Pusan National University Computer Science and Engineering Technical Report
2025-05

AI-powered Smart Notification System

202255636 Kadyrov Adilet

202255634 Yegizbayev Zholan

지도교수: 조환규

Table of contents

1. Project Objectives

1.1. Introduction

Managing multiple commitments across healthcare, work, and personal life can be challenging, especially when schedules involve complex priorities or potential conflicts. These challenges often lead to missed deadlines, forgotten appointments, or inefficient time management. Busy professionals, elderly individuals, and those managing chronic conditions are particularly at risk, as they may struggle to balance competing demands across different aspects of their lives. In many cases, caregivers and managers also lack real-time visibility into whether important tasks are being completed as planned.

To address these problems, this project proposes the development of a **Smart Cross-Domain Notification System** that uses artificial intelligence to intelligently coordinate and prioritize alerts across all areas of life. The system will allow healthcare providers to send medication instructions, employers to assign tasks, and individuals to input personal commitments - all through a unified platform that automatically detects and resolves scheduling conflicts. The AI engine will extract key details from each input, determine optimal reminder timing based on user behavior, and escalate critical alerts when needed. A smart assistant will help answer scheduling questions, while the conflict detection system will warn users about potential overlaps between medical, professional, and personal obligations.

The purpose of this project is to create an intelligent, user-friendly platform that supports individuals in managing all aspects of their lives seamlessly. It aims to improve time management and reduce errors by combining AI-powered scheduling with practical notification tools for daily use. Ultimately, the goal is to minimize missed commitments, enhance coordination between different life domains, and build a more responsive, human-centered organizational system that adapts to each user's unique needs and patterns.

1.2. Detailed Projects Goals

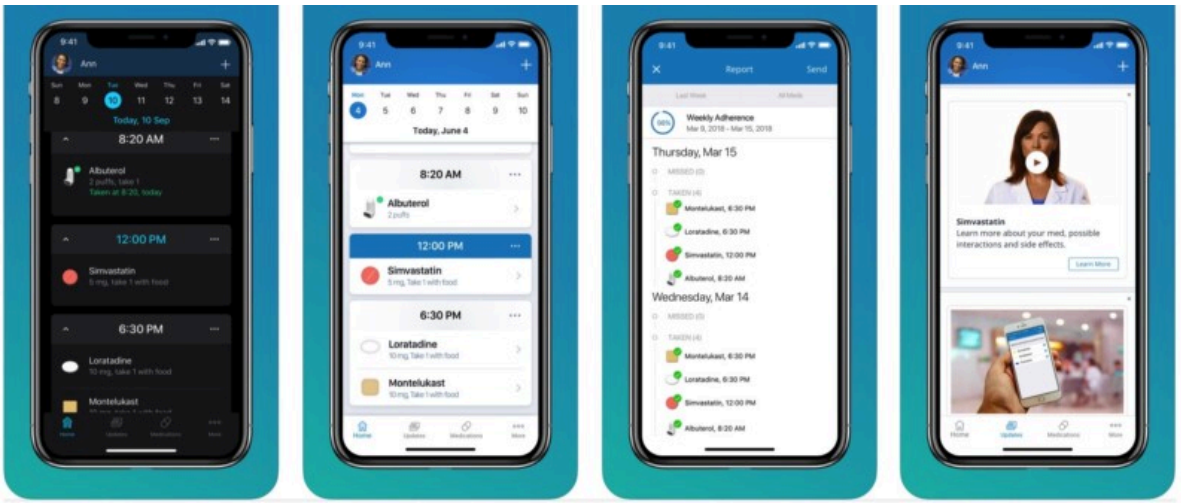
Goal	Brief Explanation
User Authentication and Role Assignment	Implement secure sign-up/login via Google, Kakao, or email (with verification). Users can register with specific roles such as doctor, patient, supervisor, or agent. Role verification may be required (e.g., hospital verification for doctors).
Task Communication and Messaging Platform	Allow task creators (e.g., doctors or supervisors) to send structured tasks or instructions to task receivers (e.g., patients or agents). Each task includes a title, description, timing (start time, repeat cycle), and optional notes.

Automatic Extraction of Task Details from Unstructured Text	Apply AI/NLP to extract task elements (e.g., who, what, when, how) from free-text instructions. For example, extract medication name, dosage, and schedule from prescription notes, or extract deadlines from plain-text instructions.
Task Conflict/Duplication Detection	Analyze the task schedule to detect conflicts, such as work overload or overlapping tasks, and alert users to adjust. Also, analyze drug-drug interactions (DDI) and medication-related activities (e.g., exercise restrictions) to identify potential risks. Provide notifications if scheduled tasks conflict with medical advice or prescriptions, prompting users to modify their plans or consult with healthcare providers.
Smart Chatbot for Task and Medication Guidance	Provide a chatbot interface that answers user questions like “What are my tasks today?”, “What is this medicine for?”, or “When is my next break?” based on their current task schedule and task history.
Timed Notifications and Critical Task Alerts	Send timely push notifications for all upcoming tasks. If the user misses a critical task (e.g., essential medication or urgent assignment), an alert is triggered and optionally sent to the task creator.
User-Initiated Task Input and Tracking	Enable users to manually add tasks (e.g., additional medications, self-directed work tasks) into their schedule. These tasks are integrated into the same task-checking and notification flow.
Progress and Adherence Report Generation	Generate reports that summarize the user’s task completion status (e.g., medication adherence, completed assignments) either periodically or on-demand. These reports are accessible by the task creator (e.g., doctor or manager).

2. Target problem and requirements analysis report

2.1. Analysis of Similar Systems

Medisafe



Purpose: A mobile app designed to help users manage and adhere to their medication schedules.

Key Features:

- Medication reminders with customizable scheduling.
- Missed-dose alerts sent to caregivers or family members.
- Drug interaction warnings for multiple prescriptions.
- Informational support about each medication.
- Integration with wearable devices (e.g., smartwatches).
- Cross-platform support (Android and iOS).

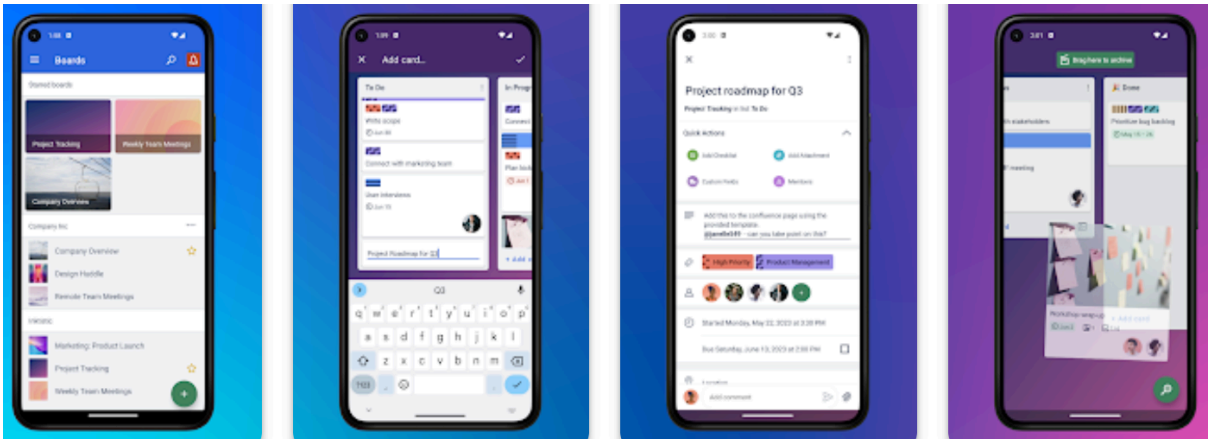
Strengths:

- User-friendly interface suitable for all ages.
- Encourages adherence through alerts and progress tracking.
- Supports multiple user profiles for family care.

Limitations:

- No direct communication between doctors and patients.
- Lacks AI-based analysis or dynamic personalization.

Trello



Purpose: A visual task and project management tool that uses a board-and-card system to organize tasks, projects, or workflows across teams or individuals.

Key Features:

- Kanban-style boards with lists and draggable cards.
- Task assignment, due dates, and checklists.
- Labels, comments, attachments, and custom fields.
- Automation with "Butler" (rules, triggers, actions).
- Integrations with tools like Slack, Google Drive, and Jira.
- Mobile and desktop app support.

Strengths:

- Intuitive, highly visual interface – easy to understand at a glance.
- Flexible for both personal and team use (from to-do lists to complex workflows).
- Free tier sufficient for many small teams and individuals.
- Strong third-party integration ecosystem.

Limitations:

- Limited native reporting and analytics tools.
- Can become cluttered with too many cards/boards over time.
- Not ideal for task dependencies or timeline/Gantt views (without plugins).

2.2. Functional Requirements

ID	Requirement	Description
F1	User Authentication	Support login/signup via Google, Kakao, or email. Role-based registration (e.g., doctor, patient, agent, supervisor).
F2	Task Messaging System	Allow task senders (e.g., doctors, managers) to schedule and assign tasks to recipients.
F3	Timed Notifications	Send push notifications for upcoming tasks and missed critical tasks.

ID	Requirement	Description
F4	Task Input & Parsing	Users can enter unstructured task descriptions; the system extracts structured data using AI/NLP.
F5	Conflict Detection	Detect task overlaps, critical medication interactions, or scheduling conflicts.
F6	Chatbot Assistant	Answer user queries related to task schedules, purposes, and reminders.
F7	Task Progress Tracking	Track task completion, missed tasks, and generate adherence reports.
F8	Multi-Domain Adaptability	System must support both medical and non-medical task types with a consistent interface.
F9	Manual Task Addition	Users can manually add personal tasks or medications.

2.3. Non-Functional Requirements

ID	Requirement	Description
NF1	Platform Compatibility	System should run on both Android and iOS (or be web-based with mobile support).
NF2	Usability	Interface should be user-friendly for both tech-savvy and non-tech-savvy users (e.g., elderly patients).
NF3	Data Privacy	Follow secure practices in handling medical and personal data.
NF4	Scalability	Able to support increasing users without major redesign.
NF5	Extensibility	Should support future features like team collaboration, wearable device integration, or analytics dashboards.
NF6	Offline Support	Basic task viewing and reminders should be available without constant internet access.

3. Results of analysis of realistic constraints and countermeasures

3.1. Realistic Constraints

1) It is difficult to test the system with a large number of real users across different roles (e.g., patients, doctors, supervisors) in a live environment.
2) Testing real-time notifications and alerts in various life domains (healthcare, work, personal) can be challenging due to unpredictable user schedules.

3) Integrating actual hospital systems, work task platforms, or personal calendars in early development stages may not be feasible.

4) Users may have varying levels of digital literacy, especially elderly individuals, which could affect usability.

5) Stable internet connection may not always be available for all users.

3.2. Countermeasures

1) Simulated user agents (bots) can be created to mimic different roles and behaviors for testing purposes, ensuring realistic interaction without needing large-scale human testing.

2) Use scheduled test scenarios and mock data to simulate real-time alerts and conflicts across healthcare, work, and personal commitments.

3) Begin with simplified, standalone modules for healthcare and work tasks, using mock data instead of full integration. Integration with external systems can be added in later development stages.

4) Design a simplified user interface with clear visuals, voice support, and guided onboarding steps to help users with low digital literacy.

5) Implement offline support with local caching and scheduled syncing. Notifications can be handled using device alarms, and SMS fallback can be used for critical alerts when offline.

4. Design Document and Implementation Plan

4.1. System Architecture and Backend Design

The system will be built with a **Node.js backend**, offering a flexible server environment. **MongoDB** will serve as the primary NoSQL database to store user data, medication instructions, interaction logs, and chatbot responses. The architecture will include:

- **Node.js + Express Server**: Handles API requests, ***business logic***, and data flow between the app and database.

- **MongoDB (via Mongoose)**: Stores structured user profiles, medication records, DDI data, chat log, and so on.

- **Firebase Cloud Messaging (FCM)**: Sends push notifications and emergency alerts to Flutter clients.

- **Firebase Authentication**: Provides secure login with Google, Kakao(Firebase Custom Auth Token), and email sign-in options.

- **JWT (JSON Web Tokens)**: Used alongside Firebase Auth to manage session handling and secure API access.

The server will be hosted on a platform such as **Render**, **Vercel**, **AWS EC2**, or **Heroku**, depending on budget and scalability requirements.

4.2. User Authentication and Role Assignment

- Users can register and log in securely through Firebase Authentication.
- Upon registration, each user selects a role (Doctor, Patient, Supervisor, or Agent).
- Role information is stored in MongoDB and determines feature access. For instance:
    - Doctors can assign medical tasks.
    - Supervisors can create work assignments.
    - Patients and Agents can receive tasks and update completion status.
- JWTs ensure secure and authenticated communication between the frontend and backend.

4.3. Task Input and AI-based Detail Extraction

- Task creators (e.g., doctors or supervisors) submit **free-form instructions**.
- The backend uses **basic NLP (regex, keyword parsing)** to extract core task components:
    - Task Owner and Receiver
    - Task Type (e.g., medication, meeting)
    - Timing (start date, cycle)
    - Priority or urgency level
- Future versions will incorporate **BERT-based NLP services** to process Korean-language input more accurately.
- Extracted data is stored in MongoDB and linked to the appropriate user ID.

4.4. Scheduling Conflict Detection and Resolution

- The system automatically scans a user's calendar for overlapping or conflicting tasks across all domains (medical, personal, work).
- Conflicts are flagged and the user receives a real-time alert with suggestions to reschedule.
- Conflict detection includes:
    - Time overlap
    - Excessive task load
    - Missed critical task escalation
    - Drug-drug and drug interaction with scheduled activities
- A lightweight scheduling engine will power these checks.

4.5. Smart Chatbot Assistant

- A built-in chatbot helps users query their schedule and tasks using natural language.
    - Examples: "What are my tasks today?", "When do I take my next medicine?", "Do I have free time this evening?"
- Development Plan:
    - **Phase 1**: Rule-based chatbot in Node.js for fixed query patterns.
    - **Phase 2**: Upgrade using BERT or Dialogflow for intent recognition in Korean.
- The chatbot references stored user data and task history from MongoDB.

4.6. Push Notifications and Critical Alerts

- A task scheduler runs on the backend using node-cron or equivalent.
- Notifications are sent via **Firebase Cloud Messaging**:
    - Task reminders
    - Missed critical task alerts (e.g., medication not taken)
- Important tasks may trigger an **escalation** alert to supervisors or doctors.
- Notifications will include actionable buttons (e.g., "Mark as Done").

4.7. Manual Task Input and User-Driven Scheduling

- Users (patients, agents, individuals) can enter their own tasks.
    - Examples: Gym session, shopping, study time, personal events.
- These are treated the same as externally assigned tasks — integrated into the system and eligible for notification, conflict checks, and reporting.

4.8. Adherence Monitoring and Report Generation

- All task interactions (completions, delays, missed tasks) are logged in MongoDB.
- Task creators (doctors, managers) can request:
    - Weekly / monthly reports
- These reports help monitor:
    - Medication adherence
    - Productivity or compliance in work
    - Personal balance and wellness

5. Development Schedule and Role Division

5.1. Development Schedule

구분	5 월				6 월				7 월				8 월				9 월			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Define detailed project requirements, finalize architecture, set up dev environment																				
Design DB schema and																				

implement user login/registration with Firebase																			
Develop core APIs for task CRUD using Node.js and MongoDB																			
Implement Flutter UI screens and navigation logic																			
Apply AI-based task parsing and priority classification using NLP/ML																			
Develop conflict detection mechanism across domains																			
Integrate chatbot interface for task management and inquiries																			
Enable push notifications via Firebase and ensure real-time updates																			
System testing, error correction, UI refinements																			
Final documentation, prepare for final demo and submission																			

5.2. Role Division

Name	Role
Common	Documentation, Testing, Web Development, Database Management
Kadyrov Adilet	1. AI Module Development 2. Cloud Deployment 3. Frontend & API Integration
Yegizbayev Zholan	1. Backend Development 2. App Development 3. API Development