NotiSKKU
Notification App for Targeted Major and Topic Information

Changyeon Seo, Dongheon Kang, Sucheol Lee, Taewon Kim, and Yesung Jo

Sungkyunkwan Unisversity, 2066,
Seobu-ro, Jangan-su, Suwon-si, Gyeonggi-do, Repblic of Korea
Capstone Design Project

Abstract. This paper introduces notiSKKU, an notification application designed to address the challenges faced by university students. The application provides personalized notifications and alarms, delivering timely updates, news, and events related to individual majors and areas of interest. The key achievements of our solution include the development of a comprehensive platform that integrates information from various sources, a user-centric customization feature that delivers tailored notifications, and the provision of timely and efficient information access to help students make informed decisions and avoid missing out on important opportunities.

Keywords: Notice · Provide information · Information integration

1 Introduction

1.1 Motivation

Our motivation stems from the realization that students often struggle with the overwhelming amount of information scattered across multiple sources. This leads to stress, confusion, and inefficiency in the search process. We firmly believe that students deserve a better way to navigate this information landscape. To achieve this, we developed a comprehensive notifying program that seamlessly integrates data from various sources. Our solution is designed with a user-centric approach, keeping in mind the specific needs and preferences of students. Through intuitive design and easy navigation, we aim to create a tool that feels intuitive and familiar to students, ensuring a positive user experience.

1.2 Goal

Unified platform Through our research and personal experiences, we have recognized the need for a more integrated and user-friendly solution. To address this issue, our proposed solution focuses on developing a sophisticated notifying program that consolidates information from various sources. Through the integration of data from multiple sources, including university websites and departmental guidelines, we aim to create a unified platform that presents students with clear and up-to-date information.

Personalized and Customized Service In addition to our proposed solution, we aimed to provide each user with a personalized and customized service. By defining their preferred subjects, majors, interests, or activities, users can receive notifications and alarms specifically related to their chosen fields. This customization feature ensures that users only receive relevant notices, eliminating the inconvenience of having to sift through information that is not of interest to them.

Stay Up-to-date Our solution enables users to stay up-to-date with the latest announcements and deadlines. Through timely alarms, users can promptly access important notices, preventing any unfortunate circumstances where they miss out on critical information or deadlines.

1.3 Technique and Approach

Crawling and MongoDB Crawling is a fundamental technique used in our project to gather information from various websites. The crawling process involves systematically browsing web pages, extracting relevant data, and storing it in a structured manner.

To accomplish the crawling task, we employed web scraping techniques and utilized Python libraries. Our crawler navigates through the website, visiting each relevant page and extracting the required information. This crawled data is then transformed and formatted into a consistent structure for storage and further processing.

To efficiently manage the extracted data, we utilized MongoDB, a NoSQL database. MongoDB's flexible document-based model allows us to store the crawled information in a schema-less manner, making it suitable for storing diverse types of data obtained from different websites.

Push Notification To implement push notifications in our project, we leveraged FCM. It provides a reliable infrastructure for sending push notifications to mobile devices.
First, We obtained user's device token. This token serves as a unique identifier for the user's device. When a new announcement is registered, we trigger the push notification process. This payload is then sent to FCM, along with the device tokens of the subscribed users. FCM handles the delivery of the push notification to the target devices. Upon receiving the push notification, the user's device displays the notification in the system tray.

Confirmation Code To facilitate the user authentication process during registration, we implemented a method that involves sending an email verification code and validating it as an authentication token. Technology utilized for email verification and authentication is commonly known as email verification or confirmation. When a user registers on our platform, they provide their email address.

To initiate the email verification process, we generate a unique verification code specific to that user. The verification code serves as a token to validate the user's email ownership.

Next, we utilize SMTP, a widely-used protocol for sending email, to deliver the verification code to the user's provided email address. SMTP enables our application to communicate with the mail server responsible for delivering the email. We construct an email template that includes the verification code and necessary instructions for the user.

Once the user submits the verification code through our platform, we compare it with the generated code to authenticate their email address. If the codes match, the user's email is considered verified, and they gain access to the full functionality of our platform. By implementing email verification and authentication, we enhance the security and validity of user registrations, ensuring that only users with verified email addresses can access our platform's features and services.

1.4 Achievement

Our approach have resulted in significant achievements in addressing the challenges students face.Followings are some notable accomplishments.

Development of a Comprehensive Solution We have successfully designed and developed a sophisticated notifying program that integrates information from various sources. This program serves as a centralized platform, streamlining the process of accessing and comprehending graduation requirements.

User-Centric Customization Our solution offers a personalized and customized experience for each user. By allowing students to set their areas of interest, we have created a system that delivers tailored notifications and alarms, ensuring they receive only the most relevant information.

Timely and Efficient Information Access By delivering notifications and alarms, our solution enables students to stay up-to-date with the latest announcements and deadlines. This ensures that students have the information they need in a timely manner, helping them make informed decisions and avoid missing out on important opportunities.

2 Design

2.1 System Arichitecture

a. Crawler Server: The crawler server is responsible for periodically crawling a designated data source. In this case, the server will crawl the data source every hour, specifically from 9 AM to 6 PM. The purpose of this server is to fetch

the latest data updates available. The scheduler used for this task is Cron. The crawler is implemented using Scrapy, which will make it easier to expand to new data sources in the future.

b. Notice Database (Notice DB): Figure 1 shows the total database system. Upon retrieving the data from the crawler server, it is stored in the Notice DB. This database acts as a repository for the crawled data and provides a centralized storage system accessible to other components of the architecture. Refer to the database diagram below.

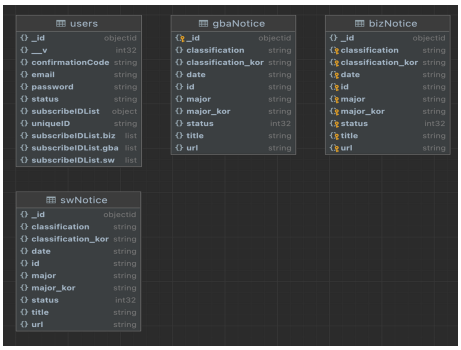

Fig. 1. System Architecture

c. API Server: The API server utilizes the data stored in the Notice DB to process and disseminate notifications to subscribed users. This server continuously monitors the Notice DB for new notifications. RESTful API was chosen for the app architecture due to its simplicity, platform independence, stateless communication, scalability, performance, standardization, and support for multiple client types. With REST, the development process is streamlined, and the API can be accessed by clients regardless of their technology stack.

d. Firebase: Firebase is the chosen notification service that enables the API server to send notifications to app users in real-time. Firebase provides a reliable and scalable platform for handling push notifications to a large number of users across different devices.

e. Client Flutter was chosen as the framework for the app development due to it being a cross-platform framework that allows for the creation of native-like applications for both iOS and Android platforms from a single codebase, reducing development time and effort.
Figure 2 shows our whole system architecture, where we have designed an overall architecture that incorporates various components to efficiently manage and deliver notice information from multiple university websites to the users.

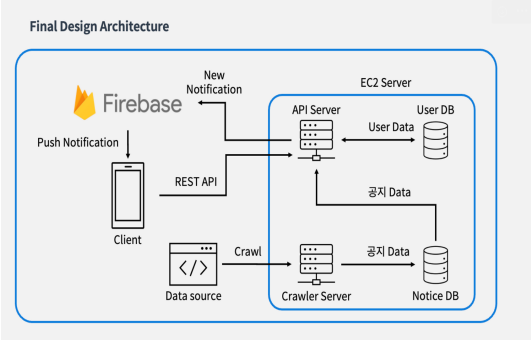

Fig. 2. System Architecture

2.2 Client system design

Client architecture NotiSKKU client app architecture borrows some ideas

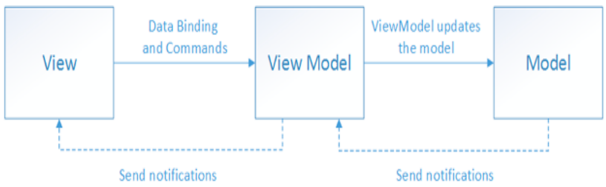

Fig. 3. Client architecture

from MVVM design pattern. In Figure 3, MVVM is a design pattern that separates an application into Model, View, and ViewModel, enabling a clear separation of concerns and enhancing code maintainability. It facilitates data binding between the View and ViewModel, promoting efficient UI updates and easy transfer of user input. MVVM improves the testability and reusability of the ViewModel, providing a flexible and scalable architecture for developing robust and interactive applications.

Declarative UI Declarative UI paradigm is an approach to building user interfaces where developers define the desired outcome or final state of the UI, rather than manually specifying each step or update. In a declarative UI, developers describe the UI structure and behavior using a markup language or a specialized syntax, and the underlying framework or library takes care of updating the UI to match the desired state. This paradigm allows for a more intuitive and expressive way of expressing UI logic, enabling easier understanding, maintenance, and reuse of code. Declarative UI helps to separate the UI description from the

imperative logic, resulting in cleaner and more concise code that is focused on the desired outcome rather than the step-by-step instructions for achieving it.

2.3 Core Skill

Frontend React Native is a powerful framework for cross-platform mobile app development using JavaScript. It provides a native-like experience on iOS and Android platforms, utilizing a single codebase. Its component-based architecture promotes code reusability and easier maintenance. By leveraging native components and APIs, React Native delivers high-performance applications with smooth animations and fast load times. The framework benefits from a large ecosystem of libraries and tools, allowing for easy integration of additional features. Hot reloading enables real-time code changes during development, enhancing the iteration process. Overall, React Native empowers efficient and visually appealing mobile app development, providing a seamless user experience on multiple platforms.

Backend In the backend, our project utilizes key technologies and skills. Node.js with Express serves as a scalable and efficient platform for server-side development, handling routing and middleware. MongoDB, a NoSQL database, offers flexibility and scalability for managing large volumes of data, facilitating seamless integration and retrieval. Scrapy, a Python-based web scraping framework, enables efficient data extraction from various websites. For development and push notifications, the options include Amazon EC2, providing scalable cloud computing resources, and Firebase Cloud Messaging, ensuring cross-platform delivery of push notifications to iOS and Android devices.

2.4 reasoning

Frontend We chose React Native as our primary technology due to its cross-platform development capabilities, which allowed us to target both iOS and Android platforms with a single codebase. This saved time and effort while ensuring a consistent user experience across devices. Additionally, React Native provided a native-like experience by leveraging native APIs, resulting in high-performance applications with responsive interfaces. Its strong developer community, hot reloading feature, and alignment with modern web development practices further solidified our decision to use React Native as our primary technology.

Backend Our project focused on developing an app service that collects and manages notices from university websites, providing timely notifications to users. To achieve this, we made specific design choices regarding the technologies we used. We opted for Node.js with Express as our backend technology for its scalability and performance. MongoDB was chosen as the database for its flexibility in handling unstructured data. Scrapy, a powerful web scraping framework, was

utilized for collecting notice data. Depending on our needs, we had the option of using either Amazon EC2 or Firebase Cloud Messaging for hosting and push notifications. These design choices aimed to ensure scalability, performance, flexibility, and platform compatibility, ultimately creating an effective app service for notice management and user notification.

2.5 Challenges

Frontend Our project involved developing a notice management app that collects and delivers notifications from university websites. The challenge was to create a mobile app that works smoothly on both iOS and Android platforms. To overcome this, we opted for React Native, enabling code reuse and effortless deployment across platforms. React Native's integration of native components ensured a native-like experience with optimal performance. The hot reloading feature allowed real-time code changes and quick debugging. Overall, React Native empowered us to build a high-quality, cross-platform app for effective notice management and timely user notifications.
Our project involved creating an app service that gathers and manages notices from multiple university websites to provide timely notifications to users. We overcame challenges such as efficient data scraping, real-time push notifications, and scalability. To address these challenges, we utilized Node.js with Express, MongoDB, Scrapy, and either Amazon EC2 or Firebase Cloud Messaging. Scrapy enabled us to navigate websites and extract notice data, which we stored in MongoDB for seamless retrieval. Integration with Amazon EC2 or Firebase Cloud Messaging facilitated real-time push notifications on iOS and Android. With the combined power of these technologies, we successfully collected and managed notice data, delivered timely notifications, and ensured scalability and performance in our app service.

3 Implementation

3.1 BackEnd

Step 1: Data Crawling
The crawler server initiates the data crawling process based on a predefined schedule. It periodically fetches data from the specified data source, ensuring that the latest updates are captured.

Step 2: Storing Data in Notice DB
Upon successful crawling, the retrieved data is stored in the Notice DB. The Notice DB acts as a persistent storage solution, housing all the relevant information for subsequent processing.

Step 3: API Server Operation

The API server continuously monitors the Notice DB for any new notifications. It compares the stored data with previous records to identify new notifications. Once new notifications are identified, the API server proceeds to the next step.

Step 4: Notification Distribution via Firebase
Using Firebase as the notification service, the API server pushes the notifications to the users who have subscribed to specific subjects or categories. The notifications are sent in real-time to ensure timely delivery and user engagement.

The described architecture design provides a robust framework for delivering real-time notifications to users based on their subscribed subjects of interest. The combination of the crawler server, Notice DB, API server, and Firebase notification service enables efficient data retrieval, storage, and distribution processes. This architecture ensures that users are promptly notified of new updates, enhancing their overall app experience and engagement.

3.2 Sign-up Flow

Intro page The first picture in Figure 4 shows our app intro screen. The logo of

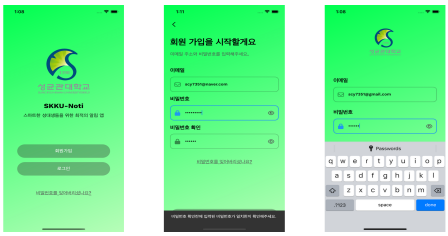

Fig. 4. intro page, sign up and sign in page screenshot

Sungkyunkwan University is engraved in the center of the top, and a brief guide on the information available from the app is provided with titles and sub-titles. On this screen, users can click the button that connects to three functions: sign up, log in, and find a password.

Sign-up page The second picture in Figure 4 shows the membership registration screen. Users must enter email and password information to sign up, and passwords must go through one more final verification process for security and stability. Each text field has a text validation rule, and if a text is entered that violates it, an error snack bar is printed as shown in the screenshot.

Sign-in and Subscription Flow

Sign-in page The third picture in Figure 4 shows the login page, where it has an email and password entry window. The text entered in each input window goes through the following verification procedure.

1. E-mail
   E-mail Check if it has the form of an e-mail format (***@***.com). Instead of ".com", it can be ".net", etc. Regular expressions were used for this validation.
2. Password
   Password I used a password of more than 6 characters for security. All passwords stored on the server are encrypted.

Notification list page The first picture in Figure 5 shows the Notification list

Fig. 5. notification list, setting, detail page screenshot

page, where users can view a list of all announcements for items that you have subscribed to. Depending on the major, the icon on the left changes, and the name of the major and the name for the announcement classification are shown in the subtitle. If user press the "More" button, you can go to the announcement details screen.

Notification setting page The second picture in Figure 5 shows the setting page, where users can view membership information and all items currently available for subscription. You can turn on and off notifications and subscriptions in the form of a toggle button.

Notification detail page (Webview) The third picture in Figure 5 shows Displays the notification screen in the web view. The in-app renders the announcement detail screen web URL contained in the crawled data.

3.3 Notification

There are two main tasks that the client has done to send push notifications. One is to add the FCM framework to the app codebase and to add the push

Fig. 6. notification screenshot

token and remote message reception monitoring logic to send the push through the FCM platform described above. The second task is to have users display push notifications as soon as they receive a remote message. This work used the "flutter_local_notification" package provided by the flutter framework.

FCM settings in client
Function that works when the app first starts running. Settings related to push notifications provided by Firebase are made. iOS and Android have completely different push notification systems, so you have to create and manage Notification settings. This operation is partially supported by the local_notifications library on the flutter.
Code to load the token required to send a push through the FCM server. This fcmToken is sent to the server using an internal api (user/registerToken), and a push notification can be sent to the user immediately when a new announcement is registered through a token stored on the server. Tokens are renewed every specific cycle when you enter the app. The NotiSKKU app sends tokens at each login point.

Send notification to user
Figure 6 shows notification screenshot and code that exposes the actual push UI to the user when a push request is received from the FCM server. Push can be sent by creating separate Notification details instances for Android and IOS and handing them over as parameters to the flutter_local_notification implementation.

4 Evaluation

Our app service underwent evaluation to assess its performance and effectiveness. The evaluation focused on data collection accuracy, empirical results, and alignment with project objectives.

Data collection accuracy was measured by comparing the collected notices with the actual notices from university websites, resulting in a perfect match and a

100% accuracy rate. However, specific empirical results for notification delivery speed and user engagement were not obtained during this evaluation phase.

Although the evaluation confirmed a high level of data collection accuracy, a comprehensive assessment of the system's alignment with project objectives was not possible due to the challenges in measuring notification delivery speed and the absence of user engagement assessment.

In conclusion, the evaluation demonstrated a 100% accuracy rate in data collection, indicating effective notice gathering. However, further evaluation is required to measure notification delivery speed and assess user engagement, enabling a comprehensive evaluation of the system's performance in meeting the project objectives.

5 Limit and Discussion

5.1 Discuss assumptions

The success and effectiveness of notiSKKU depend on the assumption that university students face challenges in accessing targeted major and topic information. It assumes that students often struggle with the overwhelming amount of information scattered across multiple sources, leading to stress, confusion, and inefficiency in the search process. However, it is essential to validate these assumptions through user research, surveys, and feedback to ensure the solution meets the actual needs of the target users.

5.2 Discuss constraints and limitations

1. Data Availability and Reliability
   The effectiveness of notiSKKU relies on the availability and reliability of data from university websites. Changes in website structure or notice formats, as well as security measures, can affect accurate data extraction. Continuous monitoring and maintenance are necessary to ensure data reliability.
2. Integration with University Systems
   Integrating notiSKKU with university systems may face constraints due to limited APIs or strict data access regulations. Collaboration with university administrators and IT departments is required to overcome these challenges.
3. Privacy and Security
   Privacy and security concerns arise from handling user data. Implementing appropriate security measures, complying with data protection regulations, conducting regular security audits, and addressing vulnerabilities are crucial to maintain user trust.

While notiSKKU aims to address the challenges faced by university students in accessing targeted major and topic information, it is essential to consider the assumptions, constraints, and limitations discussed above. Regular evaluation,

user feedback, and adaptation to changing circumstances will contribute to the ongoing improvement and success of the application.

6 Related Work

6.1 Kingo-M Application

Kingo-M is a mobile app designed to provide students with comprehensive information about school life, including shuttle bus schedules, academic calendars, library information, and study room reservations. It also features a digital student ID and unified message box for receiving school announcements. However, the app has limitations, such as the absence of an active information portal, limited search functionality, and the presence of commercial messages that may be bothersome to some users.

6.2 Everytime

Everytime is a popular mobile app that offers academic management features such as class scheduling and to-do lists. It also provides access to campus dining options and includes an anonymous community platform for students to communicate and share information. While the app has become an essential tool for students to support each other, the user-generated information may lack immediacy and reliability. Overall, Everytime serves as a valuable space for students to connect and navigate their academic journey together.

6.3 Position of our work

Our proposal stands out from previous apps like Kingo-M and Everytime by providing a centralized platform for users to access and manage notices from multiple university websites. We prioritize real-time and reliable information by directly scraping notices from official sources, ensuring the most up-to-date and trustworthy content. Our focus on notice aggregation, active information retrieval, and the delivery of real-time, reliable information aims to enhance the user experience and meet the specific needs of our target users in their academic lives.

7 Conclusion

notiSKKU represents a promising solution to the challenges faced by university students in accessing targeted major and topic information. The project's achievements highlight the effectiveness and impact of our solution in improving the overall experience of students. Moving forward, it is crucial to continuously evaluate and refine notiSKKU based on user feedback, address the identified limitations, and adapt to the evolving needs and expectations of students. By doing so, notiSKKU can continue to enhance the university experience for students, fostering academic success, and personal growth.