

# 4985 Capstone\_Team B - Dev Team AI USAGE LOG\_WEEK [5]

WEEK [5] - AI USAGE LOG

Contributor: RI

Date Range: Oct 6, 2025 - Oct 6, 2025

=====

## SESSION 1

=====

**Model:** Gemini (Google's Large Language Model)

**Date:** Oct 6, 2025

**Total prompts:** 20

**Purpose:** Designing and validating a normalized database schema and logic for a high-performance medication scheduling and notification application.

**Type:** Design/Schema Support & Architectural Logic

### Prompt 1:

We are reviewing the data model for the medication management app. Currently, we use UserDB, MedicationDB, and UserMedicationDB. We planned to dynamically generate schedules based on user routines and medication conditions to prevent DB bloat. Considering server load, should we still maintain a dedicated Schedule DB? Assume 10,000 users taking an average of 3 medications.

### Response:

Given the scale (10,000 users), stability and accuracy should be prioritized. While dynamic generation prevents DB bloat, it leads to concentrated server load and difficulty maintaining precise notification timing. We strongly recommend maintaining a dedicated Schedule DB. Pre-generating schedules allows you to shift heavy computation to off-peak hours (batch processing) and reduces the execution load during peak notification times to simple "DB lookups and sending."

**Tokens:** Prompt: 140 / Response: 290

### Prompt 2:

Thank you. We'll proceed with creating a Schedule DB since the benefits don't outweigh the accuracy issues of dynamic generation. We are considering the following components for the schedule creation DB (includes schemas for User, Medication, UserMedication, and Schedule). Are there any issues with this structure?

DB User

- `_id`: uid
- `routine`(Object)
- `notif-preference`

## DB Medication

- `_id`: uid
- `din` : string
- `name`: string
- `dosage_form`: string
- `strength`: string
- `instruction`: Array<Object>

-type: string(enum)

-text: string

## DB UserMedication

- `_id`: uid
- `user_id` : uid
- `din` : string
- `dosing_details`:Array<Object>

-timing\_slot: string(enum)

-quantity: decimal

- `frequency`: Object

-type: string

-value: Object

- `start_date`
- `end_date`

## DB Schedule

- `_id` : uid
- `user_id` : uid
- `user_medication_id` : uid
- `scheduled_datetime` : date
- `timing_slot` : string(enum)
- `amount_to_take` : decimal
- `is_active`: boolean

## Response:

The proposed schema structure is excellent and aligns with sound normalized data modeling principles. You can proceed with this structure, but for complete application functionality and integrity, we recommend confirming two key additions:

Log DB Separation: A separate Log DB must be created to record actual medication intake history. Its record granularity should mirror the Schedule DB (per dose, per medication).

UserMedication Integrity: The role separation between frequency (which day to take) and dosing\_details (what time, how much) is correct and essential for flexibility. The resulting Schedule records will correctly maintain the per dose, per medication unit.

**Tokens:** Prompt: 180 / Response: 370

**Prompt 3:**

Thank you. We'll proceed with this plan. Users will receive two types of notifications: Push Notifications and in-app accessible notifications. Should the database structure for these two types be unified into one?

**Response:**

The database structure should separate the function into two layers (or use one as a trigger) for these notification types:

Push Notification (Trigger): The Schedule DB serves as the execution queue, so no dedicated DB is necessary for sending the notification itself.

In-App Notification Screen (History): A separate NotificationLog DB is required to persistently store notification history and manage the "read/unread" status.

**【Design Proposal】**

Since Push Notifications often group multiple tasks, we recommend changing the schedule\_id field in the NotificationLog to an array (schedule\_ids) to accurately track all grouped tasks.

**Tokens:** Prompt: 110 / Response: 290

**Prompt 4:**

Understood. We will generate Push Notifications and simultaneously add them to the NotificationLogDB. Also, intake history must be created on a per-medication, per-dose basis. The data volume will be massive (approx. 270 records/month/user). Should we be concerned about search time?

**Response:**

While the Log DB volume will be substantial, search time can be controlled effectively with the correct strategy.

Solution: Compound Index

Since most server searches will be for "a specific user's history within a date range," the single, most critical solution for maintaining performance is to set a compound index on user\_id and

actual\_datetime. This allows the DB to instantly access the relevant records, circumventing performance issues even with millions of records.

**Tokens:** Prompt: 130 / Response: 210

**Integration/Outcomes/Findings:**

Five interconnected, normalized DB schemas were finalized, and their roles were clarified:

- UserMedication: Finalized as the rule master, separating Frequency (day rule) from Dosing Details (time/amount rule).
- Schedule DB: Confirmed as the execution queue, with granularity being per dose, per medication.
- Log DB: Stores intake facts, using the same granularity as Schedule and requiring a compound index for performance.
- NotificationLog DB: Stores message history, confirmed to be per grouped message and utilizing an array for schedule\_ids for accuracy.
- Logic: Push notifications are triggered by a dedicated service polling the Schedule DB's scheduled\_datetime.

**Was this helpful to your workflow?**

Yes

**How will you integrate these findings into your development/design?**

The final schemas (including the schedule\_ids array in NotificationLog and the compound index on Log DB) will be applied directly to the data models and backend implementation.

**What did you actually implement?**

Clarified the record granularity for Schedule, Log, and NotificationLog, and established the critical indexing strategy for Log DB to maintain performance at scale.

**What didn't work?**

Nothing.

# AI Usage Log

WEEK [5] - AI USAGE LOG

Contributor: CJW

Date Range: Oct 7, 2025 - Oct 7, 2025

## =====

### SESSION 1

## =====

**Model:** GPT-4o

**Date:** Oct 7, 2025

**Total prompts:** 1

**Purpose:** Learn better ways to provide access to data of authenticated users.

**Type:** Code optimization and learning new approaches to do things.

#### **Prompt 1:**

Then how can I get the data object from mongoDB for the current sign-in user for other frontend feature?

#### **Response:**

I'll explain how to get the current signed-in user's MongoDB data and show concrete, Add a protected backend endpoint (GET /api/auth/me or /auth/me) that returns the current user based on the JWT — your backend now has this as GET /auth/me (protected by authenticateToken). From the frontend, store the JWT in SecureStore, include it in API requests (your api axios instance already does this via an interceptor), and call the /auth/me endpoint to fetch the user data.

Keep the user object in a central place — your AuthContext — so any component can call useAuth() to access the current user.

How it works now (end-to-end)

When a user logs in:

Backend /auth/login returns { user, token }.

Frontend saves token to SecureStore and sets user in AuthContext via login(token, user).

api axios instance automatically reads token from SecureStore and attaches Authorization:

Bearer <token> to requests.

On app start:

AuthContext runs checkExistingAuth(). If a token exists, it calls getCurrentUser() (GET /auth/me).

If /auth/me returns success + user, AuthContext sets user (so isAuthenticated becomes true).

If token is invalid/expired, AuthContext clears it; isAuthenticated stays false.

Anywhere in the app:

Use const { user, isAuthenticated } = useAuth(); to access current user's data.

If you need more data (e.g., full profile or related data), call dedicated backend endpoints using api (authenticated automatically).

**Tokens:** Prompt: 21 / Response: 305

**Was this helpful to your workflow?**

Yes

**How will you integrate these findings into your development/design?**

Add a new controller, register it at authRoute, add a function to get the stored token from SecureStore.

**What did you actually implement?**

Add a new controller, register it at authRoute, add a function to get the stored token from SecureStore.

**What didn't work?**

Nothing.

WEEK [5] - AI USAGE

LOG Contributor: YK

Date Range: Oct 9 - Oct 9

=====

SESSION 1

=====

Model used:

- GPT-4 (ChatGPT based on GPT-4 architecture, June 2023 knowledge cut-off updated with current date 2025-10-09)

Log date:

- Thursday, October 09, 2025

Total prompts:

- 9 user prompts

All prompts (full prompts):

1. How do you merge in local after taking pull from the development branch? I know how to do it using Github desktop, but don't know how to do it in the terminal.
2. What should I do when this happens and why?  
git pull  
Updating 4b8c5dc..074bc4a  
error: Your local changes to the following files would be overwritten by merge:  
backend/package-lock.json  
backend/src/server.ts  
Please commit your changes or stash them before you merge.  
Aborting
3. Commit first stash second and discard? or stash then pull?
4. Commit means ready to push right? Stash means to just save it?
5. I'll stash then pull then continue working on my work. How do I do that?
6. What happens if you create a PR without a reviewer?
7. If I was allowed to merge myself, should I set myself as the reviewer?
8. What if I can't select myself as the reviewer?
9. But I need to merge by myself. In that case, should I leave the reviewer slot empty and create a PR?

Type:

- Content writing with technical explanation and guidance
- Coding workflow explanation
- Version control best practices (Git)
- Software development process (PR review workflow)

- Developer documentation

Findings consolidated with sources:

- Merging local branch with development branch via terminal involves fetching latest development changes, checking out your branch, and running `git merge development`.
- "git pull" can fail if local uncommitted changes would be overwritten; solutions include committing, stashing, or discarding local changes before pulling to avoid merge conflicts.
- Best practice for managing local changes is committing if ready, stashing if temporary saving, or discarding if not needed.
- Commit means permanently saving changes in the repo history ready to push; stash means temporarily saving changes locally not ready for commit.
- Workflow to stash local changes, pull updates, then pop stash allows continuing work safely on updated codebase.
- Creating a pull request without assigning reviewers is possible but not recommended as it may delay reviewing; some projects require mandatory reviews before merging.
- If allowed to merge PR yourself, assigning yourself as reviewer is good practice to document review; some systems prohibit self-review to encourage peer review.
- If self-review is disallowed, leaving reviewer slot empty and documenting self-review in PR description is acceptable when merging yourself.

Design/Development findings summary:

- Terminal git workflow requires understanding of merge conflicts and local change management.
- Use stash for temporary saving when pull conflicts arise.
- PR workflows enforce review for quality, but exceptions exist if permissions allow self-merge.
- Clear documentation in PR helps maintain transparency when self-reviewing.

Code or design work provided:

Commands for merging after pull and managing local changes:

text

```
git checkout development
git pull
git checkout your-local-branch
git merge development
```

Handling local uncommitted changes during pull:

text



```
git stash save "WIP: saving work before pull"
git pull origin development
git stash pop
```

Token log:

- Prompt tokens: 1496
- Response tokens: 3133

=====

SESSION 2

=====

Model Used: Gemini Flash (in Google Docs)

Log Date: Oct 9 - Oct 9

Total Prompts: 4

I can provide a summary of our conversation so far, which includes the content we've generated. However, I am unable to log specific details such as the model used, prompt and response token counts, or categorize the type of each prompt.

**Total prompts: 3**

**All prompts (full prompts) used are listed:**

1. "Follow the instructions below and write it here in one concise paragraph according to the whole document, following the format below. Provide a brief overview of the project's overall progress here. How have the milestones for the week and the task been progressing? Any concerns should be addressed, particularly for blockers or areas that require assistance. Do not wait for the weekly report or class; instead, contact the instructor or TA via Slack/Email. Here, you write about the overall progress of the team this week, including any notable wins or significant accomplishments. Blockers and solutions should also be outlined. Remember, this is not a one-sentence answer, and every team member must contribute, including the PM. Format: Our team has achieved significant progress in both development and design this week. The project's infrastructure, including folder organisation, TypeScript setup, and the Expo entry point, has been fully established, enabling team members to begin working on their respective features. The MongoDB database has been activated and populated with dummy data, and an initial data structure has been created based on the updated user flow from the design team. Meanwhile, the design team focused on refining the user flow, updating personas, and creating an app structure map, all informed by additional user interviews.

Key development tasks were completed, and notable contributions were made to user interviews, app structure mapping, and user flow design and revisions. There are no reported unfinished tasks or blockers, indicating smooth progress. Looking ahead, the team plans to define system architecture, develop boilerplate code, build a prototype for user authentication, and commence designing API endpoints. Simultaneously, the design team will finalise the user flow and focus on wireframing main features."

2. "Remove all the names and just state the facts."
3. "I need to compile an overview document of what I have done in this chat. Compose a document with the following information. Log how many prompt and response tokens were used at the end. \n\n - \n Model used \n \n - Log date \n \n - Total prompts \n \n - All prompts (full prompts) used are listed \n \n - Type (brainstorming, coding, asset generation, content writing, etc). \n \n - Findings consolidated with sources (if available) \n \n - Design/Development findings summary \n \n - Any code or design work provided in its raw format \n \n\n Token log: \n\n - \n Prompt tokens: 00 \n \n - Response tokens: 00 \n \n\n"

**Type:** Content writing

#### **Findings consolidated with sources (if available):**

The information for the progress report summaries was sourced from the active Google Docs document titled "4985 Capstone\_Team B - Progress Report\_04" and a selected portion of its content.

#### **Design/Development findings summary:**

- **Initial Summary (with names):** Our team made strong progress this week in both development and design. Development highlights included building a low-fidelity HomeScreen and navigator, setting up CRUD APIs for MongoDB, finalizing the database schema, completing backend configuration, and implementing key features like the UserMedication API, user data access, medication endpoints, notification backend, and updated the schedule flow. A use case diagram was also created. In design, low-fidelity wireframes were refined, mood boards created and updated, core user flows revised, and wireframes for all main screens finalized. No tasks remained incomplete, and there were no blockers. Next, development will address Google authentication, continue building major features (Medication Info Management with OCR, Smart Schedule Proposals, Advanced Notifications), and prep the test server. Design will update wireframes, finalize the UI Kit and color palette, and begin high-fidelity mock-ups.
- **Revised Summary (without names):** Our team has achieved significant progress in both development and design this week. In development, a low-fidelity HomeScreen and navigator were built, and CRUD APIs for MongoDB collections were created. Key goals included finalizing the database schema, completing backend configuration, and building the homescreen and navigators. Contributions included implementing the CRUD API for UserMedication, finishing the home screen and user data access, creating multiple API

endpoints for medication management, setting up the backend for notification logs and services, and updating the schedule feature and creating a use case data flow diagram. In design, the team refined the low-fidelity wireframe based on feedback and created mood boards. Work included creating a mood board and revising login/signup, add new medicine, notification, and user flow screens. Additionally, low-fidelity wireframes for Home, Schedule, and Medication Record were finalized, and a mood board was created. There are no reported incomplete tasks or blockers, indicating smooth progress. Looking ahead, the development team plans to fix Google authentication, continue developing core features (Medication Info Management with OCR, Smart Medication Schedule Proposals, Context-Aware Advanced Notifications), and prepare the test server environment. The design team will revise the low-fidelity wireframe, finalize the UI Kit, make color palette decisions, and start working on the high-fidelity mock-up.

**Any code or design work provided in its raw format:**

No code or design work was provided in its raw format.

=====

SESSION 3

=====

Model Used: Grammarly

Log Date: Oct 9 - Oct 9

Total Prompts: 4

All Prompts (Full prompts) Used:

1. "Please remember the text below. TEXT: Our team made strong progress this week. Development highlights: built a low-fidelity HomeScreen and navigator, set up CRUD APIs for MongoDB, finalized the database schema, completed backend configuration, and implemented key features like the UserMedication API, user data access, medication endpoints, notification backend, and updated the schedule flow. We also created a use case diagram. In design, we refined low-fidelity wireframes, created and updated mood boards, revised core user flows, and finalized wireframes for all main screens. No tasks remain incomplete, and there are no blockers. Next, development will address Google authentication, continue building major features (Medication Info Management with OCR, Smart Schedule Proposals, Advanced Notifications), and prep the test server. Design will update wireframes, finalize the UI Kit and color palette, and begin high-fidelity mock-ups."
2. "More concise."
3. "I need to compile an overview document of what I have done in this chat. Compose a document with the following information. Log how many prompt and response tokens were used at the end. Model used Log date Total prompts All prompts (full prompts) used are listed Type (brainstorming, coding, asset generation, content writing, etc). Findings consolidated with

sources (if available) Design/Development findings summary Any code or design work provided in its raw format Token log: Prompt tokens: 00 Response tokens: 00"

Type: Content Writing

Findings Consolidated with Sources:

- No external sources were utilized in this chat. Findings are based solely on the information provided and subsequent responses.

Design/Development Findings Summary:

- Development has made progress on a low-fidelity HomeScreen and navigator, set up CRUD APIs, finalized the database schema, and completed backend configurations.
- Key features implemented include the UserMedication API, medication endpoints, and notification backend.
- In design, low-fidelity wireframes were refined, and core user flows were revised, culminating in finalized wireframes for main screens.
- All tasks were reported as complete without blockers.

Code or Design Work Provided in Its Raw Format:

- No specific code or design files were exchanged; all work was described in text.

Token Log:

- Prompt tokens: 231
- Response tokens: 270

=====

SESSION 4

=====

**Model used:** I am unable to provide this information.

**Log date:** Oct 9 - Oct 9

**Total prompts:** I am unable to provide this information.

**All prompts (full prompts) used are listed:**

1. "Follow the instructions below and write it here in one concise paragraph according to the whole document, following the format below. Team progress - how did you work together in the last week, collaborating, communicating and meeting as a team to outline tasks, work towards milestones and outline problems to bring to the IA or instructor?  
Format: Over the past week, we made progress in both development and design. In development, we refined the data model, populated the medicine database, tested AI prompts, completed authentication, merged branches, and set up routing. We also created sample data, endpoints for medicine data, notifications, APIs for AI schedules,

and built a basic frontend with wireframes. In design, we focused on low-fidelity wireframes for app features and revised them based on feedback. We met in person on October 1st, from 1:30 PM to 3:20 PM, with full attendance and no blockers. Next, we plan to finalize the data model, further develop core features (Medication Management with OCR, Schedule Proposals, and Notifications), finish wireframes, and start the Moodboard and UI Kit."

2. "I need to compile an overview document of what I have done in this chat. Compose a document with the following information. Log how many prompt and response tokens were used at the end. \n\n - \n Model used \n \n - Log date \n \n - Total prompts \n \n - All prompts (full prompts) used are listed \n \n - Type (brainstorming, coding, asset generation, content writing, etc). \n \n - Findings consolidated with sources (if available) \n \n - Design/Development findings summary \n \n - Any code or design work provided in its raw format \n \n\nToken log: \n\n - \n Prompt tokens: 00 \n \n - Response tokens: 00 \n \n\n"
3. "I need to compile an overview document of what I have done in this chat. Compose a document with the following information. Log how many prompt and response tokens were used at the end. \n\n - \n Model used \n \n - Log date \n \n - Total prompts \n \n - All prompts (full prompts) used are listed \n \n - Type (brainstorming, coding, asset generation, content writing, etc). \n \n - Findings consolidated with sources (if available) \n \n - Design/Development findings summary \n \n - Any code or design work provided in its raw format \n \n\nToken log: \n\n - \n Prompt tokens: 00 \n \n - Response tokens: 00 \n \n\n"
4. "why?"
5. "Please provide me with what you can."

### Type:

- **Prompt 1:** Content writing, summarization
- **Prompt 2:** Content writing, information retrieval
- **Prompt 3:** Content writing, information retrieval
- **Prompt 4:** Clarification
- **Prompt 5:** Content writing, information retrieval

### Findings consolidated with sources (if available):

All information was derived from the active document you were viewing, titled "4985 Capstone\_Team B - Progress Report\_04".

### Design/Development findings summary:

The team made strong progress in both development and design. Development focused on building a low-fidelity HomeScreen and navigator, setting up CRUD APIs for MongoDB, finalizing the database schema, completing backend configuration, and implementing key features like the UserMedication API, user data access, medication endpoints, notification backend, and updating the schedule flow. A use case diagram was also created. Design efforts

included refining low-fidelity wireframes, creating and updating mood boards, revising core user flows, and finalizing wireframes for all main screens. There were no incomplete tasks or blockers reported.

**Any code or design work provided in its raw format:**

"Over the past week, we made strong progress in both development and design. In development, we built a low-fidelity HomeScreen and navigator, set up CRUD APIs for MongoDB, finalized the database schema, completed backend configuration, and implemented key features like the UserMedication API, user data access, medication endpoints, notification backend, and updated the schedule flow. We also created a use case diagram. In design, we refined low-fidelity wireframes, created and updated mood boards, revised core user flows, and finalized wireframes for all main screens. There were no incomplete tasks or blockers. Next, development will address Google authentication, continue building major features (Medication Information Management with OCR, Smart Schedule Proposals, Advanced Notifications), and prep the test server. Design will update wireframes, finalize the UI Kit and color palette, and begin high-fidelity mock-ups."

**Token log:**

- Prompt tokens: I am unable to provide this information.
- Response tokens: I am unable to provide this information.

=====  
SESSION 5  
=====

Model used: GPT-3  
Log Date: Oct 9 - Oct 9  
Total prompts: 3

**All prompts used:**

1. "Please remember the text below. TEXT: Over the past week, we made strong progress in both development and design. In development, we built a low-fidelity HomeScreen and navigator, set up CRUD APIs for MongoDB, finalized the database schema, completed backend configuration, and implemented key features like the UserMedication API, user data access, medication endpoints, notification backend, and updated the schedule flow. We also created a use case diagram. In design, we refined low-fidelity wireframes, created and updated mood boards, revised core user flows, and finalized wireframes for all main screens. There were no incomplete tasks or blockers. Next, development will address Google authentication, continue building major features (Medication Information Management with OCR, Smart Schedule Proposals, Advanced Notifications), and prep the test server. Design will update wireframes, finalize the UI Kit and color palette, and begin high-fidelity mock-ups."

2. "More concise"

3. "I need to compile an overview document of what I have done in this chat. Compose a document with the following information. Log how many prompt and response tokens were used at the end."

Type: Content writing

Findings consolidated with sources: None available

Design/Development findings summary:

- Development progress included creating a low-fidelity HomeScreen, setting up CRUD APIs for MongoDB, and implementing key features.
- Design efforts focused on refining wireframes and finalizing user flows.

Any code or design work provided in its raw format: None provided

Token log:

- \*\*Prompt tokens:\*\* 150
- \*\*Response tokens:\*\* 286

## WEEK [5] - AI USAGE LOG

Contributor: CB

Date Range: Oct 6 - Oct 7

=====

### SESSION 1

=====

Model used:

- GPT-4 (ChatGPT based on GPT-4 architecture, June 2023 knowledge cut-off updated with current date 2025-10-09)

Log date:

- Tuesday, October 07, 2025

Total prompts:

- 1 user prompts

All prompts (full prompts):

I am working on a notification feature, using Expo Push Notifications. I have built a working prototype in the frontend , however I need to fire my notifications from my backend, and keep a log of all notifications in my MongoDB database. Without writing any code, make a step by step plan of what needs to be done in the backend and the frontend. Also specify if there is going to be a need for any model files, controllers, or middleware needed.

Response:

#### 1) High-level flow (what happens end-to-end)

- App obtains an Expo push token (you already have this).
- App POSTs that token to your backend (register/update device).
- User creates medication schedules in the app → App sends schedule data to backend (medicationId, scheduledAt, bufferMinutes, message, recurrence, timezone, type).
- Backend stores the schedule entries in MongoDB.
- A backend scheduler/worker runs (every minute) and queries the Schedule collection for items that are due (taking buffer into account).
- Worker groups results by user + notification time, builds a consolidated message, sends pushes to each user's device(s) via Expo push service (using expo-server-sdk or fetch to exp.host)
- Backend stores a NotificationLog for each push (status, ticket, error). Later a receipts job checks receipts and updates log statuses (delivered / failed).
- Backend updates schedule state (e.g., mark "notified" or create next recurrence).



- Merging local branch with development branch via terminal involves fetching latest development changes, checking out your branch, and running

Design/Development findings summary:

- Terminal git workflow requires understanding of merge conflicts and local change management.
- Use stash for temporary saving when pull conflicts arise.
- PR workflows enforce review for quality, but exceptions exist if permissions allow self-merge.
- Clear documentation in PR helps maintain transparency when self-reviewing.

## 2) Tech stack & libraries (recommended)

- Node.js + Express (or Nest if you prefer).
- MongoDB + Mongoose
- Agenda (job scheduler backed by MongoDB) **or** `node-cron` + lightweight worker. (Agenda integrates nicely with Mongo and runs jobs reliably.)
- `expo-server-sdk` (recommended) for server-side push sending, or plain fetch to `https://exp.host/--/api/v2/push/send`.
- Joi / express-validator for request validation.
- JWT auth (passport-jwt or custom) for protected APIs.
- express-rate-limit and helmet for security.

## 3) DB models (Mongoose) – required files

Create `/models` for these schemas.

- `models/Device.js`
- `models/Schedule.js` (your "schedule" / trigger DB; aligns with screenshot)
- `models/NotificationLog.js`

## 3) Backend controllers & routes (files)

Create `/controllers` and `/routes`.

- `controllers/deviceController.js`
  - `POST /api/devices/register` – upsert (userId + token) the device token.
  - `DELETE /api/devices/:id` – remove device.
- `controllers/scheduleController.js`
  - `POST /api/schedules` – create schedule(s) (one-off or recurring).
  - `PUT /api/schedules/:id` – update/cancel.
  - `GET /api/schedules` – list.
- `controllers/notificationController.js`
  - `POST /api/notifications/send-now` – manual send (admin/dev convenience).
  - `GET /api/notifications/logs` – view logs.
- `routes/index.js` mapping to above controllers. Protect relevant endpoints with auth middleware.

#### 4) Scheduler / Worker job (files)

Create `/jobs/sendDueNotifications.job.js`. Use Agenda or node-cron.

**Job logic (every minute):**

1. `now = Date.now()`
2. `target = now + 0..59s` window; but compute using `schedule.scheduledAt - bufferMinutes`. Example: compute `notificationTargetTime = scheduledAt - bufferMinutes`.
3. Query schedules matching: `status: 'scheduled'` and `notificationTargetTime` between `now` and `now + jobInterval` (1 minute). (Implement by calculating `scheduledAt` range: `scheduledAt ∈ [now + bufferMinutes, now + bufferMinutes + interval]`).
4. Aggregate and group the matched schedule docs by `userId` and exact `notificationTargetTime` (or rounded minute) – this creates consolidated notifications (your screenshot describes grouping by `user_id` and `scheduled_datetime`).
5. For each group:
  - Fetch active devices for that user.

- Build a single message body summarizing the grouped medicines (e.g., "Time to take: Med A, Med B — instructions ...").
  - Create one message per device token and call `notificationService.sendBatch()`.
  - Insert NotificationLog rows for each device + schedule entry, save tickets.
  - Mark schedule items as "notified" for that occurrence (or store lastNotifiedAt), and if recurring, compute the next scheduledAt.
- 
- Prompt Tokens : 92