Project Requirement Documentation
Team 16: UTD Event Hub

Team Members:
Swarna Sre G
Harsha Gundavelli
Ebrahim Khan

PART 1

1. Customer Problem Statement

Problem Statement:

Students and faculty at the University of Texas at Dallas (UTD) often struggle to stay informed about campus events, mainly due to event details being scattered across multiple online platforms such as Instagram and UTD's official websites. Even on a single platform, it is unlikely that you would find information about specific kinds of events on a single page. They are usually published according to the different organizations and/or offices that are hosting them. The pages or the current online resources to find events do not reflect or work according to the perspective of the user who is actively looking to be a part of certain kinds of events on campus.

To be more specific about the different kinds of resources students and faculty use to find events, it is usually the Comet Calendar (UTD's official events calendar) or Instagram for the students, and Facebook for the faculty. The issue the users face with Instagram and Facebook is that the information is decentralized according to the respective pages of the hosting student organization or the campus office. The UTD Comet Calendar comes close to bringing different events on campus under the same platform, but does not reflect the events conducted by all student organizations.

This lack of a centralized event management system leads to missed opportunities, reduced student engagement, and inefficiencies in event discovery even when the student might have the interest to be a part of events and is actively seeking to get involved.

The UTD Event Hub project aims to solve this problem by developing a web and mobile application that aggregates event data from these diverse sources into a single, user-friendly interface. The platform will automate event collection through Facebook and Instagram scraping/APIs while offering features such as filtering, bookmarking, event categorization, and personalized recommendations.

Organizations and faculty members will also benefit from this platform, as it provides better outreach and analytics on event participation. By consolidating event details into a structured and accessible format, UTD Event Hub will enhance event visibility and participation across the campus community.

Decomposition into Sub-problem:

1. Data collection:
   - ○ Scraping event data from social media platforms and UTD websites
   - ○ Integrating APIs (Meta API, Google Calendar API) for event collection
   - ○ Handling duplicate event detection
2. Event Categorization & Filtering:
   - ○ Assigning event tags such as “Free Food,” “Tabling,” “Activities,” and “Major-Specific”
   - ○ Providing advanced filters for users to sort events by category, date, and relevance
3. User Interaction & Engagement:
   - ○ Implementing an interactive dashboard for event browsing and searching
   - ○ Enabling event bookmarking and reminder notifications
   - ○ Offering personalized event recommendations based on user preferences
4. Technical & Hosting Infrastructure:
   - ○ Backend development using Python (Flask/FastAPI) or Node.js
   - ○ Frontend development using React.js/Next.js for web and Flutter/React Native for mobile
   - ○ Data storage using PostgreSQL or Firebase
   - ○ Hosting on AWS/Firebase/Vercel

Glossary of Terms

- **Event Collection:** The process of collecting and compiling event data from multiple sources like offices and organizations pages from Instagram, Facebook and UTD official websites.

- **Meta API:** A set of tools provided by Meta (Facebook & Instagram) that allow external applications to access event data.
- **RSVP:** A feature that allows users to indicate their attendance at an event. It also provides them with the necessary steps required to register for an event (eg. filling out forms to confirm attendance, etc).
- **Personalized Recommendations:** System-generated event suggestions based on a user's past activity or preferences. This could be based on interested organizations or event types (e.g. food-served, crafty event, etc.). This includes the options to recommend content based on filters that the user can use.

2. Goals, Requirements, and Analysis

Business Goals

- Provide a **one-stop** platform for all UTD campus events.
- Automate event data collection from social media and university sources.
- Improve event discovery through filtering and categorization.
- Increase student engagement with bookmarking, reminders, and recommendations.
- Provide organizations with better tools for promoting events and tracking engagement (potential implementation).

Enumerated Functional Requirements

ID	Priority	Requirement
REQ-1	High	Scrape event data from Facebook and Instagram using Meta API or web scraping.
REQ-2	High	Store event details (name, date, location, description, RSVP links) in a structured database.
REQ-3	High	Display events in a searchable, filterable list and calendar view.
REQ-4	Medium	Allow users to save/ bookmark events and receive notifications.
REQ-5	Medium	Implement event categorization with tags such as "Free Food" and "Major-Specific."
REQ-6	Low	Enable event organizers to submit events manually.
REQ-7	Low	Provide an admin panel for event approval, editing, and removal.
REQ-8	Low	Suggest personalized events based on user preferences.

Enumerated Nonfunctional Requirements

ID	Priority	Requirement
REQ-9	High	The system must support at least 1,000 concurrent users.
REQ-10	High	Event updates should be reflected within 10 minutes of scraping.
REQ-11	Medium	The system must provide an uptime of 99.5%.
REQ-12	Medium	The mobile app should load events within 3 seconds.
REQ-13	Low	The UI must be accessible to users with disabilities.

User Interface Requirements

ID	Priority	Requirement
REQ-14	High	The system must provide an intuitive and responsive event browsing interface.
REQ-15	High	Users should have the ability to filter events using categories and dates.
REQ-16	High	The event details page should include RSVP links and event descriptions.
REQ-17	Low	Admin panel should have simple controls for event approval and edits.

3. Project Management

Team Roles and Responsibilities

Each team member has a specialized role in the development of the UTD Event Hub:

- Swarna Sre Ganesh Shankar – Data Scraping, APIs, Database.
- Ebrahim Khan – Frontend Development (React/Flutter UI).
- Harsha Gundavelli – Web Scraping & Meta API Integration.
- Entire Team – Project Management & Documentation.

The team follows an agile approach, with iterative development cycles and continuous feedback integration.

Development Timeline

The project is structured into weekly milestones for efficient development and testing:

- Week 1-2: Research Meta API, set up scraping prototype, finalize tech stack.
- Week 3-4: Develop a backend API for event storage and retrieval.
- Week 5-6: Implement frontend UI for event browsing and filtering.

- Week 7-8: Integrate notifications, refine user experience, and beta testing.
- Week 9: Final testing, deployment, and launch.

A Gantt Chart will be used to visualize task progress and deadlines.

Risk Management

Potential risks and mitigation strategies include:

- Meta API Restrictions: Research API permissions and, if necessary, use web scraping with anti-detection measures.
- Data Accuracy: Implement duplicate event detection and metadata verification.
- User Engagement: Add event reminders, push notifications, and personalized recommendations to encourage participation.

Collaboration Tools

The team uses the following tools for efficient communication and version control:

- GitHub – Code versioning and collaboration.
- Discord – Team communication and coordination.
- Discord – Task management and sprint planning.

Testing Plan

The project will undergo multiple testing phases:

- Unit Testing – Ensuring individual components work as expected.
- Integration Testing – Verifying smooth interaction between backend and frontend.
- User Acceptance Testing – Gathering feedback from target users (UTD students and faculty).

Deployment Strategy

- The application will be hosted using AWS, Firebase, or Vercel, depending on scalability and real-time data update requirements.
- The backend will be deployed with FastAPI/Flask and a PostgreSQL database.
- The frontend will be built using React.js/Next.js and mobile versions with Flutter/React Native.

PART 2

3. Use Cases
    1. Stakeholders
        a. UTD Students: Primary Users who will browse events and sign up for them
        b. UTD Faculty and Organizations: Event organizers who submit events and manage them
        c. University Administration: May post official event listings and validate entries
        d. Developers and System Administrators: Will maintain and update platform and ensure data accuracy
    2. Actors and Goals

Actor	Type	Goals
UTD Student	Initiating	Find and attend relevant events
UTD Faculty/Organizations	Initiating	Promote and manage events
University Administration	Participatin g	Approve and verify events
System (Scraper/API)	Participatin g	Gather and update event data

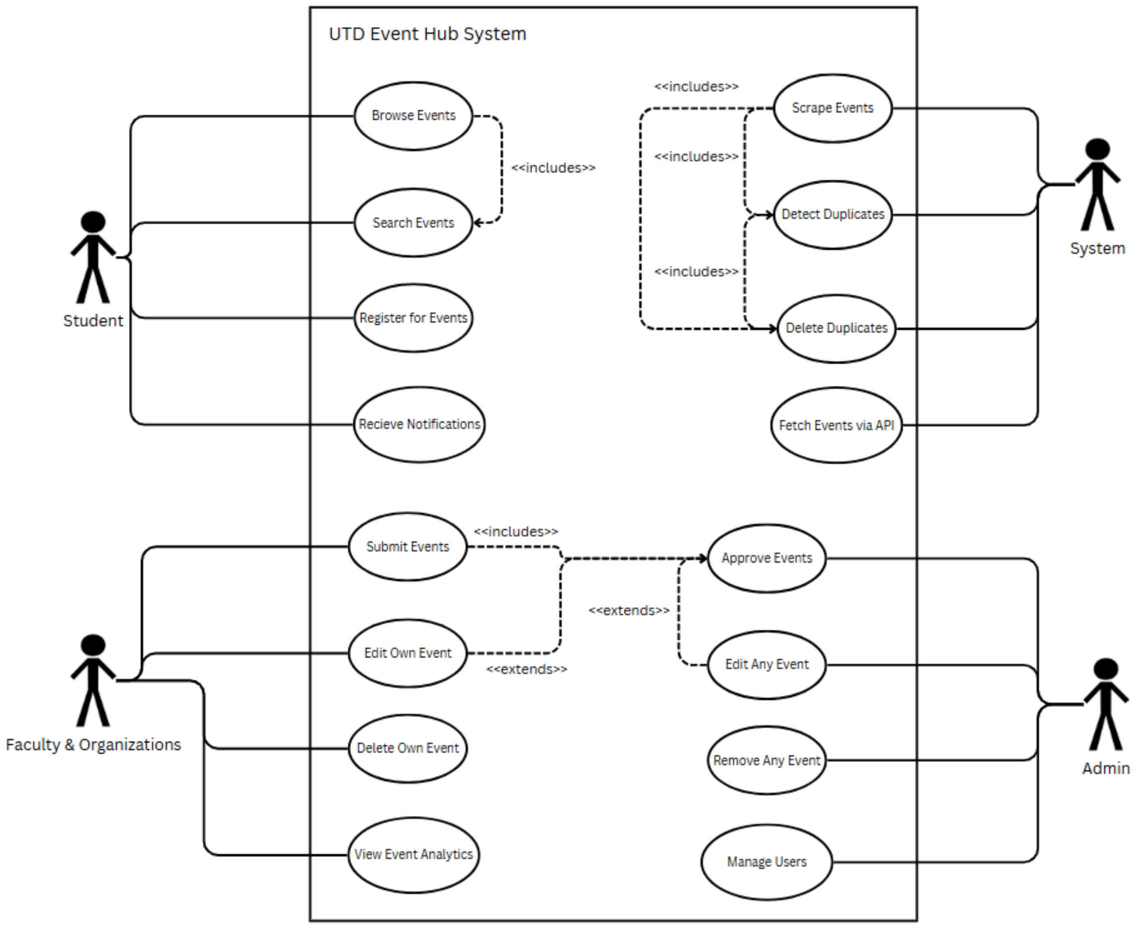
    3. Use Cases
        1. Casual Description

Use Cases	Casual Description	Responds To
Browse Events	Users can view events and filter by date and any other category	Provide a one-stop platform for UTD events
Search Events	Users enters keywords to find events	Offer a searchable event calendar
Register for	Users can register for	Allow users to

Events	events by filling out details	register for events
Receive Notifications	Users receive reminders for registered events or relevant reminders	Increase event engagement
Submit Events	Faculty and organizations submit events manually	Allow them to submit events for approval
Edit Own Event	Faculty and organizations can edit their own events	Allows them to make any changes they need
Delete Own Event	Faculty and organizations can delete their own events	Allows them to delete their events if necessary
View Event Analytics	Faculty and organizations can view analytics about events like participations	Helps them keep track of their events
Scrape Events	System will get event data from various sources	Allows to keep track of various events on campus
Detect Duplicates	System will be able to detect duplicate events	Helps in finding redundancy
Delete Duplicates	System will automatically delete duplicate events	Helps in reducing redundancy
Fetch Events via API	System will use API detection to gather event data	Help to find more events on campus
Approve Events	Admin will be able to approve appropriate events	Help to see if events are appropriate
Edit Any Event	Admin will be able to	Helps to make more

	edit events to make them more appropriate	events available for students to choose
Manage Users	Admin will manage user profile if necessary	Helps to keep inappropriate things from happening

2. Use Case Diagram



3. Traceability Matrix.

System Requirement	Use Cases	Priority Weight
R1: Users must be able to browse events.	Browse Events	4
R2: Users must be able to search for events.	Search Events	5
R3: Users must be able to register for events.	Register for Events	4
R4: Faculty and organizations must be able to submit events.	Submit Event	3
R5: Faculty and organizations must be able to approve events.	Approve Event	5
R6: Faculty and organizations must be able to edit events.	Edit Own Event, Edit Any Event	4
R7: Admin must approve events.	Approve Events	5
R8: Admin must manage users.	Manage Users	5
R9: System must detect and remove duplicate events.	Scrape Events, Detect Duplicates, Delete Duplicates	3

R10: Admin must be able to edit or remove events.	Edit Events, Remove Events	4

4. Fully-Dressed Description

Use Case: Search Events
Primary Actor: UTD Students (Initiating Actor)
Goal: To allow a student to search for events based through filtering using criteria like event name, date, or type
Preconditions: The user is logged into the system. There are existing events in the system.
Postconditions: The user is shown a list of events that match their search criteria.
Main Flow: 1. Student navigates to the "Browse Events" page 2. System displays a list of available events with a search bar 3. Student enters criteria, or uses filtering 4. System processes the search request 5. System filters events based on student's search query 6. System displays the list of events that match the search criteria 7. Student reviews the event list 8. Student selects an event to learn more
Alternative Flows: **A1:** If no events match the search criteria, system will display the message: "No events found for your search." **A2:** If student doesn't enter any search criteria, system will display all available events by default
Exception Flows: **E1:** If the system encounters an error while processing the search, the system will display an error message: "An error occurred. Please try again later."

Use Case: Register Events
Primary Actor: UTD Students (Initiating Actor)
Goal: To allow a student to register for an event they are interested in attending
Preconditions: The user is logged into the system. The user clicks on one of available events or one that is searched for The event has an RSVP link for registration.
Postconditions: The student is successfully registered for the selected event. The event registration count is updated.
Main Flow: 1. The student browses or searches for an event they wish to attend. 2. The student selects an event from the list of available events. 3. The system displays event details (e.g., date, time, location, description). 4. The student clicks the “Register” button to register for the event. 5. The system checks if there are available slots for the event. - **If available**, proceed to Step 6. - **If there are no available slots**, go to Exception Flow E1. 6. The system registers the student for the event and updates the registration count. 7. The student is notified that they have been successfully registered for the event.
Exception Flows: **E1:** If there are no available slots, the system displays a message: "Registration failed. Event is full."

Use Case: Approve Events
Primary Actor: Faculty & Organizations (Initiating Actor)
Goal: For faculty or organizations to approve an event they have submitted.
Preconditions:

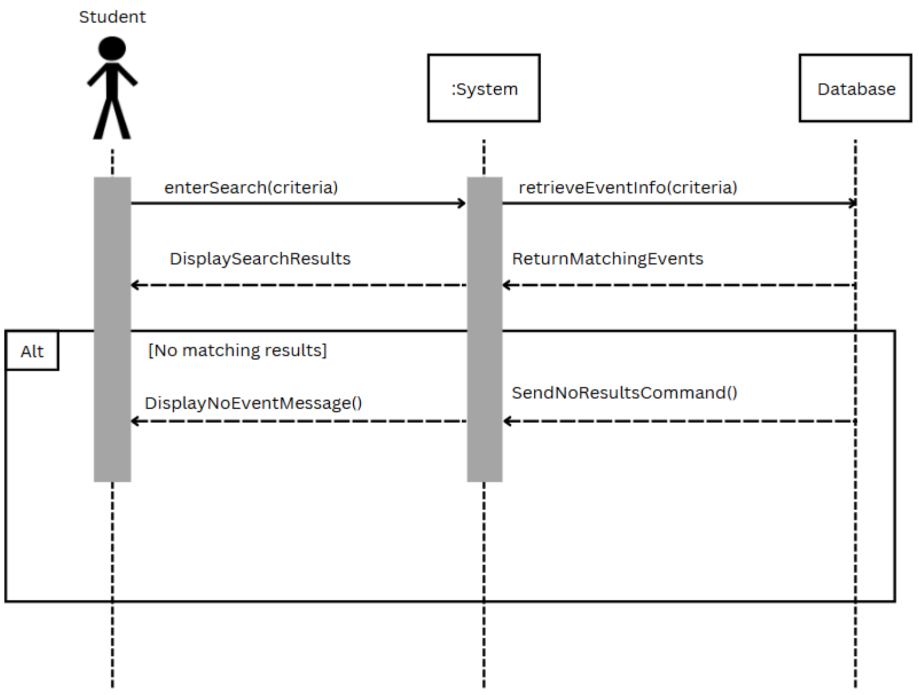
The event has been submitted by a faculty or organization. The event is pending approval. The faculty or organization has sufficient privileges to approve the event.
Postconditions: The event is approved and visible to students for registration. The event status is updated to “approved.”
Main Flow: 1. The faculty or organization logs into the system and navigates to the "Pending Events" section. 2. The system displays a list of events that require approval. 3. The faculty or organization reviews the event details (e.g., date, location, description). 4. The faculty or organization clicks the "Approve" button for the event. 5. The system updates the event status to “Approved.” 6. The event is now visible to students in the “Browse Events” section.
Alternative Flows: **A1:** If the faculty or organization decides not to approve the event, they can click the "Reject" button to decline the event.

Use Case: Scrape Events
Primary Actor: System (Participating Actor)
Goal: To automatically scrape event data from external sources or APIs to keep the event list up to date.
Preconditions: The system is connected to the external data source (e.g., API). The system has permission to access the external event data.
Postconditions: The system retrieves and processes event data from external sources. The system identifies and handles duplicate events (if any).
Main Flow: 1. The system initiates the scraping process. 2. The system fetches event data from the external source (e.g., via an API).

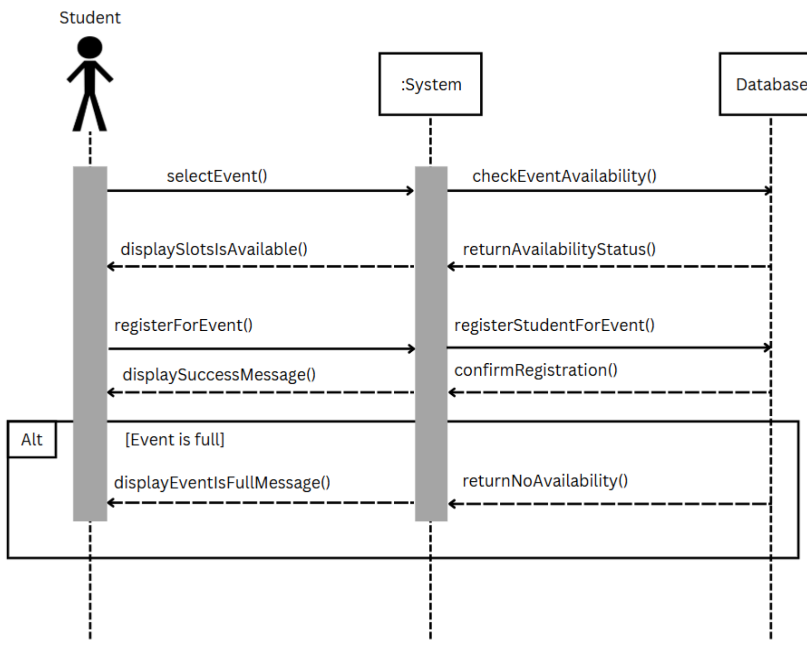
3. The system processes the fetched data, storing the events in the database.
4. The system checks for duplicate events (using the **Detect Duplicates** use case).
5. If duplicates are found, the system deletes the duplicates (using the **Delete Duplicates** use case).
6. The system stores the unique events and updates the event list.

4. System Sequence Diagrams

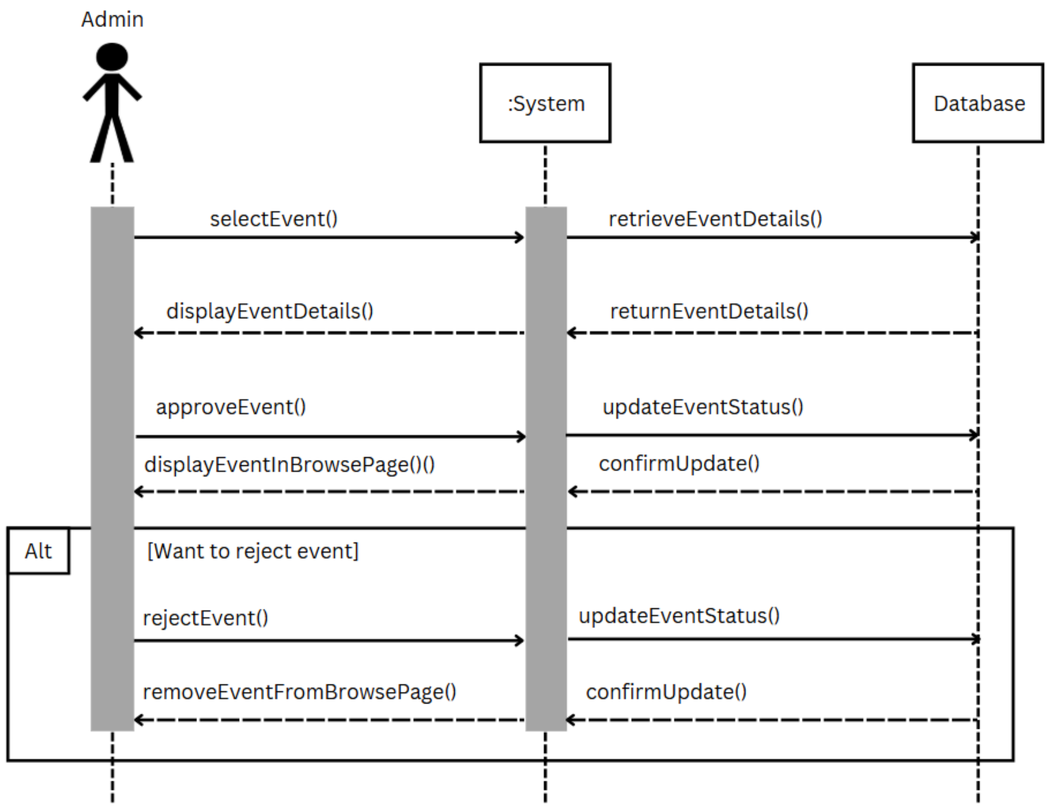
SSD: Search Events



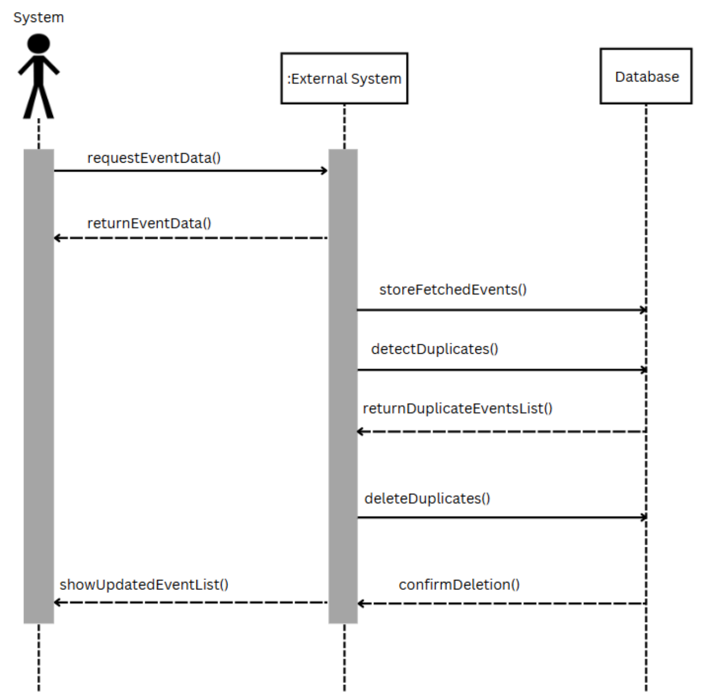
SSD: Register for Events



SSD: Approve Event



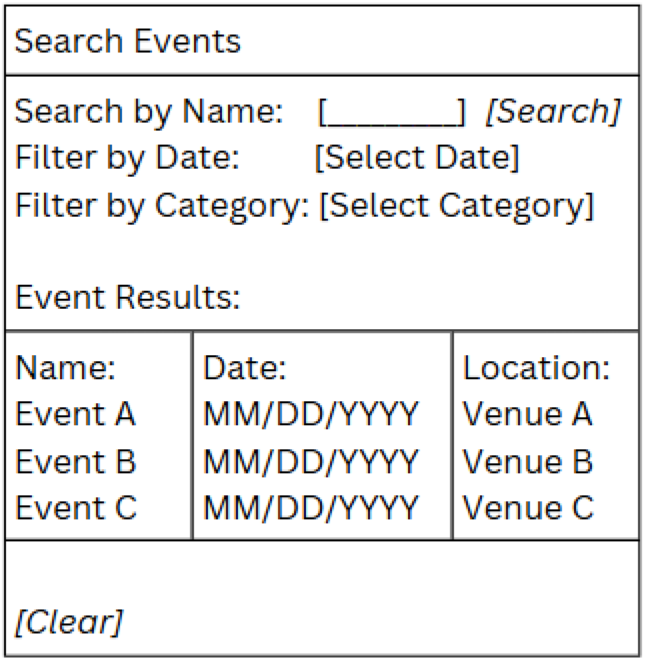
SSD: Scrape Events



4. User Interface Specification
   1. Preliminary Design

Use Case: Search Events
User Interface Elements: ● Page Title: Search Events ● Fields (User Inputs): - Search Bar (Text Input): To enter keywords (e.g., event name, location) - Event Type (Dropdown Menu): To filter by event type (e.g., Conference, Workshop) - Date Range (Dropdown Menu): To filter events by date (e.g., Today, This Week, Custom) ● Buttons & Actions: - Search (Button): Initiates the search based on input criteria. - Clear (Button): Clears all filters and search criteria.
Navigation Path: 1. User opens the homepage → navigates to "Search Events" 2. User enters search criteria in the search bar and selects filter options.

3. User clicks "Search."
4. System processes the query and displays search results.
5. User clicks on an event to view details.



Search Events

Search by Name: [________] *[Search]*
Filter by Date: [Select Date]
Filter by Category: [Select Category]

Event Results:

Name:	Date:	Location:
Event A	MM/DD/YYYY	Venue A
Event B	MM/DD/YYYY	Venue B
Event C	MM/DD/YYYY	Venue C

[Clear]

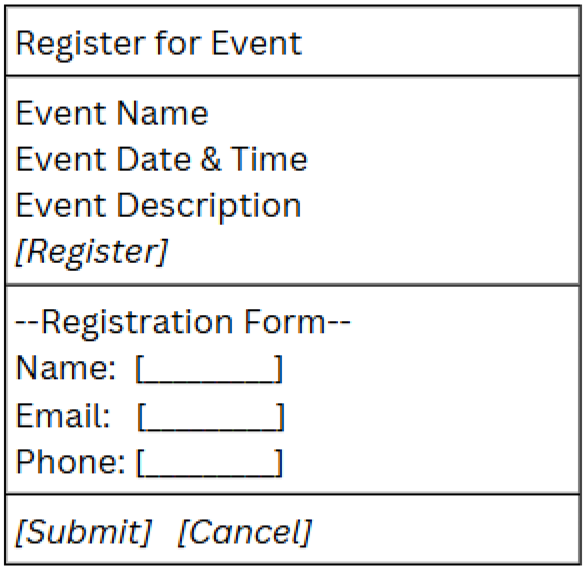
Use Case: Register for Events

User Interface Elements:

- Page Title: Register for Event
- Fields (User Inputs):
    - Name (Text Input): To enter the user's full name.
    - Email (Text Input): To enter a user's email address.
    - Phone Number (Text Input): To enter a user's phone number.
- Buttons & Actions:
    - Submit Registration (Button): Submits the registration form and payment details.
    - Cancel (Button): Cancels the registration process and navigates back to the event page or homepage.

Navigation Path:

1. User navigates to the event details page → clicks the "Submit" button.
2. User fills in their details (name, email, payment).
3. User clicks "Submit Registration."
4. System validates the inputs and payment.
5. If successful, the system displays a confirmation message: "Your registration has been confirmed!"
6. User is redirected to the "My Events" page or confirmation page.



Register for Event

Event Name
Event Date & Time
Event Description
[Register]

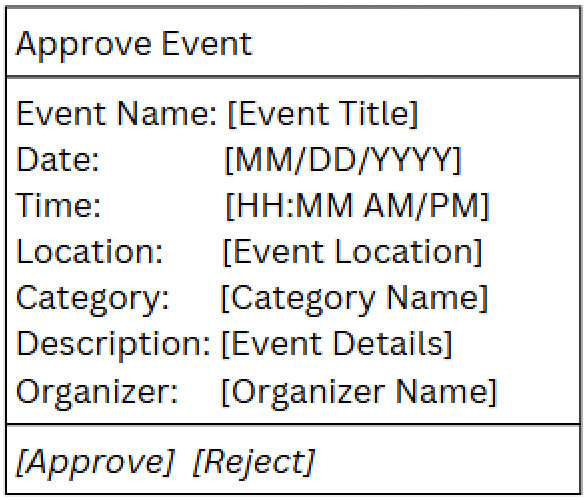
--Registration Form--
Name: [________]
Email: [________]
Phone: [________]

[Submit] *[Cancel]*

Use Case: Approve Event

User Interface Elements:
- Page Title: Approve Event
- Fields (User Inputs):
    - Event List (Table/List View): Displays unapproved events with details such as:
        - Event Name
        - Date and Time
        - Description
        - Organizer Name
- Buttons & Actions:
    - Approve (Button): Approves the event, updating its status to "Approved."
    - Reject (Button): Rejects the event, updating its status to "Rejected."

Navigation Path:

1. Admin logs into the system → lands on the Admin Dashboard. 2. Admin navigates to the "Event Approval" section. 3. Admin reviews the list of unapproved events. 4. Admin clicks "Approve" next to an event. 5. System updates event status and displays a success message: "Event Approved!" 6. Admin is redirected to the "Event Management" page or continues approving other events.


Approve Event

Event Name: [Event Title]
Date: [MM/DD/YYYY]
Time: [HH:MM AM/PM]
Location: [Event Location]
Category: [Category Name]
Description: [Event Details]
Organizer: [Organizer Name]

[Approve] [Reject]

2. User Effort Estimation

Use Case: Search Events
Details: User searched for an event based on criteria and filtering
Flow of Events: 1. Navigate to Search Events page **(1 mouse click)** 2. Enter text in search bar for event keywords **(10 keystrokes)** 3. Select Event Type from dropdown menu **(1 mouse click)** 4. Select Date Range from dropdown menu **(1 mouse click)** 5. Click Search **(1 mouse click)** 6. Review search results **(No additional clicks or keystrokes needed)** 7. Click on an event to view details **(1 mouse click)**
Total Effort: ● Mouse clicks: 5 ● Keystrokes: 10 Effort Breakdown:

- **Navigation (Mouse Clicks)**: 5 clicks = **33%** (5 clicks / 15 total actions)
- **Clerical Data Entry (Keystrokes)**: 10 keystrokes = **67%** (10 keystrokes / 15 total actions)

Use Case: Register for Events

Details: User registers for an event and fills in necessary details

Flow of Events:

1. Navigate to event page **(1 mouse click)**
2. Click Register button **(1 mouse click)**
3. Enter Name **(10 keystrokes)**
4. Enter Email **(10 keystrokes)**
5. Enter Phone Number **(10 keystrokes)**
6. Click Submit Registration **(1 mouse click)**
7. Review confirmation message **(No additional clicks or keystrokes needed)**

Total Effort:

- Mouse clicks: 3
- Keystrokes: 30

Effort Breakdown:

- **Navigation (Mouse Clicks)**: 3 clicks = **9%** (3 clicks / 33 total actions)
- **Clerical Data Entry (Keystrokes)**: 30 keystrokes = **91%** (30 keystrokes / 33 total actions)

Use Case: Approve Events

Details: Admin approves an event from the list of unapproved events

Flow of Events:

1. Navigate to Admin Dashboard **(1 mouse click)**
2. Navigate to Event Approval page **(1 mouse click)**

3. Review list of unapproved events **(No additional clicks or keystrokes needed)**
4. Click Approve next to event **(1 mouse click)**
5. Click Edit next to the event to modify details **(1 mouse click)**
6. Edit Event Details (e.g., Name, Date, Location, Description)
    a. Event Name **(10 keystrokes)**
    b. Date **(2 keystrokes)**
    c. Description **(15 keystrokes)**
7. Click Save Changes to save the edited event details **(1 mouse click)**
8. Click Approve after editing to approve the event **(1 mouse click)**
9. Review success message **(No additional clicks or keystrokes needed)**

Total Effort:

- Mouse clicks: 6
- Keystrokes: 27

Effort Breakdown:

- **Navigation (Mouse Clicks)**: 6 clicks = **18%** (6 clicks / 33 total actions)
- **Clerical Data Entry (Keystrokes)**: 27 keystrokes = **82%** (27 keystrokes / 33 total actions)

Use Case	Mouse Clicks	Keystrokes	% Navigation	% Clerical Data Entry
Search Events	5	10	33%	67%
Register for Events	3	30	9%	91%
Approve Events	6	27	18%	82%

Conclusion:

- The **Register for Events** use case requires more effort due to the additional editing process.

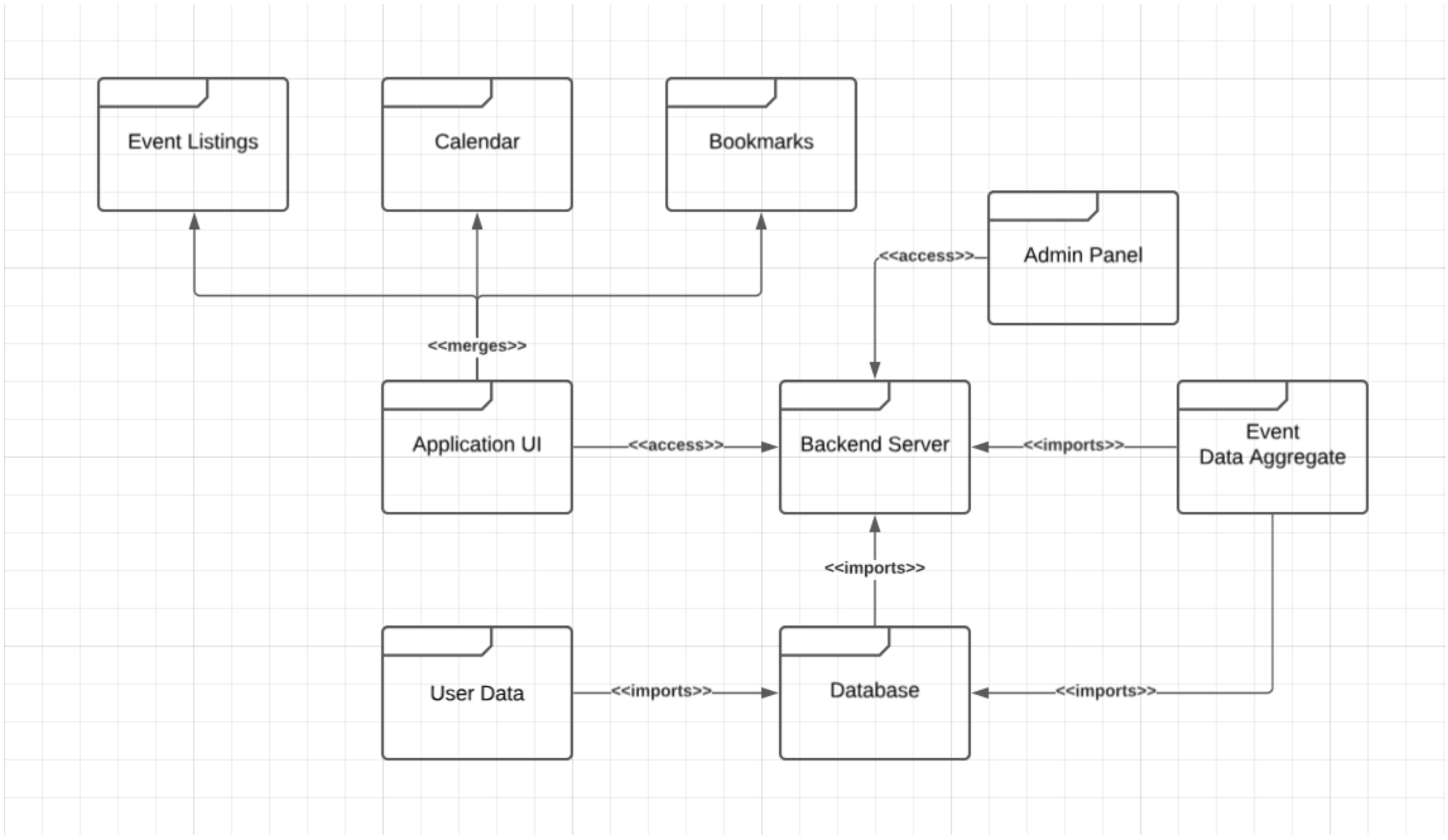
- The **editing of event details** adds to the clerical data entry (keystrokes), which increases the overall data entry effort in this use case.

PART 3

1. System Architecture

Subsystems

- **Frontend** - the main interface through which users will interact with the application. It handles UI/UX, event listings, processing user inputs, and communicates with the backend to fetch or update data
- **Backend** - acts as the intermediary between the frontend and database, processing user requests, user data, and handling API requests
- **Database** - stores and manages data related to events, user preferences, and app settings
- **Data Aggregation:** collects event data using the Meta API, any UTD official websites, and web scraping tools if API access is limited
- **Admin Panel:** only accessible by admins and will allow for manual submissions, event edits or deletion, and moderation



- *Mapping Subsystems to Hardware*
  Since this is a mobile app with a cloud-based backend, the system will run across multiple devices and servers

- **Mobile devices (frontend):** the frontend is deployed to the users' devices and the app connects to the backend to retrieve event data
- **Cloud server (backend):** runs on a cloud service to process API requests and user activity
- **Database:** a cloud database stores event data, user preferences, and authentication details.
- **Admin Panel:** hosted on the same backend server
- **Data Aggregator:** same server as the backend

Connectors and Network Protocols

The subsystems in the application all communicate with each other over a network, so proper network protocols should be chosen for efficient data transfer

- **Frontend to Backend:** communication via HTTP/HTTPS RESTful API and JWT-based authentication for user-specific data. HTTP/HTTPS APIs are lightweight, scalable, and easy to integrate. JWT provides a scalable authentication mechanism
- **Backend to Database:** PostgreSQL protocol or a Firebase Realtime Database API. PostgreSQL offers a reliable and structured data management with query support
- **Backend to Data Aggregator:** Internal calls, typically function calls
- **Web Scrapers to External Data Sources:** HTTP/HTTPS or API calls for Meta if available

Global Control Flow

- **Execution Order**
  - The system is event driven, operating primarily on user interactions. It does not enforce a step-by-step sequence.
  - The app remains in a waiting state until a user action occurs, at which it processes the request and updates accordingly
- **Time Dependency**
  - The app does not operate under strict time constraints, however event data must be updated regularly
  - If real-time event updates are necessary, it can be done through the admin panel
  - User notifications will require timed reminders for any bookmarked events

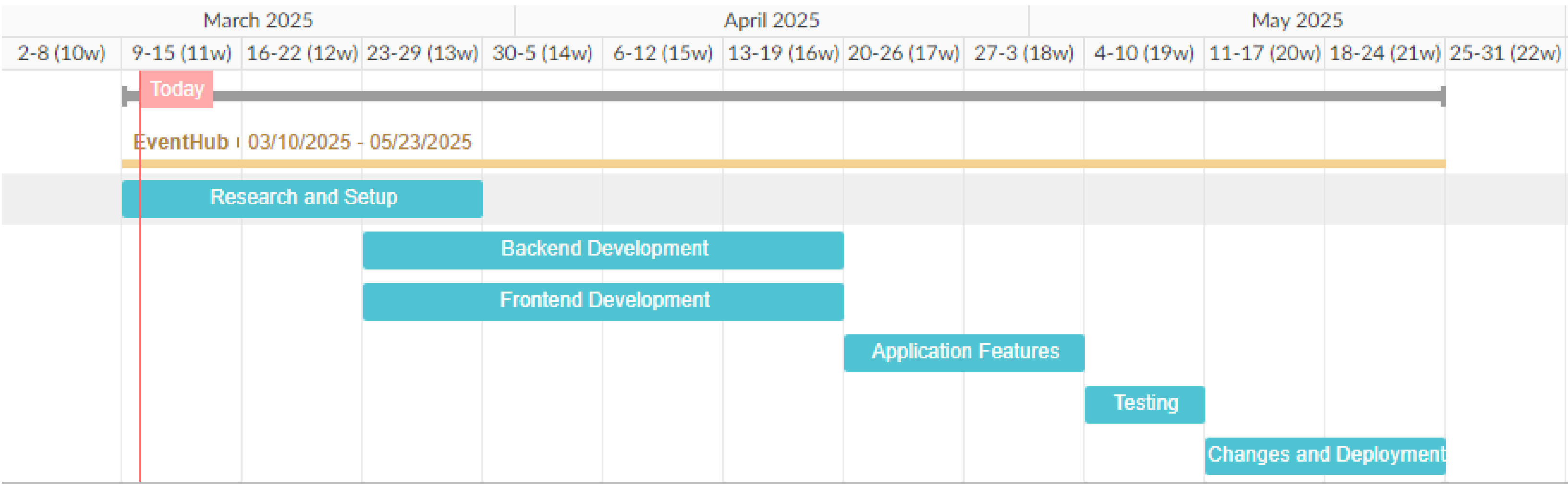
Hardware Requirements

- **Mobile Device Requirements**
    - iOS or Android smartphone with at least 2GB of RAM and recent OS releases
    - Stable internet connection through Wi-Fi, 4G or 5G
    - At least 500 MB storage available
- **Database Requirements**
    - A relational or NoSQL database with indexing for fast event retrieval
    - Sufficient storage capacity for event data and should be scalable when necessary
- **Network Requirements**
    - Stable 4G connection for users to access event listings and smooth operation of the application

2. Plan of Work

Roadmap & Milestones

Week 1-3: Research Meta API, set up scraping prototype, finalize tech stack. This phase will start with research into the Meta API, and if necessary setting up a data scraping prototype. Simultaneously, a technology stack will be finalized to ensure that the most efficient tools are selected for development.
Week 3-6: Develop backend API for event storage and retrieval. The focus will shift to the backend where API will be created for event storage and retrieval. API integrations for gathering event data will also be set up.
Week 3-6: Implement frontend UI for event browsing and filtering. These weeks will be dedicated to the frontend, where the UI will be developed for event browsing, filtering, and searching.
Week 7-8: Integrate notifications, refine user experience, and beta testing. Additional features such as push notifications and other user engagement tools will be integrated. Beta testing will also begin with the simultaneous refinement of user experience.
Week 9: Final testing, deployment, and launch. To finish, the app will go through final testing, ensuring there are no bugs and is in proper working order. Once this is complete, the app will be launched.



3. References

Canva. (n.d.). *Canva* [Online graphic design tool]. Retrieved March 9, 2025, from
https://www.canva.com
GanttPro
https://app.ganttpro.com