

DESIGN SPECIFICATION

Project: THE HIVE

Group Name: Fantastic 4

Andaç Bilgili (Team Lead/UX)

Gülsah Öykü Kirli (DevOps & Testing)

Mustafa Efe Arslan (Frontend Developer)

Yigit Aydogan (Backend Developer)

Course: Software Engineering

Date: December 14, 2025

1. INTRODUCTION

1.1 Goal

The goal of 'The Hive' is to address the fragmentation of campus event announcements at ITU by creating a centralized web application. This platform allows students to find all upcoming school events in one place, supporting student engagement and participation in campus life. The system aggregates data through two primary methods: manual entry by authorized club administrators and automated scraping of public Instagram accounts.

1.2 Organization

This document details the system design for 'The Hive.' It is organized into the following sections:

- **System Architecture:** High-level architectural style and component decomposition.
- **Low Level Design:** Detailed class structures, interaction sequences for key use cases, and data flow details.

2. SYSTEM ARCHITECTURE

2.1 System Architecture

The Hive utilizes a Client-Server Architecture (specifically a Three-Tier Web Architecture) to separate presentation, processing, and data management.

- **Presentation Layer (Client):** A responsive web application (SPA) accessible via mobile and desktop browsers.
- **Application Layer (Backend):** A RESTful API that handles business logic, authentication, and search queries. It includes a specialized background worker for the Instagram scraping component.
- **Data Layer:** A relational database (PostgreSQL) for persistent storage of event, club, and student data.

2.2 Component (Package) Diagram

The system is decomposed into four main components. Inputs, outputs, and services for each are listed below:

1. Frontend Component (UI)

Inputs: User clicks, Search strings, Filter selections.

Outputs: HTTP Requests to API, Local Display.

Services: Event Display, Search Interface, Reminder Settings, Login Forms.

2. Backend API Component

Inputs: HTTP Requests (GET/POST/PUT).

Outputs: JSON responses, HTTP Status Codes.

Services: Authentication (authClub), Event Management (saveEvent), Query Processing (getEvents).

3. Instagram Scraper Worker

Inputs: Whitelisted Instagram URLs, Cron Schedule.

Outputs: 'Draft' Event Data to Database.

Services: Page Fetching, HTML Parsing, Metadata Extraction.

4. Database Component

Inputs: SQL Queries (Insert, Select, Update).

Outputs: Data Rows / Result Sets.

Services: Data Persistence, Relational Integrity, Concurrency Control.

3. LOW LEVEL DESIGN

3.1 Class Diagrams

A class diagram for each component is defined below:

Class: EventView (Frontend Component)

- Attributes: currentFilter, searchString, eventList[]
- Methods: renderList(), updateSearch(), toggleReminder(eventID)

Class: Event (Backend Component)

- Attributes: eventID (PK), title, date, location, description, status [Draft, Published], source [Manual, Scrapped]
- Methods: publish(), archive(), validate()

Class: Club (Backend Component)

- Attributes: clubID (PK), name, instagramURL, password (Hashed)
- Methods: createEvent(details), login()

Class: ScraperService (Worker Component)

- Attributes: targetURLs, scrapeInterval
- Methods: fetchPage(url), extractMetadata(html), logFailure(error)

3.2 Sequence Diagrams

Sequence diagrams for the four primary Use Cases:

Use Case 1: Event Discovery (Search)

- 1 Student navigates to Home Page.
- 2 Student enters 'Search Keyword' and selects filters.
- 3 Frontend sends GET /api/events request to Backend.
- 4 Backend cleans search string and constructs SQL query.
- 5 Database returns matching 'Published' records.

- 6 Backend formats records into JSON array and returns to Frontend.
- 7 Frontend renders the Event List.

Use Case 2: Automated Aggregation (Scraping)

- 1 Scraper Worker triggers on schedule (Cron Job).
- 2 Worker retrieves target URLs from Database.
- 3 Worker requests public profile HTML from Instagram.
- 4 Worker parses content for Title, Date, and Time.
- 5 Worker creates new Event object with status='Draft'.
- 6 Worker saves Event to Database.
- 7 System Admin later approves the draft to 'Published'.

Use Case 3: Manual Event Creation

- 1 Club Admin logs into the system.
- 2 Admin fills out the 'New Event' form and clicks Submit.
- 3 Frontend validates inputs locally and sends POST /api/events.
- 4 Backend validates session and data integrity.
- 5 Backend saves the new Event to Database with status='Pending'.
- 6 Database returns success confirmation.
- 7 Backend returns '201 Created' response to Frontend.

Use Case 4: Reminder Setup

- 1 Student views an Event detail page.
- 2 Student clicks 'Remind Me' button.
- 3 Frontend sends POST /api/reminders with EventID.
- 4 Backend validates Student session.
- 5 Backend saves [Student_ID, Event_ID] pair to Database.
- 6 Backend returns success message.
- 7 Frontend updates UI to show 'Reminder Set'.

3.3 Data Flow Diagram (Level 2)

Function 1: Event Discovery

This describes the internal transformation of data when a student searches for an event.

- **1.1 Validate Inputs:** Checks if search string and dates are valid.
- **1.2 Build Query:** Combines valid inputs into a SQL query string (status='Published').
- **1.3 Execute Query:** Runs query against the Events Database.
- **1.4 Format Output:** Converts raw DB rows into JSON response for the Student.