Project 3: HomeStand

Software Architecture Document

Version 1.5

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 25/10/2025 | 1.0 | Compilation of Notes | Carlos Mbendera |
| 25/10/2025 | 1.1 | User Authentication and Events Handling UML diagrams | Mahdi Essawi |
| 25/10/2025 | 1.2 | Abstract UML diagram | Blake Ebner |
| 26/10/2025 | 1.3 | Logical overview 2.1, 2.2, and 4.2  Updated Table of Contents | Cole DuBois |
| 26/10/2025 | 1.4 | Process overview | Jonathan Johnston |
| 26/10/2025 | 1.5 | Project Introduction finishing touches. | Yoseph Ephrem |

Table of Contents

1. Introduction 4

1.1 Purpose

1.2 Project Synopsis

1.3 Scope 4

1.4 Definitions, Acronyms, and Abbreviations 4

1.5 References 4

1.6 Overview 4

2. Architectural Representation 5

3. Architectural Constraints 7

4. Logical View 7

4.1 Overview 7

4.2 Architecturally Significant Design Packages 8

5. Development Description 10

5.1 Overview 10

5.2 Time Taken 10

5.3 Time Tracking Tools 11

5.4 Meeting Notes 12

Software Architecture Document

# Introduction

This Software Architecture Document (SAD) outlines the architectural framework and high-level design for *HomeStand*, a web-based event hosting and community platform as part of the EECS 581: Software Engineering II coursework at the University of Kansas. The goal of HomeStand is to bring soccer fans together by allowing users to host or join match-day events at local venues such as restaurants, bars, or parks, creating a more social and engaging viewing experience.

The architecture follows a service-oriented and modular design built around core services such as user management, event scheduling, social engagement, and payment handling. The structure ensures that our platform remains scalable and maintainable for potential future expansion.

## Purpose

* The purpose of this document is to present a comprehensive architectural overview of HomeStand, defining its structure, organization, and interactions among key subsystems. It highlights architectural decisions related to data flow, modularity, and scalability, forming the foundation for implementation and future development. This document is intended for both developers and stakeholders. For developers, it defines how services and modules interconnect like the authorization, event and payment services; for stakeholders, it demonstrates how the system’s requirements are translated into a cohesive and maintainable design.

## Project Synopsis

A web-based app that connects soccer fans enabling users to host, discover, and attend local match-day events, enriched with news and live stats.

* 1. **Scope**
* The scope of this document encompasses the complete architectural design of HomeStand, covering user account management, event creation and discovery, and integration of soccer-related content such as match news and statistics. The document also introduces preliminary design concepts for the system’s core modules like event scheduling, user interaction, and data management. The architecture is designed with future scalability in mind, allowing for later expansion into features such as location-based event suggestions, live match updates, and mobile compatibility.

## Definitions, Acronyms, and Abbreviations

* **SRS**: Software Requirements Specification
* **API**: Application Programming Interface
* **UI**: User Interface
* **Firebase**: A cloud-based, NoSQL database platform designed for app development, offering real-time data storage and synchronization.
* **Streamlit**: An open-source Python library for building interactive web apps using only Python. It's ideal for creating interactive user interfaces without needing HTML, CSS or JavaScript.
* **UML:** Unified Modeling Language

## References

* EECS 348 Software Engineering Course Syllabus, Professor Hossein Saiedian, University of Kansas, Spring Semester 2024.
* EECS 581: Software Engineering II Requirements Artifacts, Professor Johnson, David, University of Kansas, Fall 2024.
* EECS 581: Software Engineering II Initial Architecture Document, Professor Johnson, David, University of Kansas, Fall 2024.
* EECS 348 Project: ShowTracker, DMH, University of Kansas, Spring Semester 2024
* ChatGPT

## Overview

This document covers our software design architecture for HomeStand. It will touch on various items such as size, interface, and structure. It covers:

* **Architectural Representation:** UML diagrams illustrating key components, data flow, and interactions.
* **Architectural Goals and Constraints:** Primary design objectives guiding system development.
* **Logical and Component Views:** Description of modules and their relationships.
* **Interface Description:** High-level outline of communication among subsystems.
* **Development Description:** Summary of the team’s collaboration and implementation process.

# Architectural Representation

The software architecture for the system encompasses multiple views that collectively provide a comprehensive understanding of the system's structure and behavior. Each view focuses on specific aspects of the architecture guiding the development and evolution of the system.

**1. Overview:**

**Description**: A high-level overview of the platform's service-oriented components and their interactions.

**Model Elements:**

* Core Runtime: An API-driven, request-response server that manages sessions, processes requests via service classes, and handles data persistence.
* State Collections:
  + Core Data Entities: User (with AdminUser/RegularUser roles), Event, Game, Payment, Chat,Calendar, News.
  + Persistence Layers (Implied): userDB, eventsDB.
* Functional Blocks:
  + User Management & Security: Authorization (login/register), User (profiles/roles), HashAlg (password security).
  + Event & Scheduling: EventService (create, find, join), Event (core data), Calendar (reminders).
  + Payment Processing: PaymentService (handles transactions) via paymentGateway, Payment (transaction data).
  + Social & Content: Chat (messaging), News (articles), Search (querying).
  + Interaction Handling (Implied): API endpoints routing user actions (e.g., joinEvent, submitText) to services.

**2. Logical View:**

**Description**: Describes the system's logical modules for user management, event logistics, payments, and social features.

**Model Elements:**

* User Management & Authentication
  + User: Base class for AdminUser and RegularUser, holding core profile data.
  + Authorization: Handles login, registration, and session logic.
  + HashAlg: Utility component for password hashing used by Authorization.
* Event & Game Logic
  + Event: Central entity holding all event details (date, location, price, capacity).
  + EventService: Manages the business logic for creating, finding (getNearbyEvents), and canceling events.
  + Game: An optional entity associated with an Event.
* Payment Processing
  + Payment: Data model representing a transaction, linked to an Event.
  + PaymentService: Handles all payment logic (handlePayment, verifyTransaction) through an external paymentGateway.
* User Engagement & Social Features
  + Chat: Provides user-to-user messaging functionality.
  + Calendar: Manages a user's personal event schedule and reminders.
  + Search: Service for querying Events and News.
  + News: Content model for articles and system updates.

**Key Interface Points & Relationships:**

* Service Layer: Primary application logic is handled by the Authorization, EventService, and PaymentServiceclasses.
* External Dependencies: The system relies on an external paymentGateway and persistence layers (userDB, eventsDB).
* Core Relationships:
  + A User can create or join multiple Events.
  + An Event can have multiple Payments.
  + A User interacts with Chat, Calendar, and Search.

**3. Process View**

**Description**: Shows the interactions between components of the software architecture. This models the process flow.

**Process Elements:**

* **Sign in and authorization:**
  + If user doesn’t have an account yet then allow for user to create account and update database with user login information
  + A hash algorithm is used to ensure safe storage of user information
  + When user tries to login check entered password against password database
  + If authentication succeeds, allow user access to app features.
* **Events:**
  + Information about users is used to show relevant events to users.
  + The event service module is used for creating, joining, and searching events.
  + If events require payment then allow users to use payment service.
* **Newsfeed:**
  + Information about users is used to show relevant news to users.
  + Allow for addition of new news articles.
* **Chat:**
  + Users can send messages to other users.
  + When a message is sent, deliver it to the other user.
  + Notify the user that they have received a message.

* **Data Flow:**
  + Sign in/create account → Authorization:
    - Events:
      * Create events → update event data base
      * Join events → update event data base
      * Search for events → filter event data base
      * Payments → Payment processing
    - Newsfeed:
      * Read articles
      * Post articles → update news database
    - Chat:
      * Message another user → deliver message → notify recipient of the message

4. **Development View**

**Description**: Focuses on the organization of the development effort, including the structure of source code, version control, and development environments.

**Model Elements:**

* **Code Layout:** Multiple scripts representing different backend and frontend functions such as AuthBackend.py, AuthFrontEnd.py and more.
* **Version Control**: Git / GitHub.
* **Tooling & Environment**: Visual Studio Code (Python + Firebase + Streamlit extensions), standard Python interpreter.
* **Process & Workflow**: Incremental feature layering ( Authentication → Events → UI polish).

# Architectural Constraints

* **Developmental tools**: We will utilize Visual Studio Code and PyCharm for its versatility, Git and GitHub for version control and PyCharm / VSCode with the Python Extension, ensuring a robust development environment.
* **Design:** Object Oriented Programming - to facilitate clear modularity and extensibility of the software components.
* **Usability**: Develop an intuitive user interface and solid technical backend to enhance the development experience via a modular architecture.
* **Reuse**: Architect the codebase to promote code reuse and modularity for easier maintenance and scalability.
* **Off-the-shelf product:** Integrate third-party APIs or libraries for features such as content recommendation
* **Design and implementation strategy:** We plan to follow an incremental feature layering methodology, to ensure phases and requirements are met in a structured progression on a sprint by sprint basis.

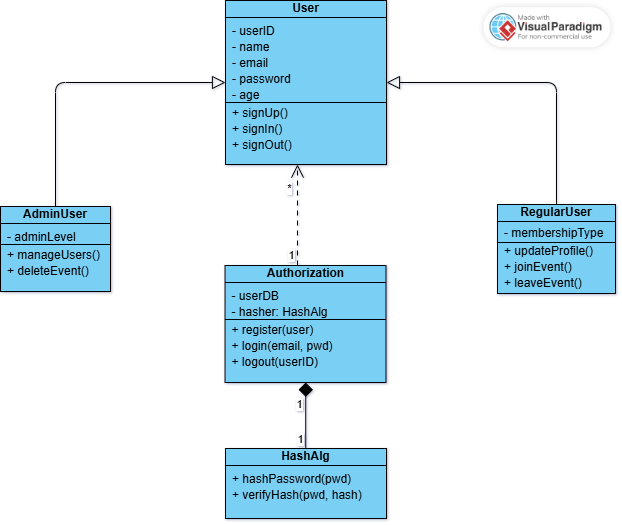
# Logical View

## Overview

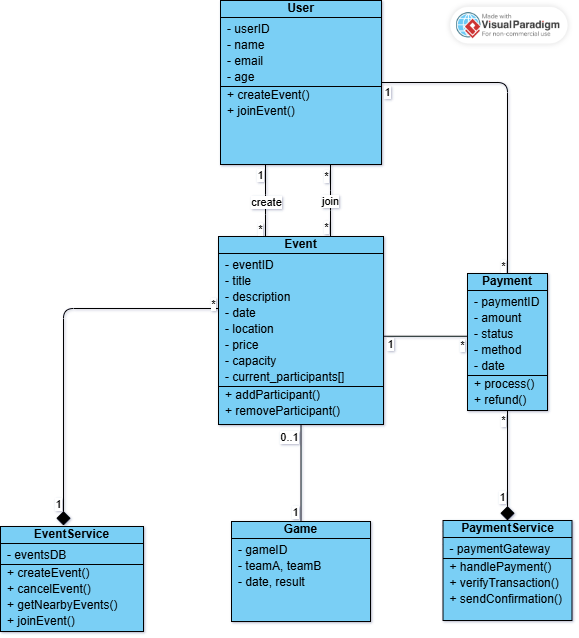
The overall design of HomeStand is divided into various major subsystems that interact to deliver the application's functionality. These are the:

* **User & Authentication Service:** Manages user accounts, roles (AdminUser, RegularUser), and handles security via Authorization and HashAlg.
* **Event & Scheduling Logic:** Central services (EventService) for creating, finding (getNearbyEvents), and managing Event and Game entities.
* **Payment Processing:** Handles all transactions (Payment) and payment logic (PaymentService) via an external paymentGateway.
* **Social & Engagement Features:** Provides user-facing modules like Chat, Calendar, Search, and News.
* **Core Data Models:** The main entities representing the application's state (User, Event, Payment, News, Game).
* **Input & Service Layer (Implied)**: Dispatches user requests (e.g., API calls, UI interactions) to the correct business logic in EventService, PaymentService, etc.
* **External Interfaces (Implied):** Connectors to third-party systems, specifically the paymentGateway and persistence layers (eventsDB, userDB).

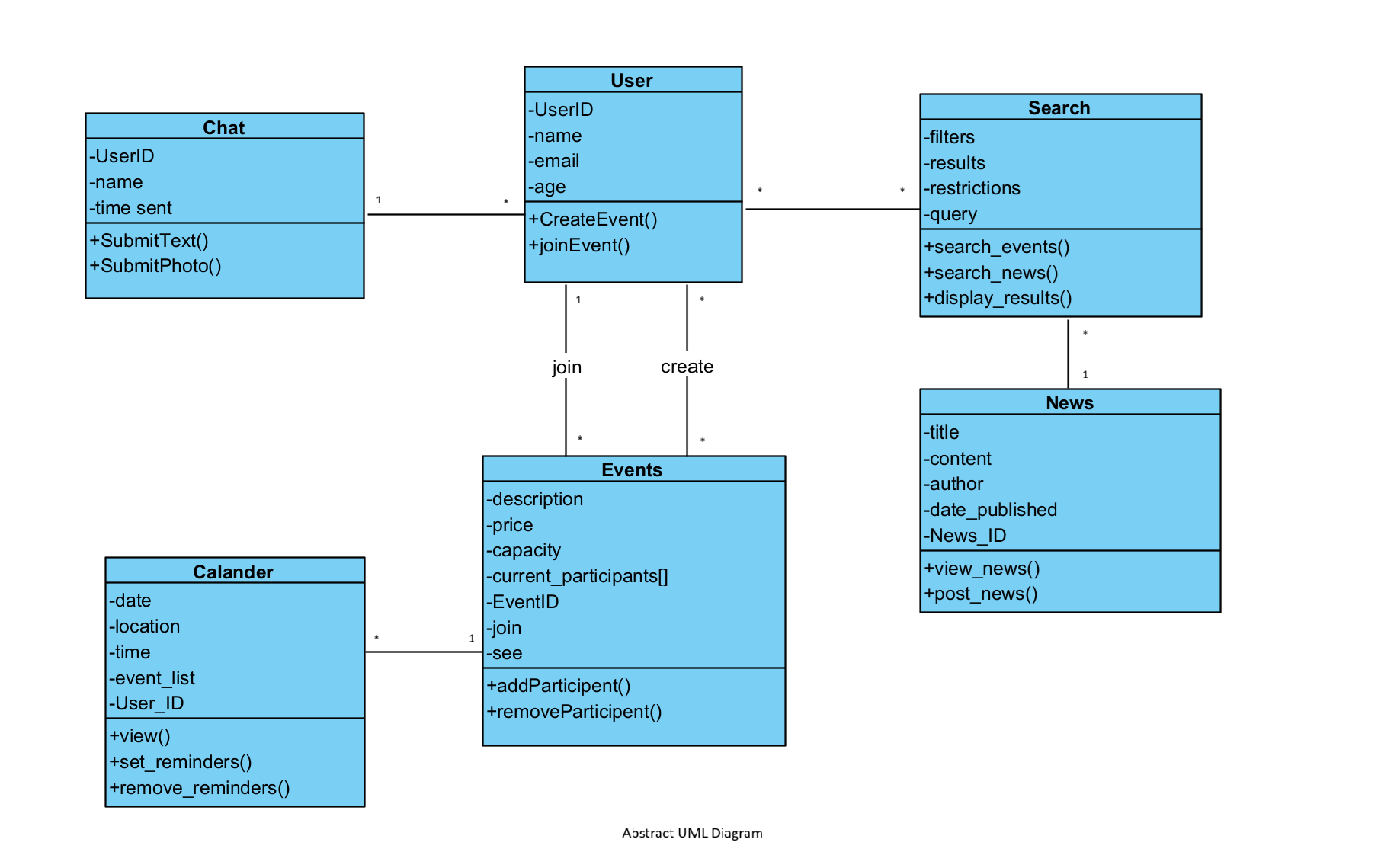
## Architecturally Significant Design Modules, Packages, and UML Diagrams



User Registration and Authentication UML Diagram



Event Handling UML Diagram



# Development Description

## Overview

The updates were built following an Agile system with scrums lasting a week. Every Monday, we met with our scrum master and discussed upcoming and completed work. The expected and actual times for the work shall be filled after completing our first sprint.

## Time Taken

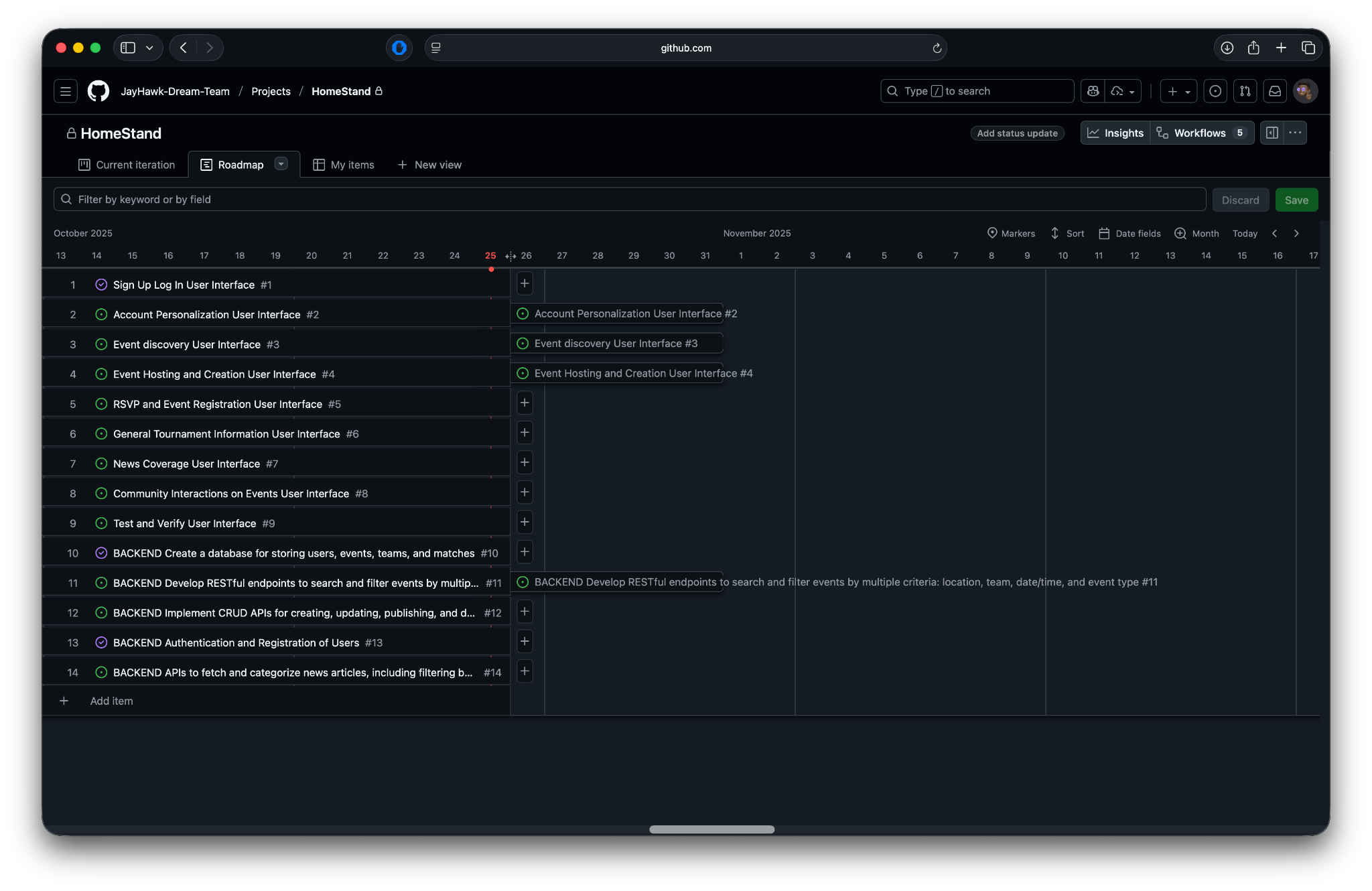
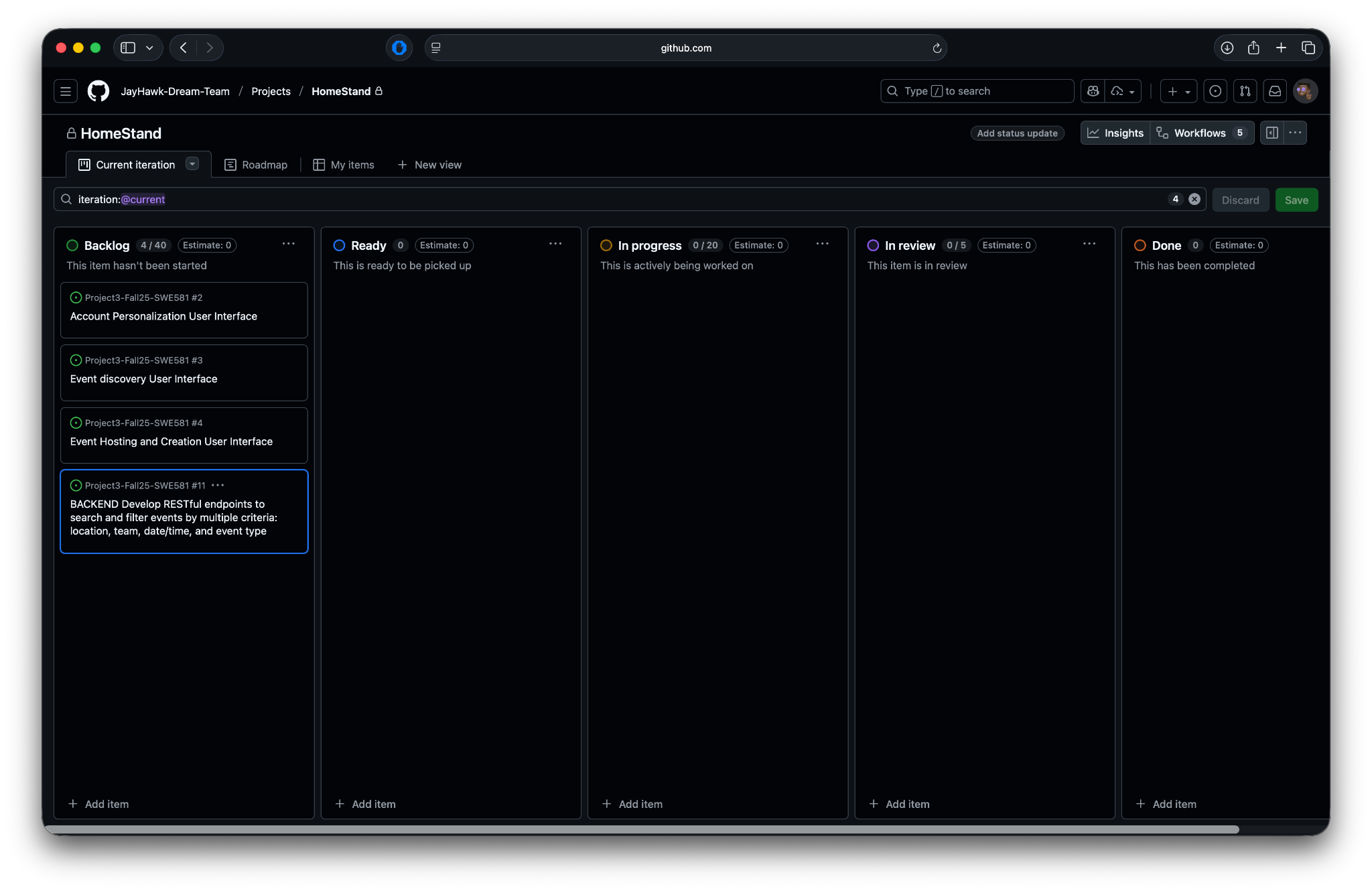
| **Name** | **Expected Time** | **Actual Time** |
| --- | --- | --- |
| Carlos Mbendera | 2 Hours | 2 Hours |
| Yoseph Ephrem | 1.5 Hours | 1.5 Hours |
| Cole DuBois | 1 Hour | 1 Hour |
| Blake Ebner | 1 Hour | 1.5 Hours |
| Jonathan Johnston | 1 Hour | 1 Hour |
| Mahdi Essawi | 2 hours | 1.3 hours |

## Time Tracking Tools

Similar to Project 1 and 2, we used GitHub extensively and GitHub Projects to keep track of our roles, tasks and timelines of each other's work.

By observing our [GitHub Project](https://github.com/orgs/JayHawk-Dream-Team/projects/4/views/1), you shall notice that we had a planning board to separate tasks into 3 sections, Todo, In Progress, Done. This made it possible for everyone to be synchronized on the status of components they may be dependent on.

Additionally, we also had used the Roadmap feature to get a rough timeline of when everything was needed on an iteration by iteration basis. In this case, each iteration was a week.



## Meeting Notes

For this project, as of October 25, 2025, we had 3 meetings. One for figuring out how to make the requirements and user reference story spreadsheets. Another one during our weekly meeting with our TA during fall break and most recently, agreeing how to write this Software Documentation and who does what.. Further communication occurred via a group chat in Group Me. 8 October, 2025, 20 October, 2025 and 23 October, 2025

**Meeting Notes – 8 October, 2025**

We were discussing how to distribute the work required to make the spreadsheets needed. Following the discussion this is how we agreed to work on it.

**Task Assignments:**

* Carlos – Front End Requirements for HomeStand
* Blake – Assisting Yoseph make the General Requirements
* Yoseph – Make the General Requirements
* Jonathan – Reference User Stories
* Cole – Back End Requiresments for HomeStand
* Mahdi – Reference User Stories

**Meeting Notes – 20 October, 2025**

Following the weekly meeting with our TA, we discussed how to distribute the technical work and programming. As a result we split ourselves into backend and frontend teams. Hence, we came up with the following distributions.

**Task Assignments:**

* Carlos – Front End
* Blake – Back End
* Yoseph – Back End
* Jonathan – Front End
* Cole – Back End
* Mahdi – Front End

**Meeting Notes – 23 October, 2025**

We were discussing how to distribute the work required to make the software architecture document needed. Following the discussion this is how we agreed to work on it.

**Task Assignments:**

* Carlos – Development Environment and Team Project Management sections
* Blake – UML Diagrams
* Yoseph – Introduction and Overview of what HomeStand is
* Jonathan – Process View of HomeStand
* Cole – Logical Technical View of HomeStand
* Mahdi – UML Diagrams