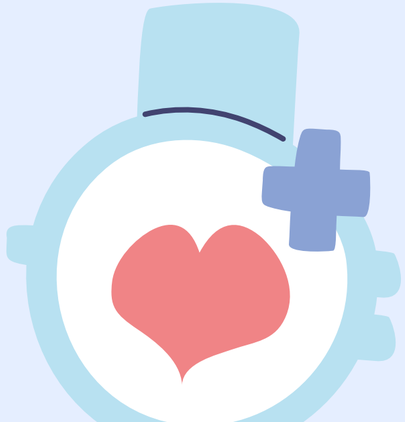
Smarter, Stronger & Safer

By the community for the community

1. Hu Han
2. Raj
3. Naomi
4. Yujia
5. Wang Ke
6. Akanksha



What is our Problem to Solve?

Lack of health-related information

Singaporeans lack real-time information to make informed decisions about their health and safety.

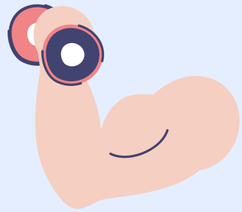
Lack of real-time community updates

No platform available for Singaporeans to share verified health updates in real-time

Our Solution?
Smarter, Stronger & Safer (SSS)

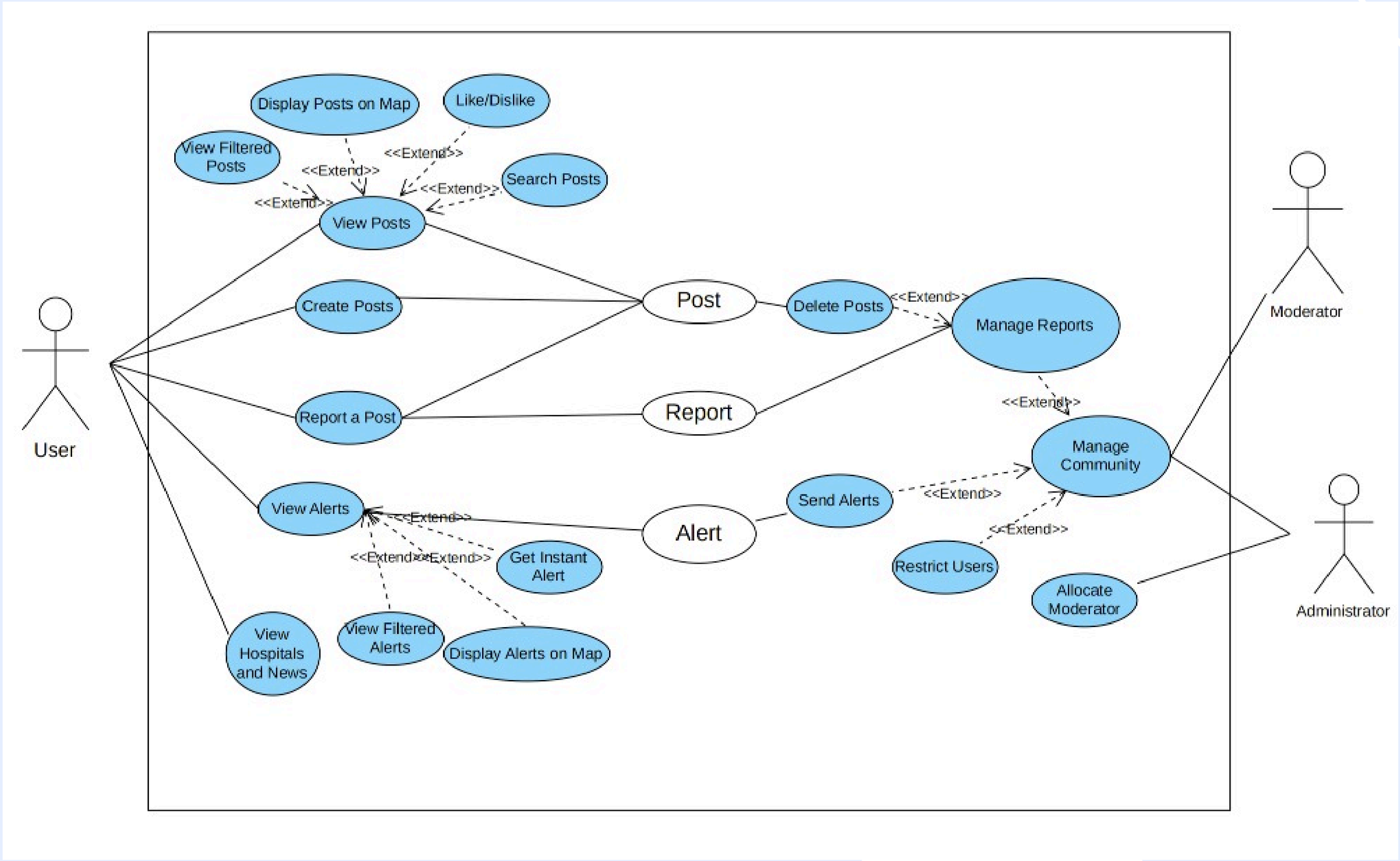
Real-time updates on health and safety issues

By the community, for the community



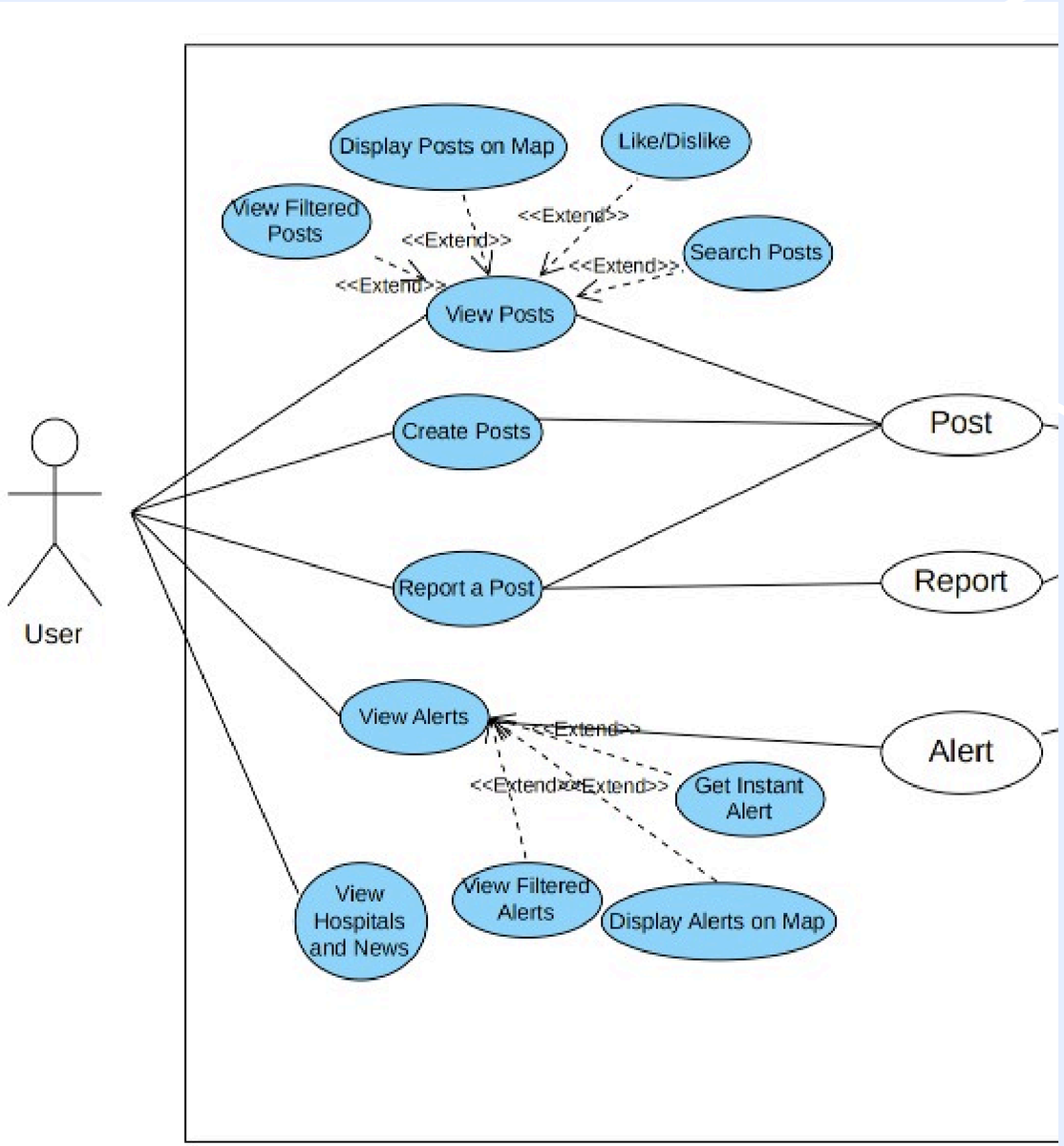
Personalised safety alerts

Use Case Diagram



Key Features

1. **User**
   - Create Posts
   - View Posts
     - Like/Dislike/Comment on posts
     - View filtered posts
     - Search Posts
     - Display Posts on Map
   - Report posts
   - View Hospitals and News
   - View Alerts
     - Filter alerts
     - Display Alerts on Map
     - Get Instant Alert



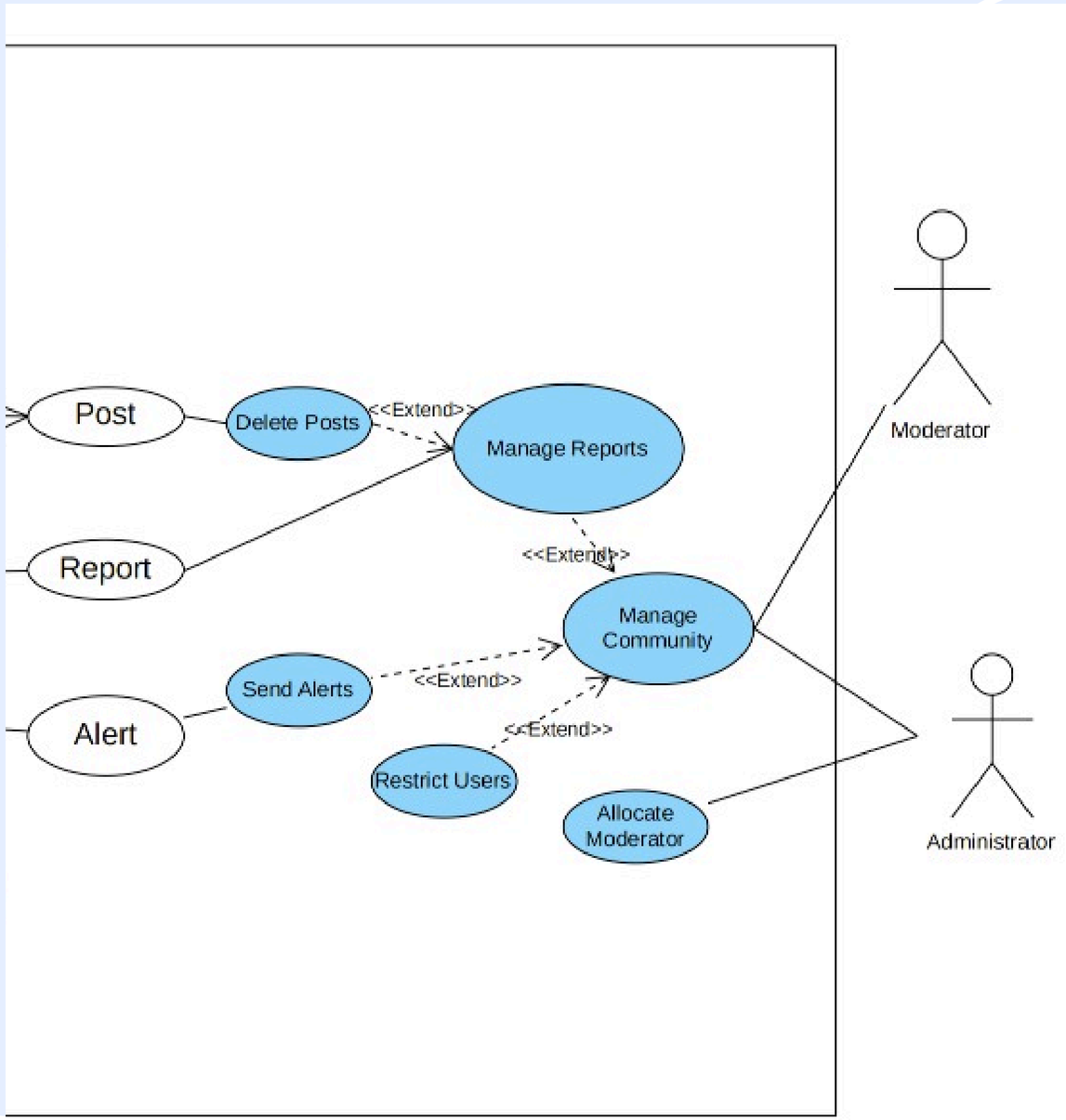
Key Features

2. Moderator

- Manage Community
  - Manage Reports
  - Delete Suspicious Posts
  - Send Alerts
  - Restrict Users

3. Administrator

- Manage Community (Same as Moderator)
- Allocate Moderators



External APIS

1. Google Maps

2. Singapore’s Open Data Portal
   - Dengue Cases
   - Traffic Accidents

3. 24-hour Weather Forecast

4. News Headlines API

01

Good Software Engineering Practices

01

Good Software Engineering Practices

1. Documentation
2. Readability
3. Scrum

Documentation–README

Social Features Implementation

This document explains how to set up and use the likes and comments functionality in the health community application.

Database Setup

The application requires two additional tables in the Supabase database to support likes and comments functionality:

1. **post_like** - Stores post likes
2. **comment** - Stores post comments

To create these tables and set up the necessary policies, execute the SQL commands from the `database/tables.sql` file in your Supabase SQL editor.

Features Implemented

Likes

- Users can like and unlike posts
- Like count is displayed on both post cards and post detail views
- Like status is preserved between sessions
- Only authenticated users can like posts
- The UI updates reactively when users like/unlike posts

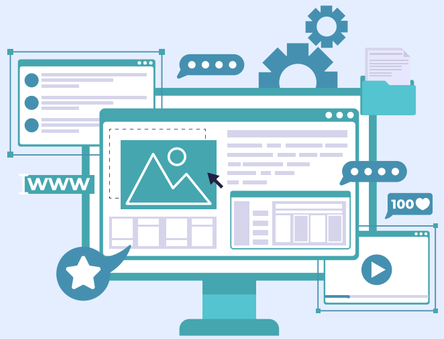
Comments

- Users can add comments to posts
- Comments are displayed in chronological order
- Comment count is displayed on post cards
- Only authenticated users can add comments

Documentation–Comments

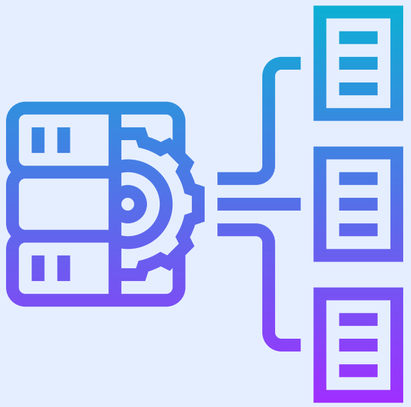
```
34    // Load alerts from both services
35    async function loadAlerts() {
36      isLoading.value = true
37      error.value = null
38      try {
39        // Get notification alerts
40        const notificationAlerts = await getLatestAlerts(50)
41
42        // Get system alerts
43        const apiAlerts = await getAlerts()
44
45        // Process and merge alerts
46        const combinedAlerts = [
47          ...notificationAlerts.map((alert: NotificationAlert): CombinedAlert => ({
48            id: alert.id,
49            title: alert.title || 'Alert',
50            content: alert.content || alert.message || '',
51            category: alert.category || 'general',
52            created_at: alert.created_at || new Date().toISOString(),
53            user_id: alert.user_id,
54            latitude: alert.latitude,
55            longitude: alert.longitude,
56            location: alert.location,
57            isOfficial: true
58          })),
59          ...apiAlerts.map((alert: ApiAlert): CombinedAlert => ({
60            id: alert.id,
61            content: alert.content,
62            category_id: alert.category_id,
63            created_at: alert.created_at,
64            user_id: alert.user_id,
65            latitude: alert.latitude,
66            longitude: alert.longitude,
67            isOfficial: false
68          }))
69        ]
```

Readability



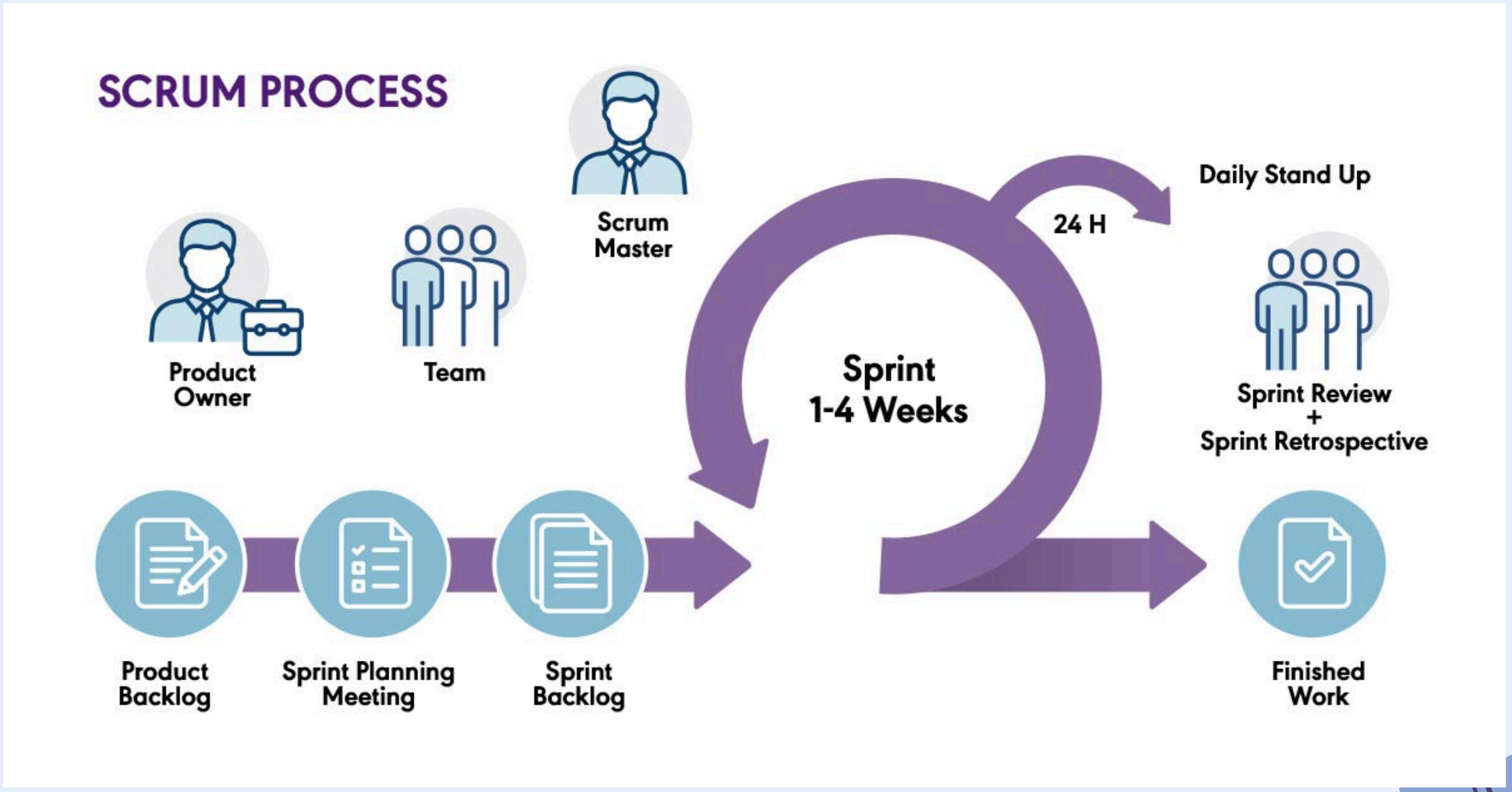
Frontend (View Layer + Service Layer)

-Service layer acts as our backend



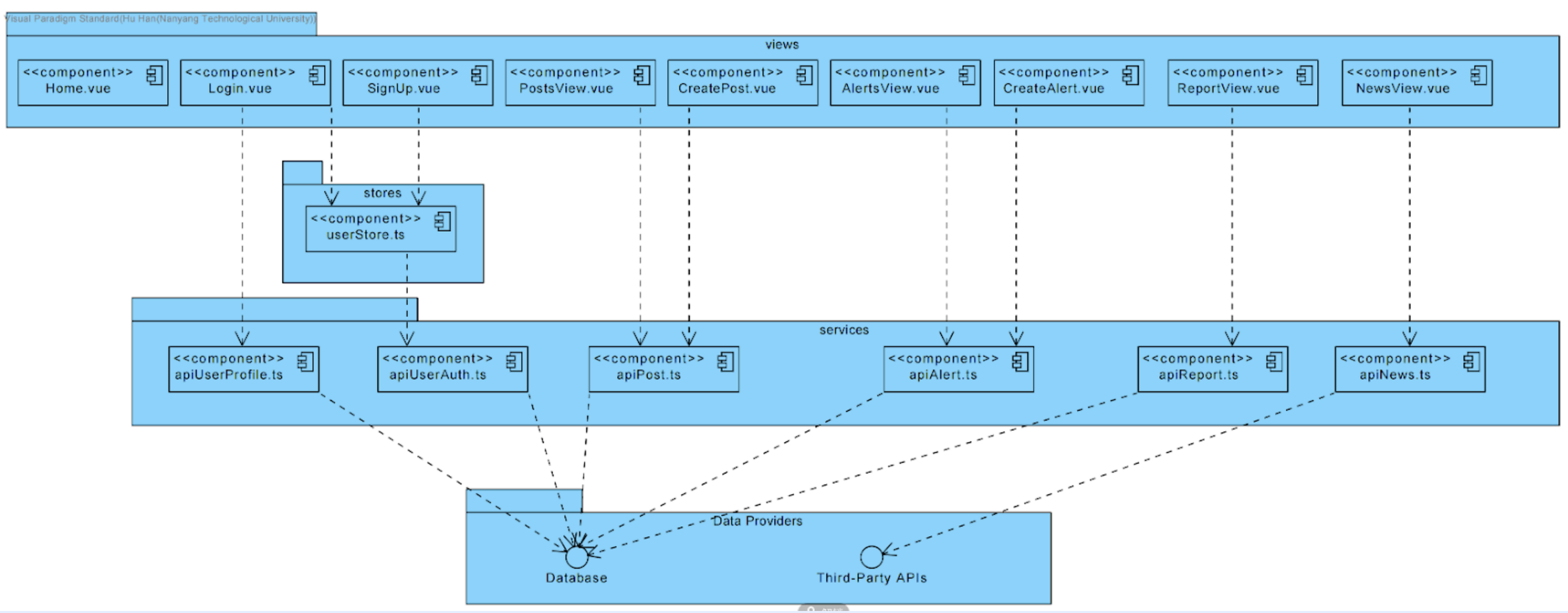
Database

Scrum

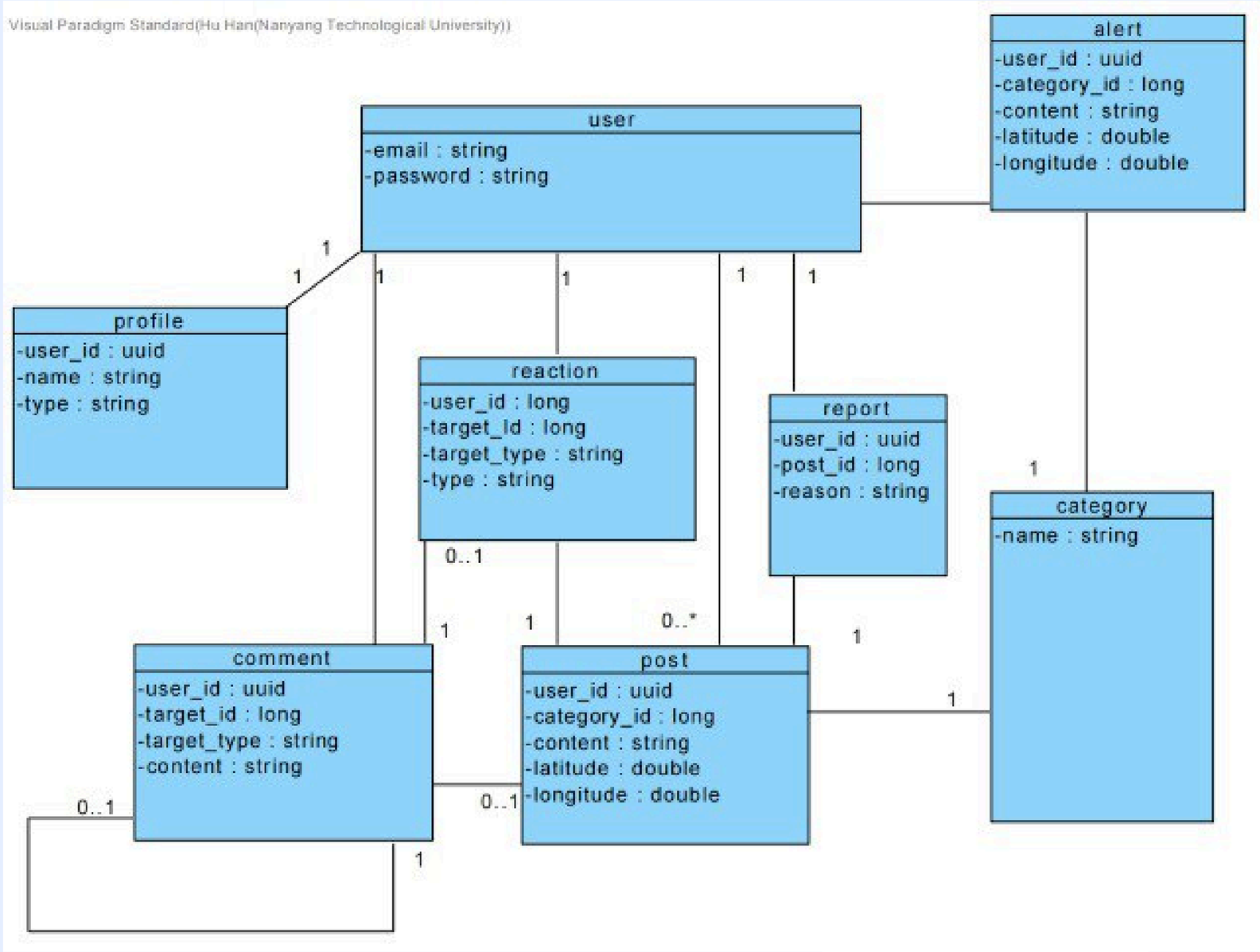

SCRUM PROCESS
Scrum Master
Product Owner
Team
Daily Stand Up
24 H
Sprint 1-4 Weeks
Sprint Review + Sprint Retrospective
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Finished Work

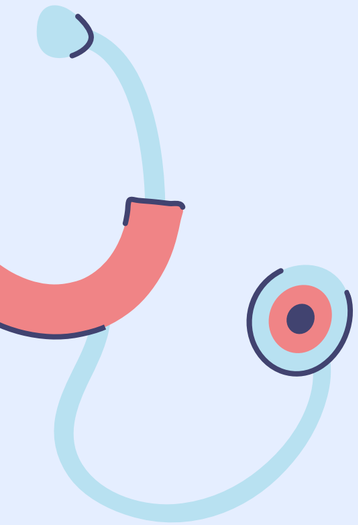
02
System Design

System Architecture



Class Diagram





03
Design Patterns

1. Respository Pattern

2. Observer Pattern

3. Strategy Pattern

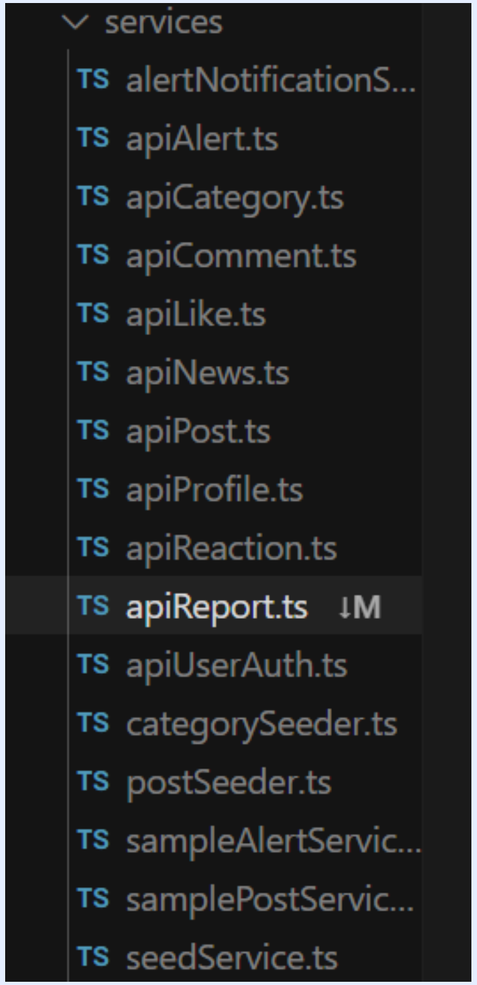
4. Module Pattern

5. SOLID principle

Respository Pattern

Implementation: Centralised data access through service modules
Examples: apiPost.ts, apiReport.ts, apiComment.ts

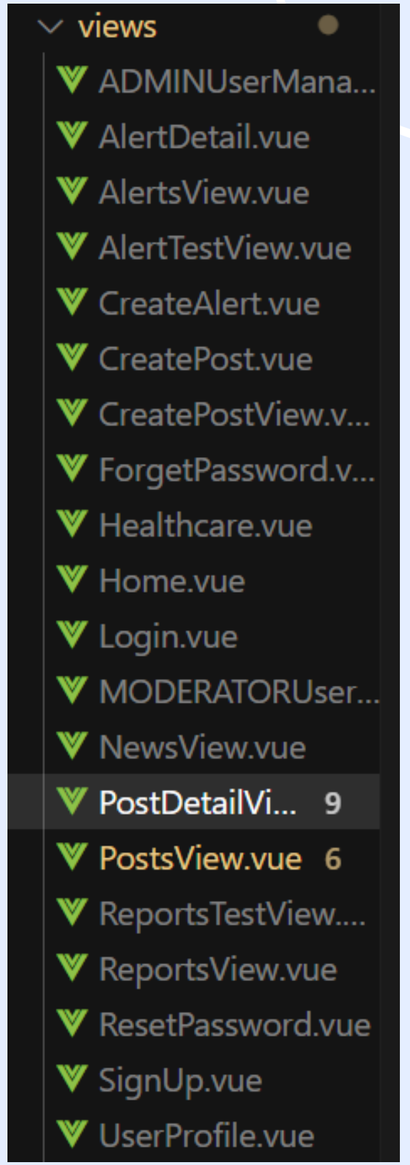
This pattern abstracts all database operations into dedicated service files, providing a clean API for the rest of the application to interact with data without knowing the underlying storage details.



Observer Pattern

Example 1: The Observer pattern, implemented through Vue's reactivity system, automatically updates the UI when data changes.
Implementation: Vue's reactive system with ref, computed, and watch

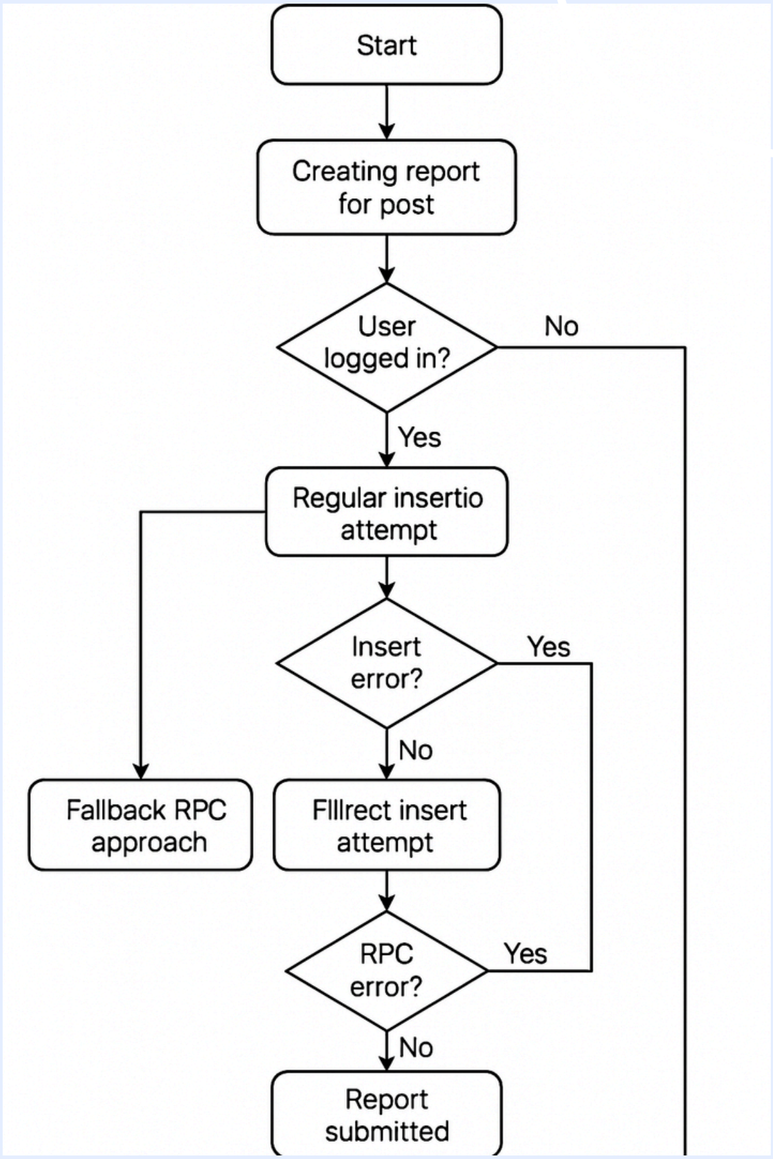
Example 2: AlertNotificationSystem.vue
Implementation: Line change subscription features provided by Supabase



Strategy Pattern

Implementation: Multiple approaches for report submission with fallbacks
Examples: Report submission logic in apiReport.ts

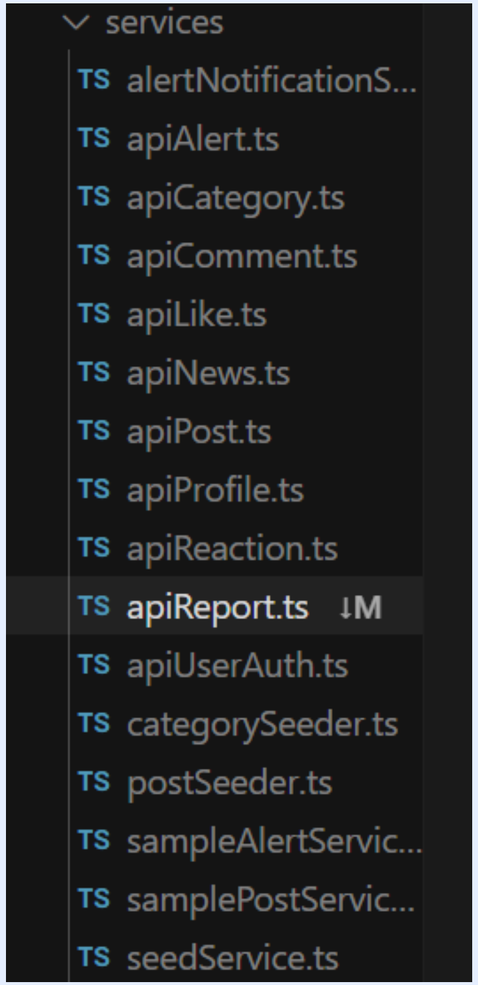
The Strategy pattern allows you to define a family of algorithms, encapsulate each one, and make them interchangeable.



Module Pattern

Implementation: Service files with private variables and exported functions
Examples: apiReport.ts, apiPost.ts, and other service files

The Module pattern encapsulates implementation details while exposing only the necessary interface.



SOLID principle

S - Single Responsible Principle

A class should have only one reason to change, meaning it should have only one responsibility or job.

Example: Each service file has a clear, focused responsibility

- **apiPost.ts** manages post data
- **apiComment.ts** manages comments
- **apiReport.ts** manages reports
- **userStore.ts** manages authentication state

SOLID principle

O - Open/Closed Principle (OCP)

Software entities (classes, modules, functions) should be open for extension but closed for modification.

Example: report submission
New submission strategies can be added without changing existing code

SOLID principle

L - Liskov Substitution Principle (LSP)

Objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program.

Example: Consistent patterns for event handling

```
// In PostsView.vue
const handlePostClick = (post: Post) => {
 // Navigate to post detail
 router.push(`/posts/${post.id}`);
}
// In AlertsView.vue
const handleAlertClick = (alert: Alert) => {
 // Navigate to alert detail
 router.push(`/alerts/details?id=${alert.id}`);
}
```

SOLID principle

I - Interface Segregation Principle (ISP)

No client should be forced to depend on methods it does not use.

Example: interfaces

- apiPost: CreatePost, UpdatePost, Post
- report-related interfaces: CreateReport, Report

SOLID principle

D - Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules. Both should depend on abstractions.

Example: component-service relationship through service imports

In PostDetailsView.vue

```
import { getPostById, type Post } from '../services/apiPost'
import { createComment, getComments } from '../services/apiComment'
import { createReport, hasReported } from '../services/apiReport
```

04

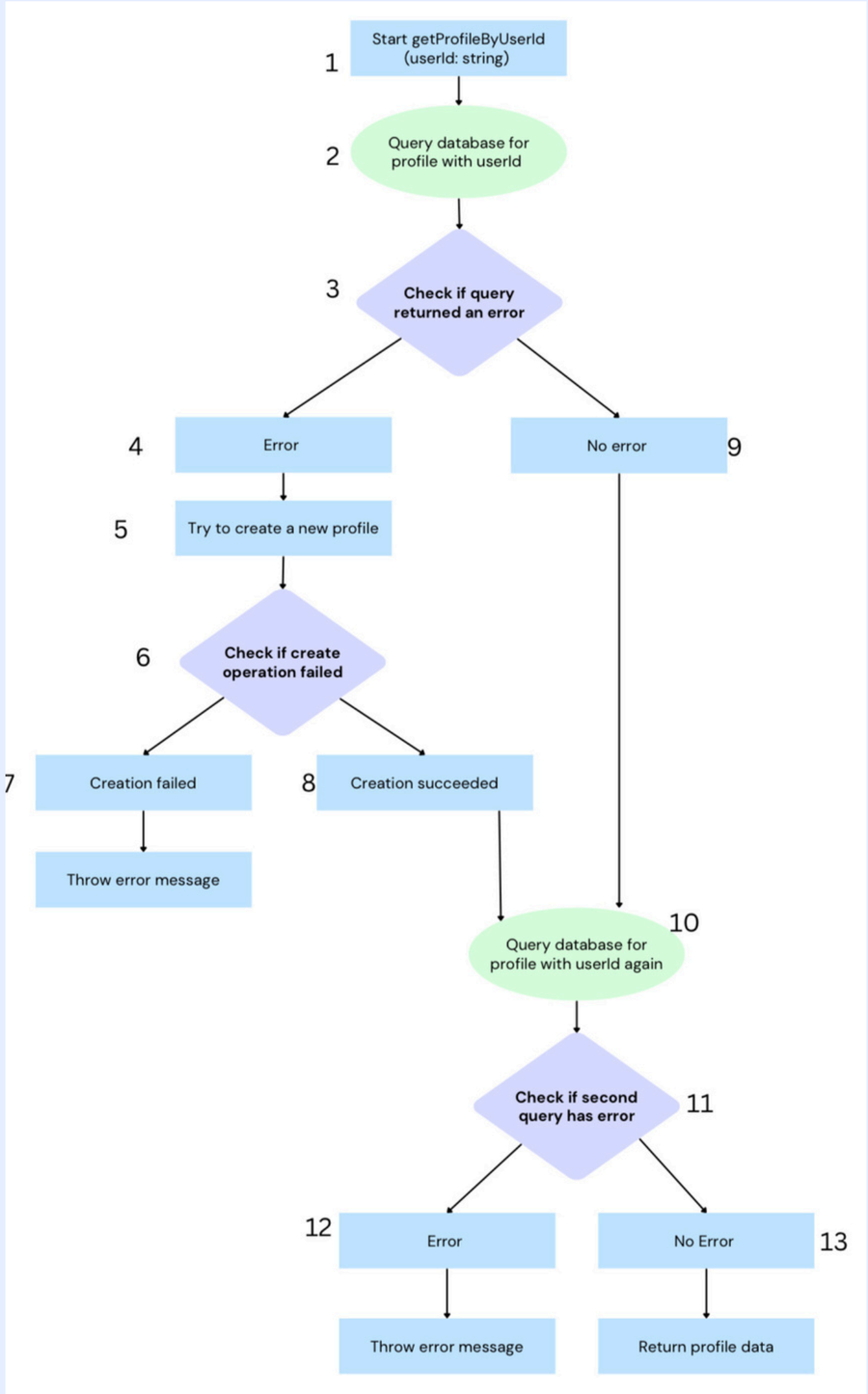
Testings

Black Box Testing

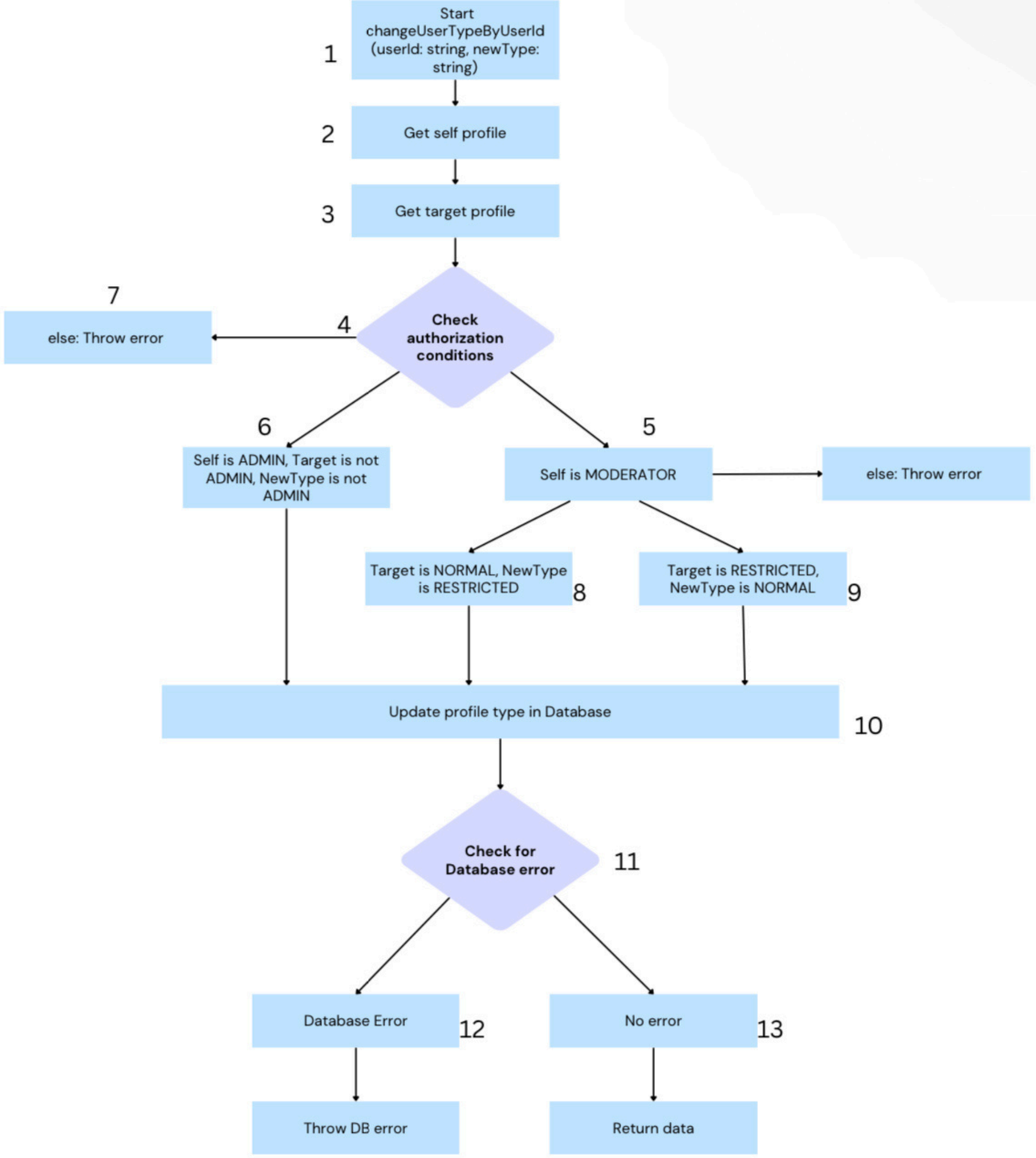
Black Box Test cases:

1. Profile Management

Test Case ID	Action	Input	Expected Output	Output	Pass?
TC2-1-1	Get profile	Current user is a new user without profile	{ id: [profileId] name: “User_”+[UserId] type: “NORMAL” }	{ id: [profileId] name: “User_”+[UserId] type: “NORMAL” }	Yes
TC2-1-2	Get profile	Current user is an existing user with an existing profile	{ id: [profileId] name: [custom_profile_name] type: “NORMAL” }	{ id: [profileId] name: [custom_profile_name] type: “NORMAL” }	Yes
TC2-2-1	Update profile name	New name with length within [1,255]	Success	Success	Yes
TC2-2-2	Update profile name	New name with name length >=256	Error	Error	Yes



White Box Testing– getProfileByUserId



White Box Testing- changeUserTypeBy UserId

Live Demo!!

Thank You