plant disease, the symptoms of the disease, and the cause of the disease in a database. Note that the relationship between the plant database and the plant disease type database is a many-to-many relationship.

  * Plant Disease Record : The relationship between the Plant and Plant Disease Type databases is many-to-many. For a many-to-many relationship, instead of connecting the two databases at once, we need to create another database in between to form a one-to-many relationship with the Plant database and a one-to-many relationship with the Plant Disease Type database. This is where the Plant Disease Record database comes in. When a user takes a picture of a plant’s leaf to check if it has a disease, the ’Date diagnosed’, ’Disease picture’, ’Plant ID’ and ’Disease ID’ are stored. In the case of the disease picture, we minimize the load on the database by storing the URL and, as mentioned earlier, foreign keying the relationship to the Plant database using the Plant ID and to the Plant Disease Type database using the Disease ID.

– serializer.py : It is a file that serializes the dataset in the model. Since the model is divided into 4 types, the serializer is also divided into 4 types. For Plant Type, Plant Disease Type, and Plant Disease Record, all three serializers are set to serialize all data received from each model. However, in the case of Plant, we also need to retrieve the user information. Therefore, in this case, the user’s information is obtained from the ID that the user entered when logging into the serializer. The user information should not be modified, so we made sure to get it as a ReadOnlyField.

– utils.py : This is a file that defines additional functions needed to provide the dataset as a model as an API for users to use functions related to plants.

  * Determine is Harvested : When viewing the list of plants, we worked on the frontend to change the plants displayed in the list based on whether the plant is harvested or not. To do this, the backend needs to check if the plant has been harvested. Plant data has a harvest date stored in the database called ’harvested at’. If the plant has not been harvested, the default value is null, and if it has been harvested, the harvest date is stored in the database. This function is set up to check the value stored in the database and

return true if the date value is stored or false if there is no value.

* Calculate Growth Level : Each of your plants has a germination period, a growing period, and a harvest period, all with different durations. So this function will calculate which period is how many days after the planting date. We have stored the start and end dates of the germination, growing and harvest periods in the Plant Type database. We've set it up to calculate the difference between the current date you're connected to the application and the date you planted the plant, and if that value is between each time period, it returns that time period. If you checked right after adding a plant, it might show up as day 0, so we included it in the germination period and made sure that if it hadn't been harvested by the end of the harvest period, it would still show up as harvested.

* Resnet AI to Check Disease : This function is for the diagnosis of plant diseases. It uses AI to determine whether a leaf in a photo is healthy or diseased. The backend sends the diagnostic photos to the function via a URL in the database. There are a total of 14 classes that the AI can recognize, including diseased and healthy conditions for corn, strawberries, potatoes, and tomatoes. By storing the URLs of the photos in the database separately locally, the stored photos are evaluated by the AI. The AI model uses Resnet AI and returns the results of the diagnosis to the user. The function returns 14 of the class, numbered 0 through 13.

* Get Predicted Name : This function associates the numeric value returned by the Resnet AI to Check Disease function with the corresponding disease name. This function places the disease names in the same order as the class name set in the parent function and returns the name of the disease by determining its location from the received numeric value.

– views.py : The role of View in Plants folder describes how users can register and see plant information, diagnose their plants, edit palnt information(like the picture of the plant), and delete plants.

* Plant List : A user can use a GET request to get a list of the plants they have planted. Since there are many different users of the application, the plants stored in the database on the deployed server will contain all of the users' plants. To retrieve only a specific user's plants, we use JWT tokens. We use a filter to retrieve only the plants whose user is the same as the user in the plants database from the list of all plants in the GET request. In the list of plants, the user should be able to see if each plant has been harvested and at what growth level. This is because we need to display the list separately for harvested plants and for plants that are still growing. To do this, the determine is harvested and calculate growth level functions are used to retrieve the harvest status and growth level along with the plant information stored in the database. In addition, the number of plants that can be planted is limited by the size of the garden, so the size of the TIIUN used by the user is set to be retrieved and returned.

* Plant Type : This is the part about the plants that users can plant. This is added by the administrator directly on the backend via a POST request, using the IsAdminUser permission. It is distinguished by a JWT token to verify that the person sending the POST request is an administrator. You can add asset details by entering any data required by the model in the request body and submitting it. Currently, 4 plants have been added to the backend: strawberries, potatoes, tomatoes, and corn.

* Create Plant : Users can add assets via a POST request. To use this feature, the user must be logged in, so we set up a JWT token for verification. To add a plant, the user must include all the data values set in the plant model in the request body. The user who plants the plant is automatically registered as the user who sends the POST request when the plant is registered.

* Plant Detail : The user can check the details of the plants they have planted, i.e. the registered plants, by making a GET request. In this case, since we need to retrieve information about a specific plant, we can identify the plant by entering the plant ID in the query string. Also, to determine if the plant is registered by the user, the JWT token is used to determine if the registered user and the user currently sending the request are the same. If they are not the same user, we return the message 'You do not have access to this asset'. The plant

details include details related to the plant's type, which are added by the administrator in the backend. The Plant Type ID can be used to retrieve information about the type of plant the user has planted. In addition, to determine the growth level of the plant, the Calculate Growth Level function is used to determine the germination period, growth period, and harvest period. For all serialized information, the user ID is not needed in the detail record because the user planted the plant. We also exclude the harvest date because we want to ensure that the plants whose details are available are only those that have not been harvested. If a plant is close to harvest time, the user can harvest it using a POST request. This request adds the harvest date to the 'harvested at' data for that plant. Users can also change their plant information with a PATCH request. However, we have limited the changes that can be made directly to the name of the plant and the photo of the plant. The rest of the information is fixed and cannot be changed. You can delete a plant with a DELETE request. If you have diagnosed a plant, you can delete it if it is sick. This can be done with the appropriate request. When you delete a plant, the related information that is linked to the plant by a foreign key is also deleted.

* Plant Disease Type : Plant disease information is added by the administrator via a POST request. To do this, we use IsAdminUser in the permissions and use the JWT token to determine if the person who made the POST request is an administrator. Enter the record in the Plant Disease Type model in the body and send it to add the disease information.

* Plant Disease Record : The diagnosis of plant diseases is done using the leaf parts of the plant. The diagnosis of the plant is done by the AI, which will be explained in detail in the AI section below. On the backend, application can determine whether a plant has a disease or not through a function that calls the AI. The backend first uses JWT tokens to determine if the user has properly registered with the application. Then it checks if the plant user wants to diagnose is the one he registered. Only plants registered by the user can be diagnosed. In the backend, the storage capacity is reduced by storing the URL processed by the frontend in the database instead of directly storing the photos taken by the user for disease identification. Users can verify the diagnosis using AI by making a POST request with the plant ID and the URL of the photo. Users can also retrieve the diagnosis results for each plant they grow by making a GET request. The user can know the growth level and disease name of the plant through the request. When storing plant disease records in the database, the user can also check the name of the plant set by the user in connection with the plant database through the plant ID stored together.

* Plant Disease Record Detail : If the user wants to retrieve specific diagnostic information from a list of diagnosed plants, he can do so using a GET request. Since he's looking for one of several diagnostic results, he'll want to include the Disease ID in the query string to distinguish them. The JWT token is used to verify that he is an authorized user with access to the diagnostic results for this plant. While the disease information shows only the name, symptoms, and cause of the disease, the diagnostic details should also include a brief description of the diseased plant. Therefore, we load the Plant database information and the associated Plant Type database to extract the necessary information and return it to the user. The information the user should be able to see on the Plant Diagnosis Details page includes not only information about the disease, but also the plant variety and the user-assigned plant name. Other growth levels should also be included. For the AI diagnosed results, we set up the get predicted name function to return the exact name of the disease.

E. Module 3 - Machine Learning

1) Purpose
   When designing BloomMate, our team centered around "whether or not the plant's owner is in the SmartCottage." For each situation, the core functionality is as follows
   a) The owner is not in the SmartCottage
      - use generative AI to talk to the plant.
   b) The owner is in the SmartCottage
      - take a picture of the plant and **use AI to diagnose it**.

   This Machine Learning Repository is for the AI used in the second situation. Train the AI using Custom Dataset and convert it to tensorflow lite so that it can be used in real projects.
2) Functionality
   Initially, acquaint with custom data by training a ResNet-50 model. Subsequently, convert the trained model to

TensorFlow Lite. The resulting converted model is then used in BloomMate-BE.

3) Location of source code : https://github.com/BloomMate/BloomMate-ML
4) Class Components
   - Dataset: As we'll see later, Resnet-50 is a model that has been pre-trained on multiple datasets. However, BloomMate needs to diagnose plant diseases for four different crops: corn, potatoes, tomatoes, and strawberries. Unfortunately, Resnet-50's image classification model was not pre-trained with the appropriate data for this specific situation. Therefore, we had to find a custom dataset. Fortunately, we found the best one for our needs. However, we encountered an issue with over-fitting during the initial training. To address this, we limited the data for all classes to 1000 samples. Please refer to the table below for the dataset lists.

tomato	Tomato Early blight Tomato Late blight Tomato Target Spot Tomato Yellow Leaf Curl Virus Tomato healthy
potato	Potato Late blight Potato Early blight Potato healthy
corn	Corn Common rust Corn Gray leaf spot Corn Northern Leaf Blight Corn healthy
strawberry	Strawberry healthy Strawberry Leaf scorch

   - resnet-all-datasets.ipynb : At first, we followed tensorflow's image classification guide, which was great for distinguishing between completely different shapes of leaves (e.g., tomatoes and corn), but once we started to have more classes that needed to be distinguished, such as diseased or not diseased, it didn't work as well. So we started looking for other proven models, and chose Resnet-50. ResNet-50 is a convolutional neural network (CNN) architecture commonly used for building deep neural networks. It is part of the ResNet (Residual Network) series developed by Microsoft Research, known for its outstanding performance in image recognition and classification tasks.

     Key features of ResNet-50:

     a) Depth: ResNet-50 consists of 50 layers, representing a deep neural network. This depth addresses the vanishing gradient problem during training and allows for high-level abstraction.

     b) Residual connections: ResNet introduces residual connections, enabling each block to learn the residual (difference) between input and output. This facilitates the flow of information without loss, easing training and improving performance.

     c) Transfer learning: Pre-trained on the large-scale ImageNet dataset, ResNet-50 serves as a pre-trained model. This allows for transfer learning, applying the knowledge gained from ImageNet to various computer vision tasks.

     d) Batch normalization: ResNet incorporates batch normalization at each layer to stabilize training and accelerate learning.

     Resnet-50 models deployed in tensorflow or pytorch are pre-trained. However, as mentioned above, the images we wanted were not pre-trained and we had to train the model on our own dataset and fine-tune it. After fine-tuning, I created a model and converted it into a tflite file. Finally, I verified that the converted model performs similarly to the original model.

   - resnet-tflite-test.ipynb : Actually, this code is a test implementation to make the TFLite model work on the backend. The main difference compared to "all-datasets" is that it performs image classification on photos obtained through a URL.

VII. SOFTWARE INSTALLATION GUIDE

A. Android Installation

To install the Android application, follow these simple steps:

1) Go to the Google Drive link provided below:
   Link: Google Drive Link to install APK file

Please take note of the following:

1) Starting from September 2024, the application may not function properly. This is due to the possibility of the server operation being discontinued when the free plan of AWS EC2 ends.

2) Since the application was not downloaded from the Play Store, you may encounter a message indicating a potential presence of malicious software. Rest assured that BloomMate does not contain any malicious code. You can safely ignore the message and proceed with the installation.

B. IOS Installation

For iOS, the only way to install the app is through the official App Store. Therefore, you need to install it directly via source code. Please note that the following steps can only be performed on macOS, as it is not possible to install iOS on Windows without xCode.
Follow these steps:

1) Configure iOS by referring to the official react-native website's iOS setup page.
2) Download the source code from the BloomMate-FE GitHub repository.
3) Open the terminal and enter the command "yarn" to download the required libraries.
4) Create a .env file in the root directory. Contact our team for the contents to include in the .env file, and we will respond within 3 business days.
5) Enter "yarn start" in the terminal and use "yarn ios" to install BloomMate on your iOS device.

Similar to Android, there is a possibility that the application may not function properly after September 2024.

VIII. USE CASES

 # Primary Tab - Plant List

After logging in, users can easily check and manage their plant list through the Primary Plant List tab. This tab includes two main sections: 'Growing' and 'Harvested'.

- At the top of the plant list screen, there are two tabs, 'Harvesting' and 'Harvested'. These tabs help users easily understand the current status of their plants. The 'Harvesting' tab displays plants that are in the 'Germination', 'Growth', or 'Harvest' periods, while the 'Harvested' tab displays the plants that the user has already harvested.

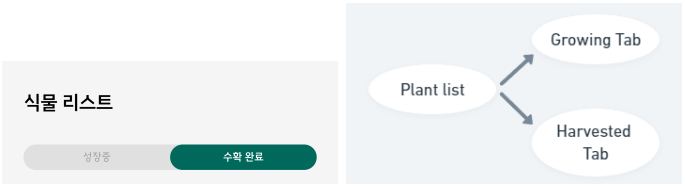


Fig. 40. Plant List Tab

1) Plant Add

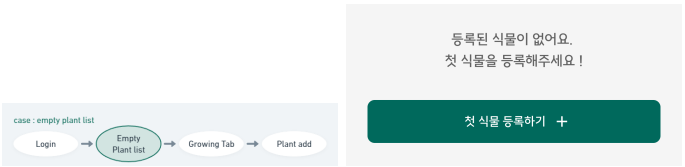


Fig. 41. Empty Plant List

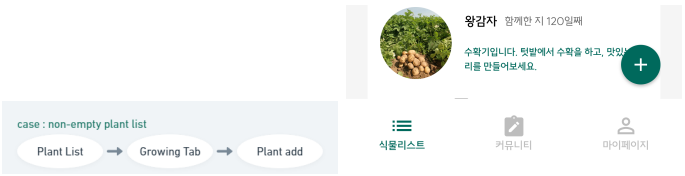


Fig. 42. Plant List

- Plant addition mainly occurs in two scenarios. The first scenario is when a user signs up and logs in for the first time. In this case, the user can view the initial state of the plant list screen, and add a plant using the 'First Plant Registration' button.The second scenario is when the user has already completed the membership registration and registered one or more plants. In this case, the user can add a new plant through the 'Add Plant' floating button located at the bottom right of the plant list screen.
- When adding a plant, the user must enter information such as photo, species, nickname, and planting date. This information helps users to manage their plants more effectively.

2) Plant Details

Fig. 43. Plant Details

- Users can check the detailed information of each plant by clicking the 'Plant Detailed Information' button at the bottom of each plant component. In this screen, users can update the plant's photo and nickname, check basic information about the plant's species and its current growth status.In the growth information section, users can visually check which growth period the plant is currently in and how far that period has progressed through a Progress bar. Also, users can more easily manage the plant's status through the Primary color phrase that changes depending on the growth degree, and the 'AI Diagnosis' or 'Harvest' button at the bottom.

3) Plant Update

Fig. 44. Use case: Plant Update

- Users can update the plant's photo and nickname on the plant's detailed information screen. Users click the 'Update' text button located at the top right of

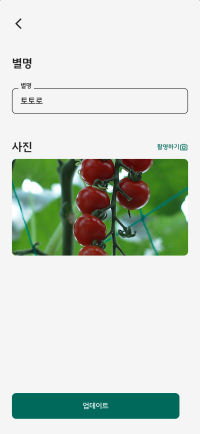


Fig. 45. Plant Update

the photo. They are then taken to a page where they can update the nickname and photo. If the user does not want to make a change, they can return to the plant's detailed information screen by clicking the back arrow icon at the top left.

- To update the nickname or photo, the user modifies the desired item and then clicks the 'Update' button at the bottom. The user then returns to the plant's detailed information screen, where they can see that the changes have been reflected.

4) AI Diagnosis

Fig. 46. Use case: AI Diagnosis

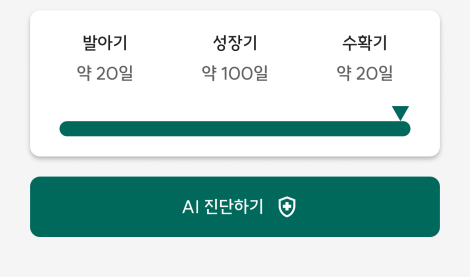


Fig. 47. AI Diagnosis Button

- The button at the bottom of the plant's detailed information screen is displayed as the 'AI Diagnosis' button when the plant is not in the harvest period.
- When the user clicks the diagnosis button, they can check the diagnosis record of the plant and move to the screen where they can diagnose it. Users can get information about the current status of the plant through chatting and growth information.

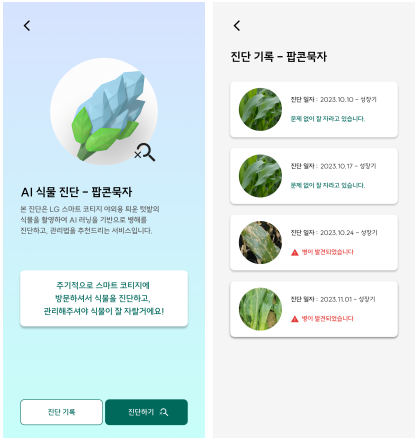


Fig. 48. Diagnose Intro & History

Based on this information, users can visit the LG Smart Cottage and diagnose the status of the plant planted in the Tiiun garden through AI diagnosis. The diagnosis-related function screen includes the plant's nickname so the user knows they are only dealing with that plant.

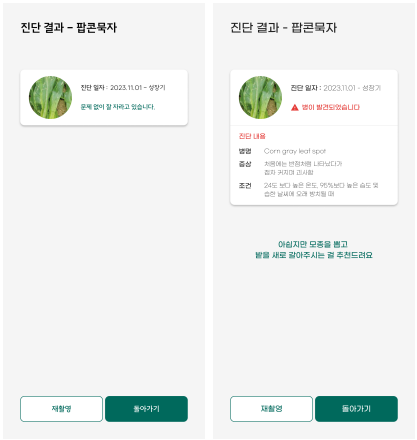


Fig. 49. Diagnose Result Example

- When the user clicks the diagnosis button, they can take a photo directly or select a photo from the gallery for diagnosis. The photo selected by the user is analyzed based on AI machine learning, and this is used to diagnose the status of the plant. Depending on the diagnosis results, the user can get information on whether the plant is healthy or diseased.
- If a disease is found in the plant as a result of the AI diagnosis, the user receives detailed information about the disease diagnosis. This information includes the type of disease, cause, and treatment methods. In addition, the user is recommended to replace the seedlings. For this case, a one-click purchase button is provided on the screen to order

new seeds for the same species.

- If the user clicks this button, a dialog appears notifying the user that the existing plant will be deleted. If the user clicks the 'Confirm' button in the dialog, new seeds of the same species are automatically ordered. After the order is completed, the user returns to the plant list screen and can confirm that the original plant has been deleted.

Fig. 50. Diagnose Result - Oneclick Purchase

5) Harvest

Fig. 51. Use case: Harvest

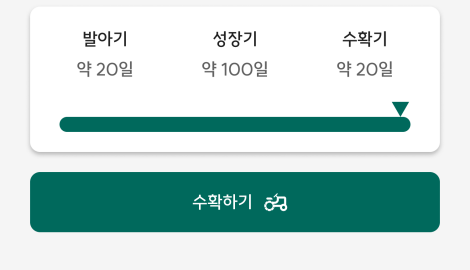


Fig. 52. Harvesting Button

- The button at the bottom of the plant's detailed information screen is displayed as the 'Harvest' button when the plant is in the harvest period. When the user clicks this button, the plant is harvested, and afterwards, the user can confirm that the plant has been deleted from the plant list screen. The harvested plant moves to the 'Harvested' tab in the top tab of the plant list screen.

6) Chatting with GPT

Fig. 53. Use case: Chatting

- Today's Chat in the plant list tab:
  Users can move to the chat screen by clicking the 'Chat' button on the plant component. When the user clicks the 'Check Today's Report' button in this screen, BloomMate (Chat GPT-based chatbot) analyzes the current weather, temperature, humidity, etc. based on the detailed information of the plant as a plant and the user's location information, and tells the state of the plant. In addition, BloomMate provides information about the soil condition through Tiiun. If the soil condition is poor, the user can purchase Tiiun-specific fertilizer through the 'One-click' button.

Fig. 54. Chat - Today's report

- Previous Chat :
  Users can check previous chat records. When the user clicks the 'calendar-month' icon labeled 'Previous Report' in the floating button on the right bottom of the screen, a calendar modal is displayed. Through this calendar, users can check the chat of the previous date. If there is no chat record on the date selected by the user, they can see the message, "There is no conversation on the date you selected!"

Fig. 55. Chat - Previous Chat

- Return to Today's Chat :
  Users can return to today's chat window by clicking the 'calendar-today' icon labeled 'Today's Report' in the floating button.
  In this screen, users can check today's report and today's conversations. Users can freely chat with the plant. For example, users can send messages like "How do you feel today", "I miss you", etc., and the plant responds appropriately to these.

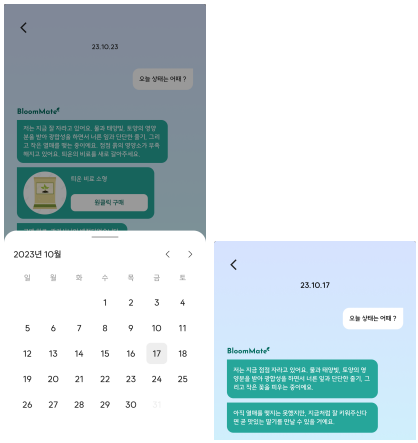


Fig. 56. Chat - Return to Today's Chat

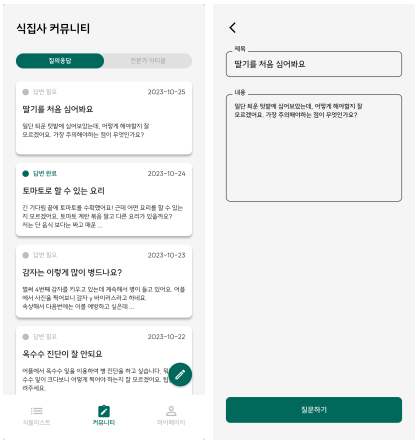


Fig. 59. create Q & A

 # Primary Tab - Community

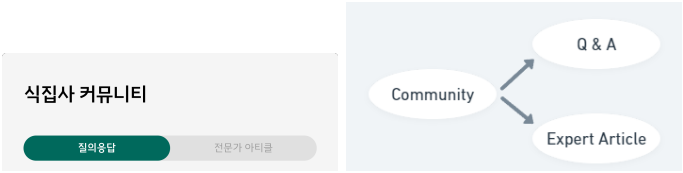


Fig. 57. Community Tab

After login, users can obtain various information about plants through the Primary Community tab. This tab includes two main sections: 'Q & A' and 'Expert Article'.

7) Q & A
   - In the Q & A tab, users can ask various questions. Users can write a new question through the floating button, which includes a pencil-shaped icon located at the bottom right.

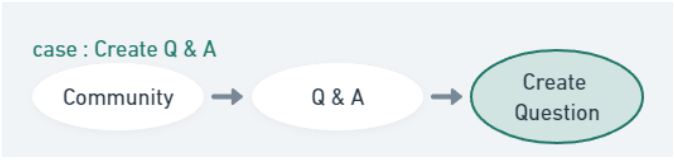


Fig. 58. Use case: Create Q & A

- Based on the information obtained from chat or diagnosis process, users can ask general questions about the plant or methods of caring for Tiiun.When the user clicks the floating button, they are directed to a text input form screen to write their question. BloomMate provides an answer for each question as soon as possible, and the answer status is displayed at the top left of each question component. Through this, users can easily check the status of their question.

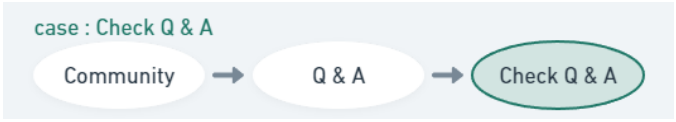


Fig. 60. Use case: Check Q & A

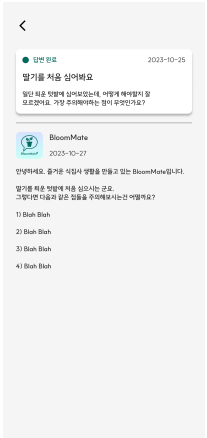


Fig. 61. Question Details

8) Expert's Article
   - In the Expert Article tab, users can read expert articles in a sequential episode format. Users can read articles containing a variety of information,

including detailed information about the breeds that can be grown in Tiiun, cultivation and management methods, and ways to utilize the crops.

Fig. 62. Use case: article

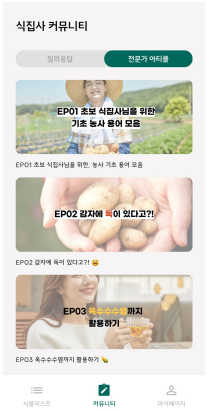


Fig. 63. Expert's Article

 # Primary Tab - Mypage

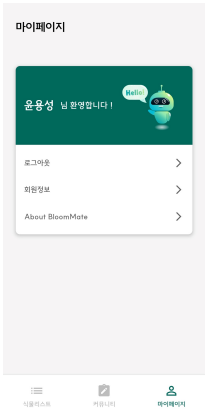


Fig. 64. My page

9) Logout

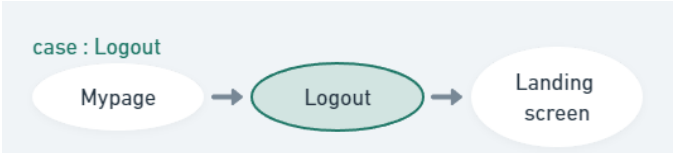


Fig. 65. Use case: log-out

Users who want to logout can click the 'Logout' button. When this button is clicked, the user's authentication information is removed from the system, and the user is automatically redirected to the landing page where they can login and register.

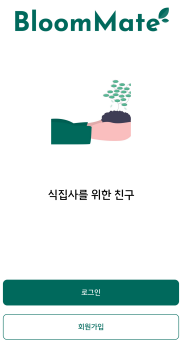


Fig. 66. After Logout - Landing Page

10) User Info

Fig. 67. Use case: user info

Fig. 68. Info

Users can check the member information entered through the registration process. When the user clicks 'User Info', they are directed to a page where they can check the user's name, ID, Tiiun product key, Tiiun size, and address information. If the user wishes, they can return to the Mypage tab at any time by clicking

the back arrow icon at the top left.

11) About BloomMate

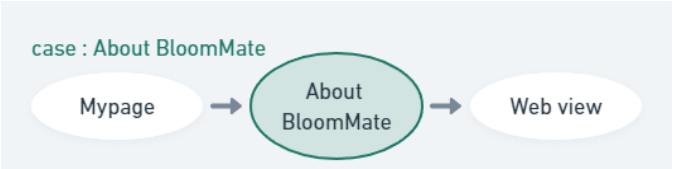


Fig. 69. Use case: about BloomMate

Users can click 'About BloomMate' to check information about the team that developed BloomMate. On this page, users can refer to the background of BloomMate app development, demo videos, and related links about the development process. Also, users can check the technology stack used, core features, and information about the team members.

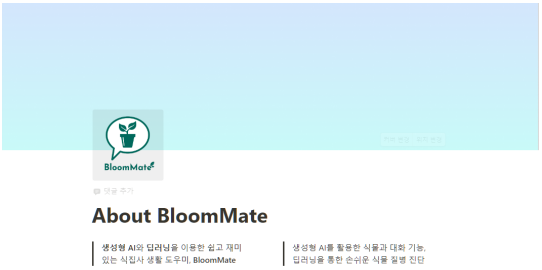


Fig. 70. About BloomMate

IX. DISCUSSION

A. Things to Improvement

BloomMate had several shortcomings. The most significant one is that it violates various legal aspects, making it ineligible for official distribution. The issue of personal data protection is particularly problematic. To address this, terms and conditions should have been established, and the back-end team should have developed a system to promptly dispose of personal information. However, this was not accomplished. The second shortcoming is the lack of test code. We conducted QA solely through manual testing without properly writing test code. As a result, a bug that caused plant duplication was not fixed three days before the final application submission. Lastly, the AI aspect was lacking. In the initial design of BloomMate, we intended to perform real-time object detection using AI. This would have allowed users to observe plant information in real-time even when they are not present in the smart cottage. However, due to insufficient dataset and a lack of human resources to develop and deploy code for the new server, we compromised by using image classification.

B. Non-Technical Difficulties While Making Software

The most challenging aspect was ensuring a fair distribution of tasks during the software development process. While we made an effort to distribute the work fairly, we did not take into account variations in experience and skill level. Consequently, some individuals had to wait for others to complete their tasks, while others faced excessive workloads and extreme stress. Thankfully, we were able to address these issues through effective communication and weekly meetings.

C. Conclusion

Although there were some mentioned issues, BloomMate is a well-made software. All team members have fulfilled their roles, and we have successfully designed and implemented features that are now ready for commercialization. There are no bugs, and users will greatly benefit from using BloomMate for gardening with Tiiun. The creation of BloomMate has provided us with an opportunity to think more concretely about various dreams, such as those of developers and planners.

REFERENCES

[1] J. Young-seon, H. So-young, and P. Y. Jin, "Effects of companion plants on mental health of elderly women living alone in the city," *Journal of the Korea Institute of Garden Design*, vol. 6, no. 3, pp. 218–227, 2020.
[2] H. F. R. Institute. (2020, May) Changes in consumer behavior brought by covid-19. [Online]. Available: http://www.hanaif.re.kr/boardDetail.do?hmpeSeqNo=34411
[3] C. Da-hyun. (2021, April) [the trend is indoor plants] part 1: Perfect for relieving pandemic blues and stabilizing the emotions of the elderly. [Online]. Available: https://www.ajunews.com/view/20210401172057744
[4] SPC-magazine. (2023, June) [trend] the trend of communicating with 'leaves' in companion plants.
[5] K. Yangjin. (2020, November) Urban farmers in seoul increased 14 times in 8 years, agricultural space expanded 7 times. [Online]. Available: https://www.hani.co.kr/arti/area/capital/972100.html
[6] C. Dong-soo. (2021, November) Top housing preferences after retirement: 'detached houses' and 'townhouses' lead the way. [Online]. Available: https://www.donga.com/news/Economy/article/all/20211108/110143322/1
[7] Y. Kwang-won. (2023, May) Seoul enacts 'pet plant industry promotion and support ordinance'. [Online]. Available: https://www.mediapen.com/news/view/818763
[8] LG Electronics. (n.d.) Lg brand guidelines. [Online]. Available: https://www.lge.co.kr/company/info/ci
[9] "Typescript," https://www.typescriptlang.org/, 2023.
[10] V. Authors. (2016) Yarn: A new package manager for javascript. Accessed on 2023-10-16. [Online]. Available: https://engineering.fb.com/2016/10/11/web/yarn-a-new-package-manager-for-javascript/
[11] E. Elliott. (2023) Streamline code reviews with eslint prettier. Accessed on 2023-10-16. [Online]. Available: https://medium.com/javascript-scene/streamline-code-reviews-with-eslint-prettier-6fb817a6b51d
[12] (2023) React native. Accessed on 2023-10-16. [Online]. Available: https://reactnative.dev/
[13] Tanstack. (2023) tanstack/query. Accessed on 2023-10-16. [Online]. Available: https://tanstack.com/query/v3/
[14] (2023) Recoil. Accessed on 2023-10-16. [Online]. Available: https://recoiljs.org/
[15] (2023) Python quick reference. Accessed on 2023-10-16. [Online]. Available: https://library.gabia.com/contents/9256/
[16] (2023) Django tutorial - django girls. Accessed on 2023-10-16. [Online]. Available: https://tutorial.djangogirls.org/ko/django/

[17] AboutJoo. (2022) Django 50 questions 50 answers (5) - velog. Accessed on 2023-10-16. [Online]. Available: https://velog.io/@aboutjoo/Django-50%EB%AC%B8-50%EB%8B%B5-5
[18] A. W. Services. (2023) Amazon ec2 documentation. Accessed on 2023-10-16. [Online]. Available: https://docs.aws.amazon.com
[19] WikiDocs. (2023) Python gui programming (pyqt) - wikidocs. Accessed on 2023-10-16. [Online]. Available: https://wikidocs.net/195613
[20] Oracle. (2022) What is tensorflow? - oracle developer. Accessed on 2023-10-16. [Online]. Available: https://developer.oracle.com/ko/learn/technical-articles/1481879245868-120-what-is-tensorflow
[21] G. D. Korea. (2014) This is material design - google developers korea blog. Accessed on 2023-10-16. [Online]. Available: https://developers-kr.googleblog.com/2014/07/this-is-material-design.html
[22] hxyxneee. (2021) What is git? - velog. [Online]. Available: https://velog.io/@hxyxneee/git%EC%9D%B4%EB%9E%80-%EB%AC%B4%EC%97%87%