SC2006-SCSC-GRP 1

HobbyHive

Connecting people through shared experiences.

Presented by

Chua Jing Yi Jax, Ishita Dhananjaya, Swetha Sudhakar, Jain Divisha, Palagiri Afreen Mahtaj, Teo Rong Xuan

Date

13 November 2025

1

Problem, Solution & Introduction

2

Live Demo

3

Software & Design principles

4

Future Plans

HobbyHive

The Problem

In Singapore's work-focused culture, many young adults find themselves stuck in a routine:
wake up, work, commute, scroll, sleep, repeat.

After long hours at the office, it's hard to make time for hobbies, let alone find people to share them with.

Many groups are scattered across different online platforms, making discovery inconvenient and inconsistent.

As a result, many they experience social isolation, poor work-life balanace and a disconnection from their interests and community.

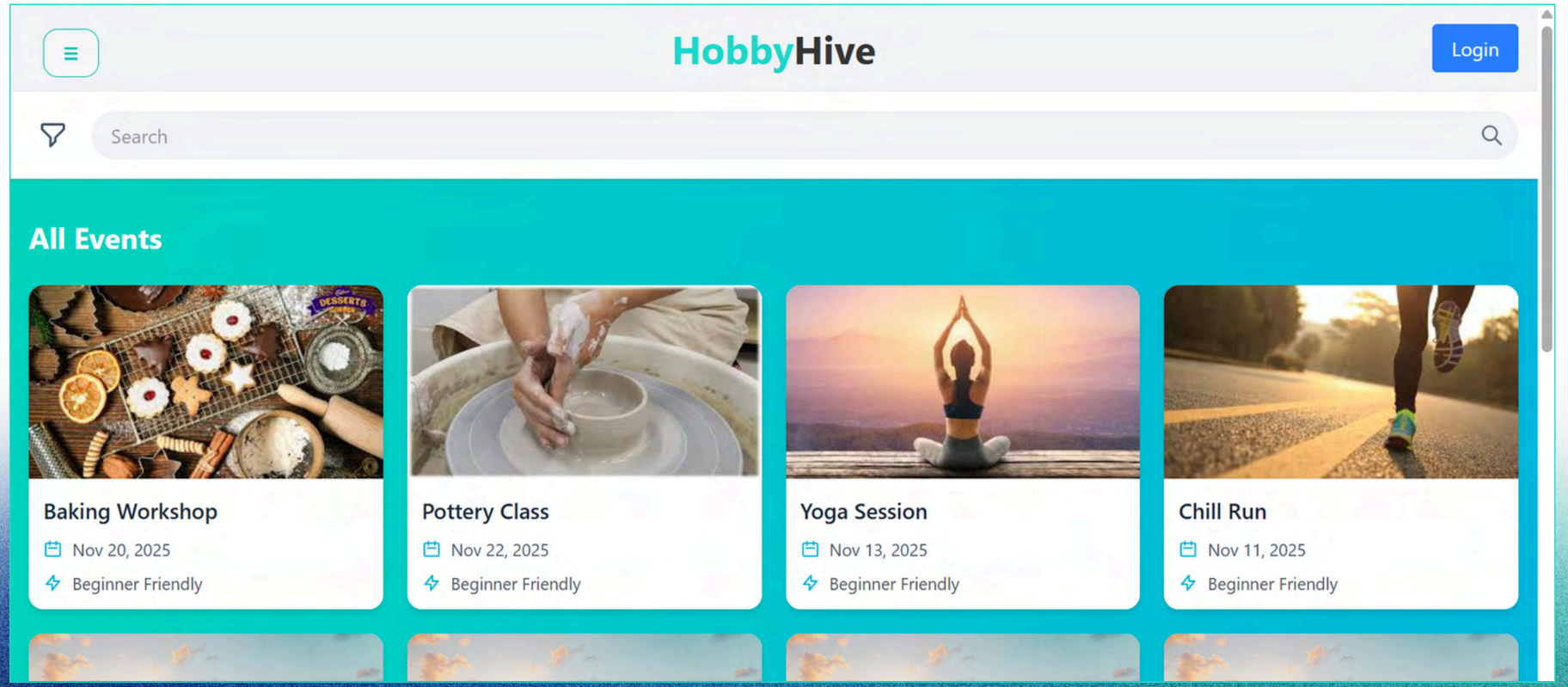
HobbyHive

Our Solution

HobbyHive is a **community-based platform** designed for young adults in Singapore to discover and pursue their hobbies

Users can:

- **Join** hobby events happening nearby
- **Connect** with individuals who share similar interests.
- **Host** their own hobby events and invite others in the neighbourhood



Expected Users

Young Working Professionals (ages 22–30)

Recently started their careers and are looking for ways to unwind after work.

University Students (ages 19–25)

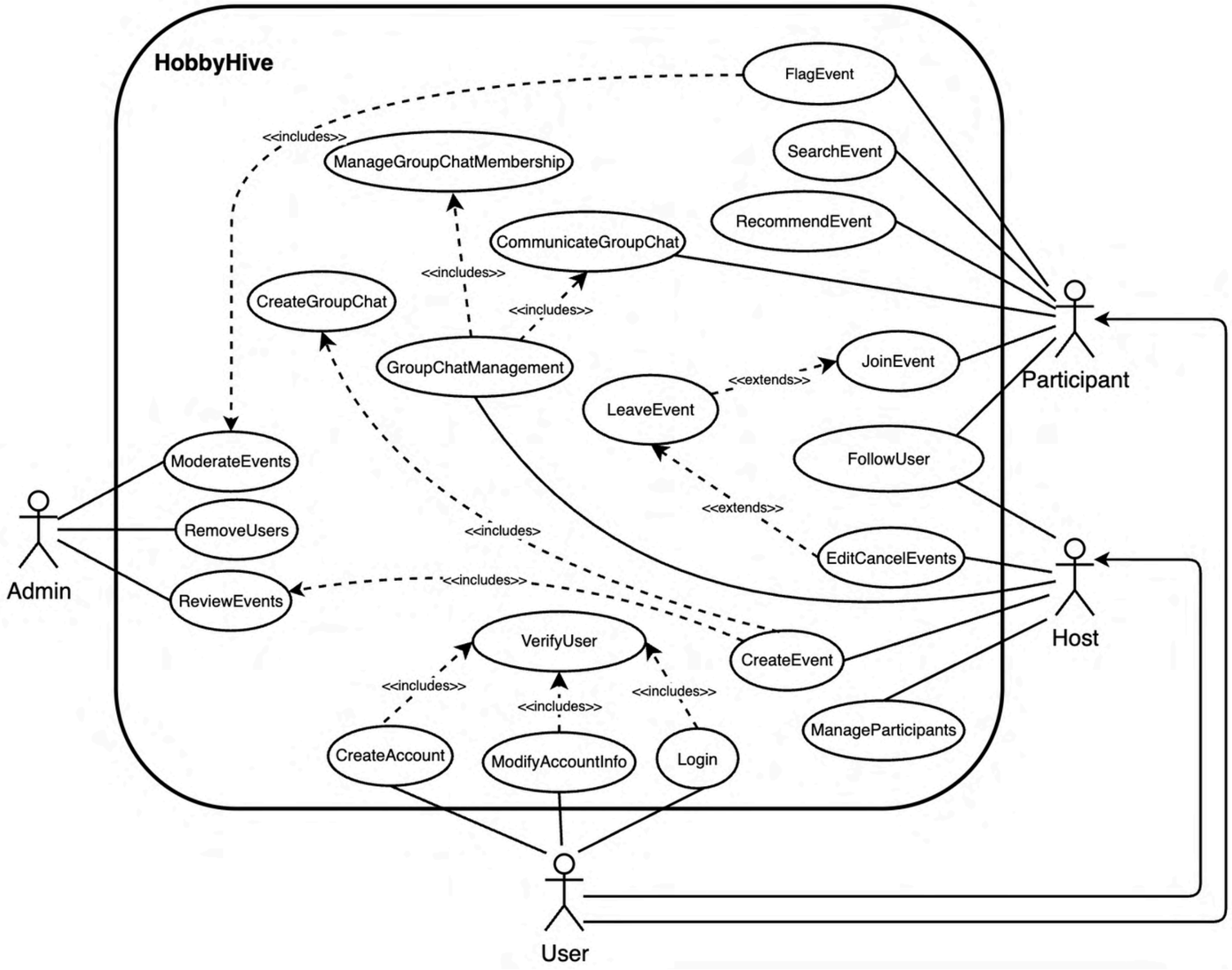
Keen to expand their social circles beyond campus.

Often on a budget, so they seek affordable community-based activities.

Young Expats new to Singapore (ages 25–35)

Use hobby-based events as a way to integrate into new communities and lifestyles.

Use Case Diagram



Key Features

Event Hosting

Host events, big or small.

Live Group chat for Events

Built-in chat system keeps everyone up to date with event.

Locations recommendation

Assist users in choosing a suitable event location.

Events recommendation

Recommend based on proximity.

Admin Moderation

Event Management

Search & Discovery

Profile Management

APIs

OneMap API
Nearest MRT API

Address geocoding
Map rendering

GEOJSON datasets

- **Community Clubs by PA**
- **Libraries by NLB**
- **Parks by NPARKS**

Location Services
Location Recommendations

Supabase Authentication API
Supabase Database API
Supabase Storage API

Secure login & User management
Events, users, real-time chat
Media uploads

Tech Stack

Frontend

Next.js
routing and API integration.

TypeScript
static typing, code reliability.

TailWindCSS
styling.

DaisyUI
Tailwind plugin for prebuilt UI components

Backend

supabase

BaaS - Provides Auth, Database, Storage, and Realtime features

PostgreSQL Database	Relational DB to store platform data
Row-Level Security (RLS)	Secure rule based access
JWT-based Authentication	Session management, secure login
Real-time WebSocket Sync	Live groupchat
Storage Buckets	Media upload
Serverless Architecture	Scalability and maintenence

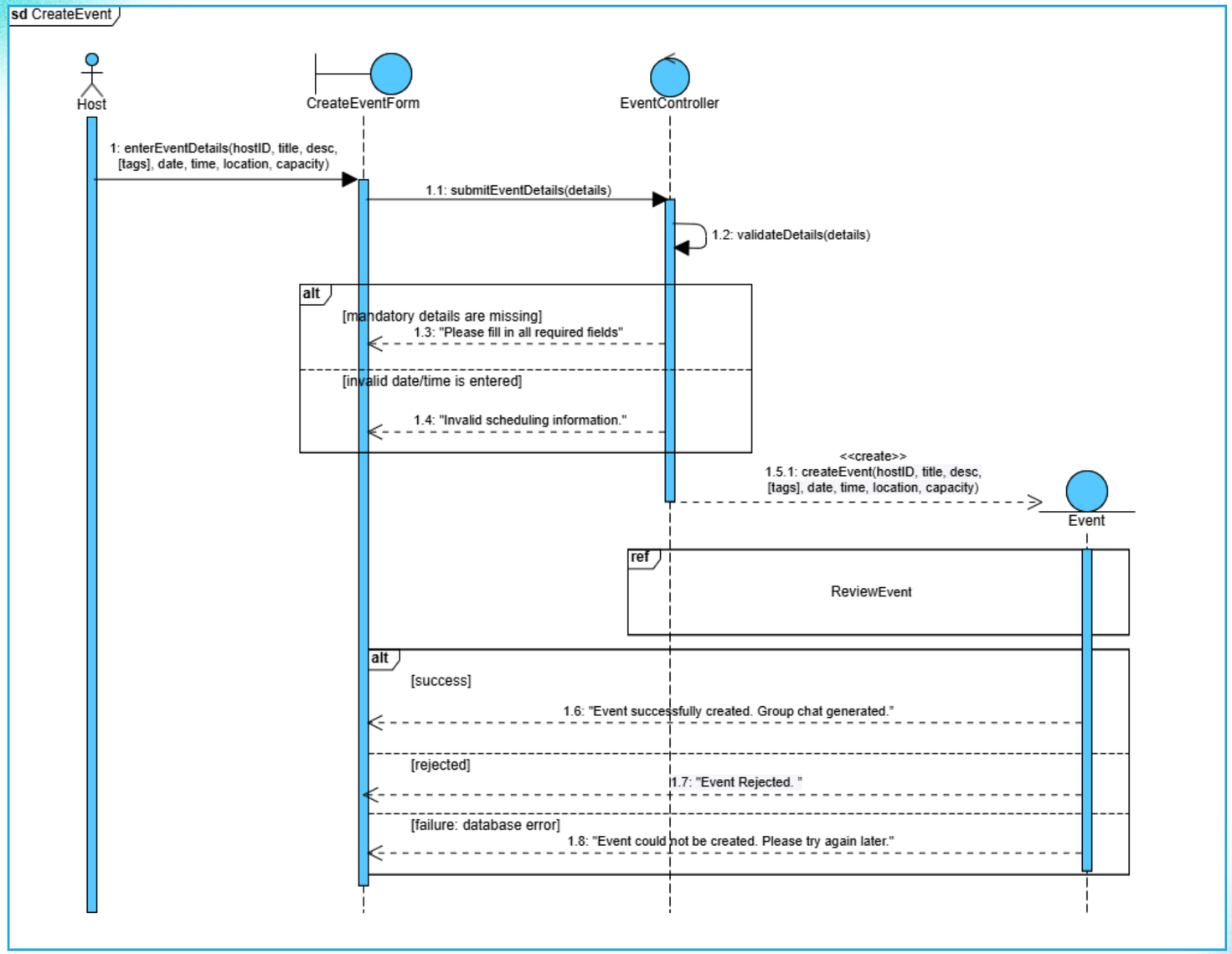
Hosting

Vercel

Official hosting platform for Next.js

Specific Use Case

Create Event



LIVE DEMO!

HobbyHive

Host

Joy

Age: 27
Interests: Swimming, Hiking, Badminton
Working full-time at Accenture

Participant

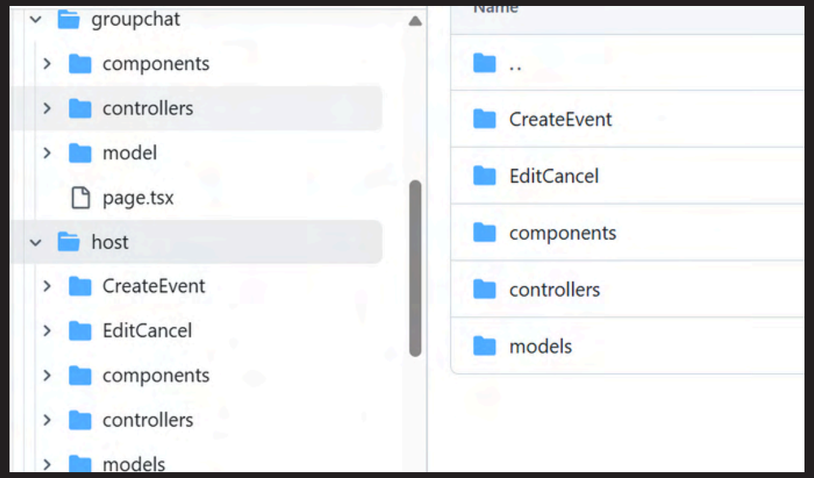
James

Age: 25
Interests: Yoga, Painting, Hiking
Working full-time at Apple

Software Engineering Practices

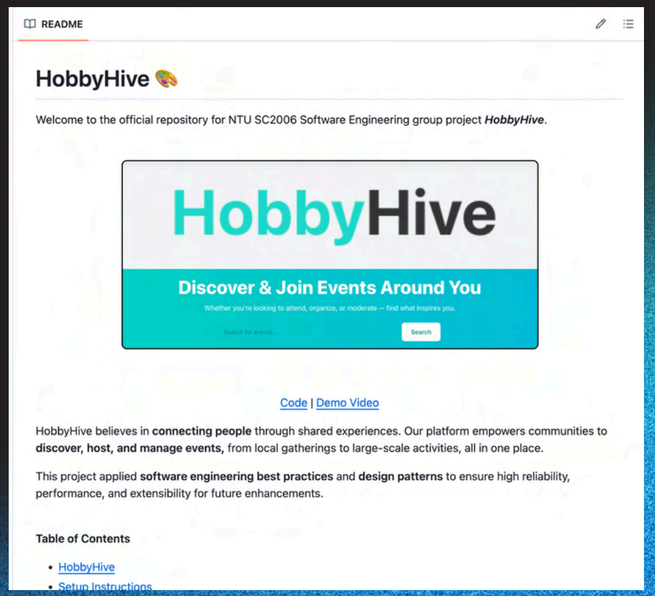
1. Clear Separation of Concerns (SoC) through MVC-style Structure

- HobbyHive separates
  - models,
  - controllers,
  - view
- This improves **maintainability, debugging, and testability.**



2. Comprehensive and Structured Documentation

- Shows that the system was **planned** and defined before implementation,
- Makes **development consistent** across team members,



Software Engineering Practices

3. User stories for main functions

Event Creation
User must first log in and navigate to the “create event” form in the hamburger bar in the top left. User will fill up event creation form details. If the date or time is invalid system will display a message: “please select a future date and time.” Once the input passes validation, event will be created and stored in the database with all submitted details with existing events. If event creation is successful, user will be able to navigate to the “Events” page and see their listing and user will be able to access the events details page.

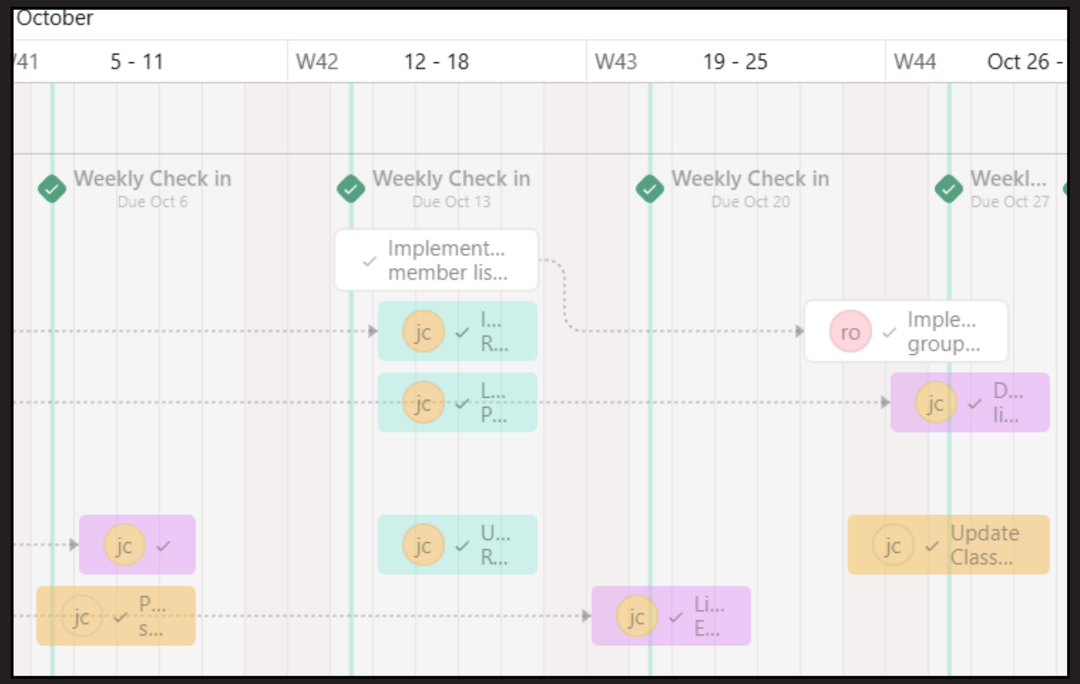
Groupchat
Users must first be logged in and must have joined an event or is hosting an event. When user creates an event, a group chat will automatically be created for the event. When user joins an event, the respective group chat will be added into their list of group chats under “Groupchats” page. Once user is assigned into a group chat, user will be able to send messages through the input bar on the bottom right. If user is the host of the event, user will be able to remove other participants by clicking on the “View members” button on the top right and click “Remove” next to their name. If user is a participant, user will be able to leave the event and groupchat by clicking on the “View members” button on the top right and click “leave” next to their name.

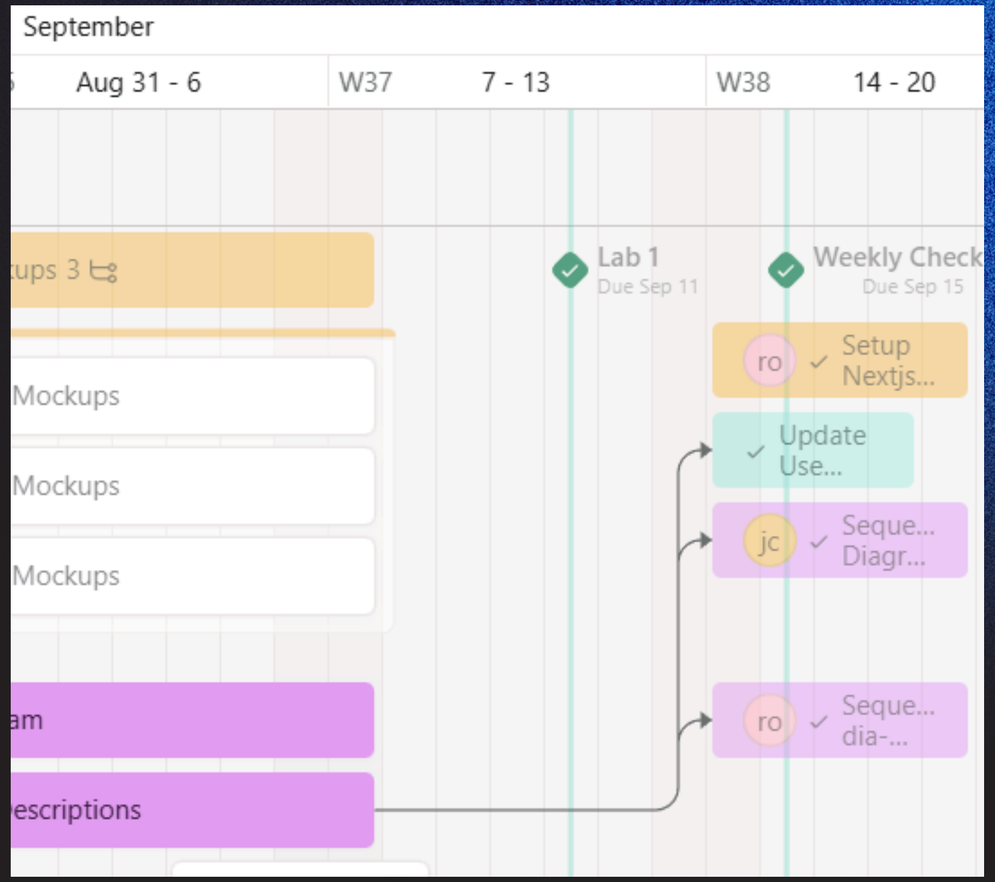
Software Engineering Practices

4. Scrum (2 weeks sprint cycle)

Scrum Master - Divisha

Product Manager - Ishita

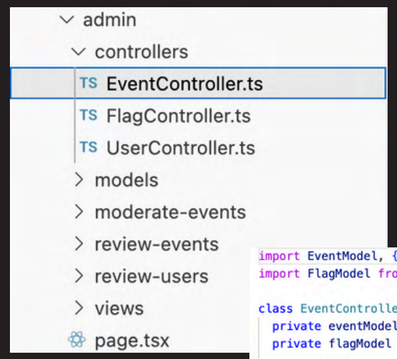




Design Patterns

1. Facade Pattern

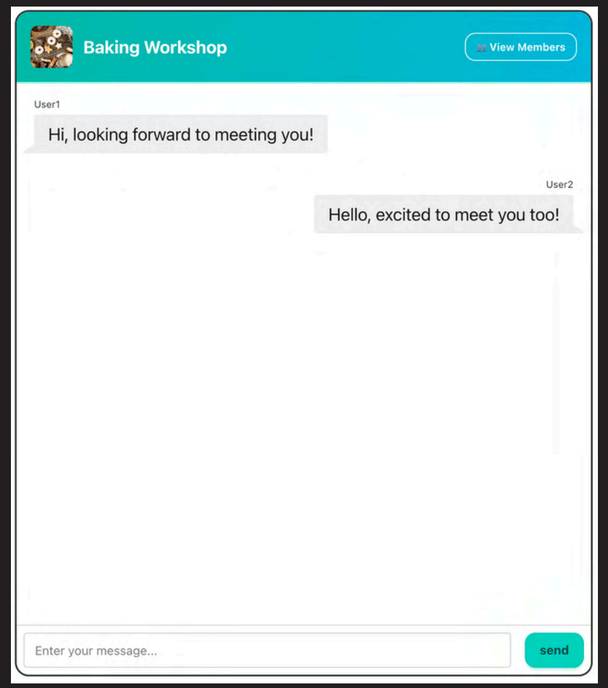
- Controllers simplify complex operations




```ts
import EventModel, { Event, FlaggedEvent } from "../models/EventModel";
import FlagModel from "../models/FlagModel";

class EventController {
  private eventModel = new EventModel();
  private flagModel = new FlagModel();

  /**
   * Load all pending events
   */
  async loadPendingEvents(): Promise<Event[]> {
    try {
      return await this.eventModel.fetchPendingEvents();
    } catch (error) {
      console.error("[EventController.loadPendingEvents] Error:", error);
      throw error;
    }
  }

  /**
   * Load all flagged events with details
   */
  async loadFlaggedEvents(): Promise<FlaggedEvent[]> {
    try {
      return await this.eventModel.fetchFlaggedEvents();
    } catch (error) {
      console.error("[EventController.loadFlaggedEvents] Error:", error);
      throw error;
    }
```

2. Publisher-Subscriber Pattern

- Real-time messaging between users



3. Singleton Pattern

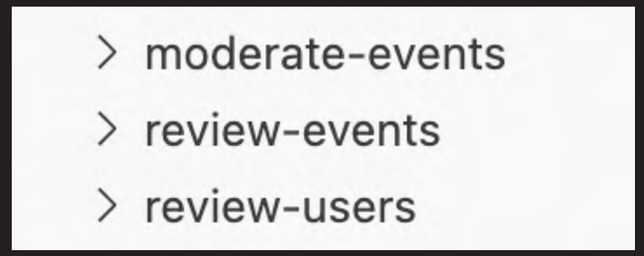
- One global OneMap authentication token

```ts
export const getInMemoryAccessToken = () => {
  if (inMemoryToken && Date.now() < tokenExpiry - 3600000) {
    return inMemoryToken;
  }
  return null;
};

export const getAccessToken = async (): Promise<string> => {
  try {
    const cachedToken = getInMemoryAccessToken();
    if (cachedToken) {
      return cachedToken;
    }

    const response = await fetch(
      `${process.env.ONEMAP_BASE_URL}/api/auth/post/getToken`,
      {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          email: process.env.ONEMAP_EMAIL,
          password: process.env.ONEMAP_PASSWORD,
        }),
      }
    );
```

SOLID Design Principles

Single Responsibility Principle (SRP)

- Each module has one clearly defined responsibility.



Open-Closed Principle (OCP)

- The application is open for extension but closed for modification.

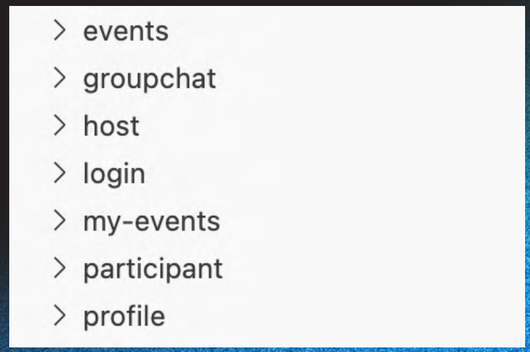
```
export const categoryLocationMapping: Record<string, CategoryLocationConfig> = {
  "Sports & Fitness": {
    allowedLocationTypes: ['parks', 'clubs'],
    primaryType: 'parks',
    showMapPicker: true,
    description: "Select a park for outdoor sports or community club for indoor activities"
  },

  "Arts & Crafts": {
    allowedLocationTypes: ['clubs'],
    primaryType: 'clubs',
    showMapPicker: true,
    description: "Select a community club with arts and crafts facilities"
  },
```

Liskov Substitution Principle (LSP)

- TypeScript interfaces ensure objects can be substituted without breaking functionality.
- Eg: Event objects from different sources (hosted/attending) use the same Event interface

Interface Segregation Principle (ISP)

- Specific, focused interfaces are used instead of large, monolithic ones.



HobbyHive

Test Cases

Test case ID (REGISTER)	Description	Input	Oracle	Log	Pass?
Test case 1	Valid signup	Username = “johnnytest” Email = “user1@gmail.com” password= “johnboy” Confirm password= “johnboy”	User successfully created; profile inserted in Supabase profiles table	User successfully created; profile row verified in profiles	Yes
Test case 2	Invalid email	Username = “johnboy225” Email = “@gmail.com” password= “hello225” Confirm password= “hello225”	“Please enter a part followed by ‘@’.’@gmail.com’ is incomplete” (unable to signup)	“Please enter a part followed by ‘@’.’@gmail.com’ is incomplete” (unable to signup)	Yes
Test case 3	Missing email	Username = “johnboy225” Email = “” password= “hello225” Confirm password= “hello225”	“Please fill out this field.” (unable to signup)	“Please fill out this field.” (unable to signup)	Yes
Test case 4	Missing password	Username = “johnboy225” Email = “johnboy@gmail.com” password= “” Confirm password= “”	“Please fill out this field.” (unable to signup)	“Please fill out this field.” (unable to signup)	Yes
Test case 5	Duplicate signup	Username = “johnnytest” Email = “user1@gmail.com” password= “johnboy” Confirm password= “johnboy”	“User already registered” (unable to signup)	“User already registered” (unable to signup)	Yes
Test case 6	Password Mismatch	Username = “johnboy225” Email = “johnboy@gmail.com” password= “hello225” Confirm password= “hello224”	“Passwords do not match” (unable to signup)	“Passwords do not match” (unable to signup)	Yes
Test case 7	Invalid password	Username = “johnboy225” Email = “johnboy@gmail.com” password= “alice” Confirm password= “alice”	“Password should be at least 6 characters” (unable to signup)	“Password should be at least 6 characters” (unable to signup)	Yes

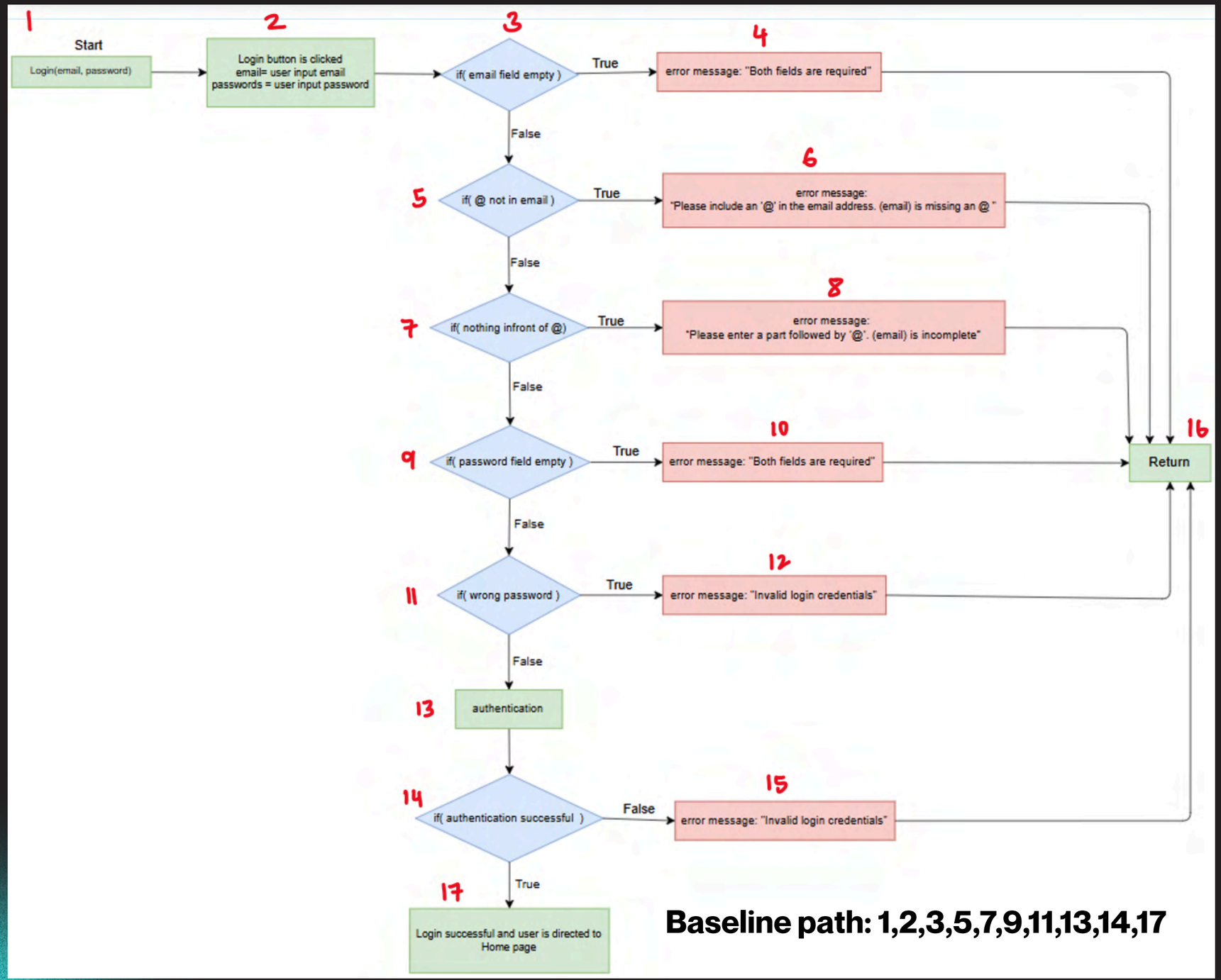
HobbyHive

Test Cases

Test case ID (LOGIN)	Description	Input	Oracle	Log	Pass?
Test case 1	Valid login	Email = "user1@gmail.com" password= "johnboy"	Successful login; Supabase returns user object	Successful login; user object returned	Yes
Test case 2	Invalid email (nothing in front of @)	Email = "@gmail.com" password= "johnboy"	"Please enter a part followed by '@'.'@gmail.com' is incomplete" (unable to login)	"Please enter a part followed by '@'.'@gmail.com' is incomplete" (unable to login)	Yes
Test case 3	Invalid email (without @)	Email = "user" password= "johnboy"	"Please include an '@' in the email address. 'user1 is missing an '@'" (unable to login)	"Please include an '@' in the email address. 'user1 is missing an '@'" (unable to login)	Yes
Test case 4	Missing email	Email = "" password= "johnboy"	"Both fields are required." (unable to login)	"Both fields are required." (unable to login)	Yes
Test case 5	Missing password	Email = "user1@gmail.com" password= ""	"Both fields are required." (unable to login)	"Both fields are required." (unable to login)	Yes
Test case 6	Wrong password	Email = "user1@gmail.com" password= "123456"	"Invalid login credentials" (unable to login)	"Invalid login credentials" (unable to login)	Yes
Test case 7	Unregistered email	Email = "user2@gmail.com" password= "hello225"	"Invalid login credentials" (unable to login)	"Invalid login credentials" (unable to login))	Yes

HobbyHive

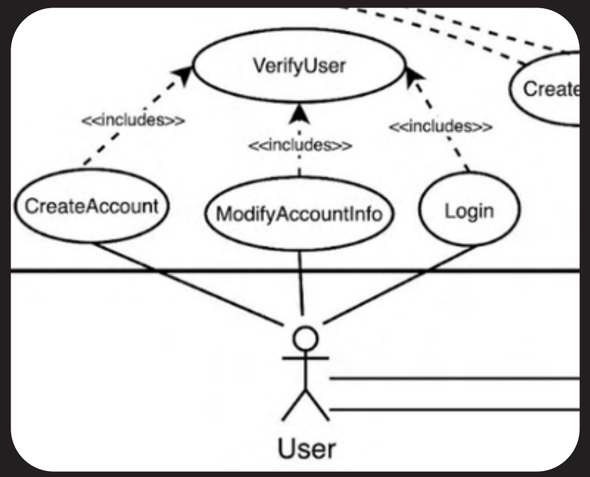
White Box

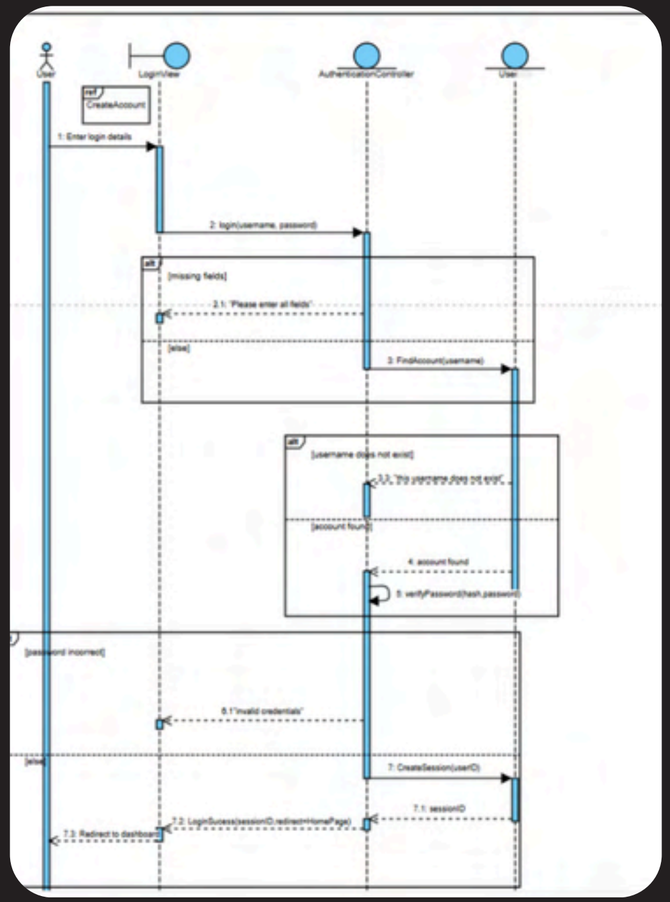


Baseline path: 1,2,3,5,7,9,11,13,14,17

HobbyHive

Traceability - Login Feature






```
14      const handleSubmit = async (e: React.FormEvent) => {
27        const { data, error: loginError } = await supabase.auth.signInWithPassword({
28          email,
29          password
30        });
31
32        if (loginError) {
33          setLoading(false);
34          setError(loginError.message);
35          return;
36        }
37
38        // Step 2: Check if user is banned
39        console.log("[Login] Checking ban status for user:", data.user.id);
40        const { data: profile, error: profileError } = await supabase
41          .from("profiles")
42          .select("is_banned, ban_reason, banned_until, role")
43          .eq("id", data.user.id)
44          .single();
```

```
14      const handleSubmit = async (e: React.FormEvent) => {
100       // Step 4: If not banned (or ban expired), check role and redirect
101       console.log("[Login] User is not banned, checking role...");
102
103       // Check if user is admin
104       if (profile.role === "admin") {
105         console.log("[Login] Admin user, redirecting to admin dashboard");
106         setLoading(false);
107         router.push("/admin");
108       } else {
109         console.log("[Login] Regular user, redirecting to homepage");
110         setLoading(false);
111         router.push("/");
```

```
41    export async function login(email: string, password: string) {
43      const { data, error } = await supabase.auth.signInWithPassword({ email, password });
```

Future Plans

1

AI-Powered Event Summaries

- Automatically generate **short summaries** of past group chats and events.
- Helps users recall key discussions and maintain event continuity.
- Uses **AI text summarization** to extract relevant highlights.

2

Cross-Platform Chat Integration

- **Sync group chat messages** across Telegram and WhatsApp.
- Increases **accessibility** for users on multiple platforms.
- Enables seamless communication without needing **multiple logins.**

Thank You!