Final Report

UTD Events Hub

Team 16 :
Swarna Sre Ganesh Shankar, Harsha Gundavelli, Ebrahim Khan

CS 3354.004

Date: May 4, 2025

Table Of Contents

1. Individual Contributions

1.1 Frontend Development

Ebrahim Khan: UI/UX Development & Calendar Integration

- Designed and built the **main UI of the UTD Event Hub** using a modular React.js architecture with Inline CSS.
- Implemented **tag-based filtering**, as well as sorting mechanisms for time, department, and location.
- Developed a **calendar export feature**, enabling users to seamlessly add selected events to **the in-built calendar** using React Big Calendar
- Built intuitive **modal redirects** to external registration pages (when available) for seamless sign-up experiences.
-
- **Technologies Used**: React.js, Node.js, JavaScript, Inline CSS, Framer Motion, React Router, React Big Calendar

1.2 Data Collection

Swarna Sre Ganesh Shankar: Comet Calendar Web Scraping & Dynamic Input Integration

- Developed and maintained the backend scraping pipeline to extract structured event data from UTD's Comet Calendar.
- Implemented parsing logic to extract event details such as **title, date/time, image URL**, and **department/location** via multi-level scraping.
- Normalized raw event data into a clean CSV format and structured events.json to feed directly into the frontend.
- Implemented **Dynamic URL Input** capabilities to allow filtering based on user-defined query parameters like **event count**, **tags**, or **date range**, reducing manual intervention and making the scraper adaptable.
- To make real-time updates possible, used Flask and Requests to create an API that would enable that.
- Working on integrating the scraping system with Firebase/MongoDB for scalable, real-time data access.
- **Technologies Used**: Python, BeautifulSoup, Requests, JSON, CSV, Flask

Harsha Gundavelli: Instagram-Based Event Data Extraction

- Led the initiative to scrape event announcements from public **Instagram accounts** of UTD clubs and organizations.
- Utilized tools like **Instaloader**, **Tesseract, and Selenium** to gather flyer images from recent posts.
- Integrated **OCR with Gemini API** to convert text from flyer images into structured data (event titles, locations, times).
- Currently improving accuracy through text filtering algorithms.
- Implemented a duplicate detection system to check for and delete the events from Instagram that are the same as each other.
- **Technologies Used**: Python, Instaloader, Tesseract, Selenium, Gemini API, Regex

1.3. Documentation and Compilation

- **README -** Swarna Sre G
- **PowerPoint -** Swarna Sre G
- **Compilation of Source Code** - Ebrahim Khan
- **Compilation of Data Collection** - Harsha Gundavelli
- **Unit & Integration Test -** README.txt - Ebrahim Khan

1.4 Project Management

- **Communication**: The team used **Discord** for daily updates and asynchronous discussions, with in-person meetups after class for collaborative debugging and brainstorming.
- **Task Delegation**: Responsibilities were assigned based on skillsets and task complexity, ensuring that each member worked on an area that leveraged their strengths while pushing for growth.
- **Deliverable 1 Workflow**:
    - Progress was tracked weekly.
    - Regular peer reviews enabled rapid iteration and error handling.
    - Brainstorming sessions were held to align on data model changes, interface design, and future features.
- **Deliverable 2 Workflow:**
    - Progress was tracked every 5 days, due to shorter time availability.
    - Regular peer reviews and check-ins enabled rapid iteration and error handling.
    - Handling of tasks and collaboration was in a better state than ever before in the project.

1.5. RACI Responsibility Matrix

Task \ People	Ebrahim Khan	Harsha Gundavelli	Swarna Sre G
FrontEnd	Responsible	Informed	Consulted
Web Scraping	Informed	Consulted	Responsible
Instagram Scraping	Consulted	Responsible	Informed

1.6. Comments

- It is to be noted that everyone was accountable for the tasks that were happening, especially given the small nature of our team.
- Everyone contributed equally to the project, taking up tasks as and when they came by, helping each other out, and holding others responsible for the tasks.

2. Summary of Changes

2.1.1 Dynamic URL Input for Event Extraction

1. **Flexible Parameters**: Allow admins to input how many events they want to fetch and users to set filters like date range, category (e.g., cultural, professional), or club names. Currently, it allows for admins to choose the number of days of events to be put in the ’events.json’ file.
2. **Dynamic URL Builder**: Implement a utility module that constructs appropriate Comet Calendar URLs based on developer requirements (number of days). This avoids hardcoding and enables real-time generation of requests.
3. **Crawler Adaptation**: The crawler script interprets these dynamic URLs and adaptively parses structures based on the type of event or source page (handling minor variations in HTML structure).

2.1.2 Text Extraction from Posts Scraped from Instagram

1. **OCR Extraction:** Use Gemini API to extract text from flyers, including event titles, dates, times, locations, and organizers.
2. **Contextual Understanding:** Leverage the Gemini API to extract text from complex flyer layouts for certain event information like, title, date, location, and description

2.1.3. Front-End Enhancements and Interactive User Experience

1. **Updated & Improved UI:** Added a navbar with the school's mascot, with updated titles and color. Used Framer Motion to add a hover scale and shadow effect on each event card. Adjusted dropdowns to look more simplistic. Added recent, upcoming, and past events sections to separate each event by date. Additionally, updated fonts throughout the app.
2. **Saved Events:** created a Saved Events page in which users can view all upcoming and past events, grouped by date. Events can be saved by pressing a button denoted by a star located on the bottom right of each event card.
3. **Calendar Integration:** integrated react-big-calendar with a date-fns localizer to render a full month-view UI. Mapped saved events into the calendar tiles, with clickable days. Added a side bar that lists all saved events on a selected date.

2.2. Future Implementations

2.2.1. Event Registration Link Integration

Overview

The current system focuses on displaying event details but lacks direct interaction features. A crucial next step is to enhance user utility by integrating **registration links** directly within each event. This will allow users to seamlessly register for events before adding them to their calendars, increasing engagement and platform stickiness.

Proposed Enhancements

- **Scraper Update**: Modify the scraping logic to identify and extract registration links (if present) from event pages (e.g., Comet Calendar, Presence.io, or org websites).
- **Metadata Field**: Add a new field registration_link in the events.json structure and the database schema.

- **Frontend Integration**: Update the frontend UI to display a prominent "Register" button alongside event listings if a valid link is available.

2.2.2. Full Database Integration with Firebase or MongoDB

Overview

Right now, the project relies on a local events.json file which is overwritten on each scrape. Moving to a robust **database system** allows long-term storage, efficient querying, and complex logic like duplicate prevention and filtering.

Proposed Enhancements

- **Firebase**: Use Firestore for real-time updates and easy web integration.
- **MongoDB**: Use a schema-less design for flexibility and scalability.
- **Backend Logic**: Update refresh.py to insert scraped events into the database.

2.2.3. Facebook Event Integration

Overview

A large number of UTD events are only posted on Facebook. Integrating those into the Event Hub will dramatically increase coverage and make the hub more representative of real campus activity.

Proposed Enhancements

- **Facebook Graph API**: Use authenticated requests to fetch events from specific pages (e.g., UTD Student Government, clubs).
- **Scraper Module**: Create a dedicated facebook_scraper.py that handles data parsing from Facebook.
- **Rate Limiting & Privacy Handling**: Use tokens responsibly and handle visibility settings (some events may not be public).

2.2.4. Admin Panel for Event Moderation

Overview

To ensure quality control, consistency, and legitimacy of event listings, the UTD Event Hub will benefit from a secure admin panel that allows verified users (such as student org officers or faculty members) to manually submit, approve, edit, or remove events. While currently low priority, this feature becomes essential as the platform scales and incorporates data from diverse, user-submitted, or semi-automated sources.

Proposed Enhancements

- **Admin Dashboard**: A secure login-based dashboard where authorized personnel can view a list of pending, approved, and flagged events.
- **Submission Portal**: Allow manual event uploads via form fields for title, description, time, location, registration link, and tags.
- **Moderation Tools**: Approve or reject user-submitted events; edit scraped event details to fix formatting errors or add missing fields.
- **Audit Logs**: Record all admin actions (edit, delete, approve) for accountability.

2.2.5. Swipe-Based Event Registration for Mobile Users Based on Preference and Recommendation

Overview

To make the mobile experience more engaging and intuitive, a swipe-based event interface can be implemented, similar to popular social apps. This feature would allow users to quickly express interest or skip events by swiping left or right based on event previews and personalized recommendations. The system will learn user preferences over time to suggest more relevant events, improving engagement and ease of use.

Proposed Enhancements

- **Swipe UI Component:** Build a card-based interface where users can swipe right to save/register for an event or swipe left to skip.
- **Preference Learning:** Track swipe behavior to refine event recommendations using collaborative filtering or tagging systems (e.g., department, location, time).
- **Instant Action Integration:** If the event has an RSVP link, allow direct registration with a tap after a swipe right.
- **Profile Preferences:** Let users set initial interests (departments, event types, times) to jumpstart recommendations.

3. Customer Statement of Requirements

Students and faculty at UTD need a unified and customizable platform to discover campus events based on their individual interests, availability, and preferences. The current ecosystem—fragmented across Comet Calendar, Instagram pages, and Facebook events—fails to present a user-centered approach for finding relevant events. Users are often forced to manually

search through multiple student organization accounts or platforms, making it difficult to stay engaged with the vibrant campus community.

The UTD Events Hub must address this need by aggregating event information from both official UTD sources and social platforms like Instagram and by offering advanced filtering options such as tags, departments, event categories, and time. This allows users to streamline their experience, find the events they care about, and avoid information overload. The platform should prioritize ease of use, real-time updates, and a clean interface to ensure that students don't miss out on opportunities just because they weren't following the right pages.

Ultimately, the customer expects a single hub where all campus events—regardless of their origin—are accessible, searchable, and tailored to their preferences. This will reduce friction in event discovery, increase participation, and improve the overall sense of community at UTD.

4. 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.
- **Comet Calendar:** The official events calendar of The University of Texas at Dallas. It lists academic, social, and cultural events organized by various departments and student groups.
- **Web Scraping:** The automated process of extracting information from websites. In this project, it's used to gather event details from platforms like Comet Calendar and Instagram.
- **Dynamic URL Input:** A system that generates scraping URLs based on filters like event type, date, or department, allowing the scraper to adapt instead of relying on hardcoded links.
- **OCR (Optical Character Recognition):** A technique used to convert text within images (like Instagram event flyers) into machine-readable text. Helps extract event info from graphics.

5. System Requirements

USER STORIES:

Outlined Task: Being able to browse through the UTD Event Hub and filter the events based on preference of date, time, location, and department/organization. Additionally, users will also be able to add the event they want to attend to the calendar that is inbuilt into the website.

User Story 1 (Student Perspective):

As an undergraduate student actively involved in campus life, I often try to find events hosted by student organizations that align with my free time between classes. However, it becomes frustrating having to follow and scroll through dozens of Instagram accounts to stay updated, and I still miss out on events I would have loved to attend. One day, I decided to try the UTD Event Hub instead. It allowed me to filter events based on the time slots I'm available and select specific student organizations I'm interested in. I could view everything in one place, easily decide which events to attend, and directly add them to the built-in calendar so I wouldn't forget. Now, I don't need to waste time switching between social media pages—I have a centralized place tailored to my schedule.

User Story 2 (Professor Perspective):

As a professor in the Erik Jonsson School of Engineering and Computer Science (ECS), I like to stay involved with departmental activities, especially during high-stress periods like finals week when student support events are common. In the past, I had to rely on word-of-mouth or scattered emails to find out when such events were happening. With the UTD Event Hub, I was able to filter events specifically hosted by ECS-affiliated organizations and departments during finals week. This made it easy for me to identify which sessions I could participate in or recommend to my students. I also appreciated being able to add selected events to the built-in calendar feature, which helped me plan my week without missing opportunities to engage with the student body.

6. Functional Requirements Specification

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, and RSVP links) in a functional structure.
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 filtering location, department (major-specific), and time
REQ-6	Low - Future Task	Enable event organizers to submit events manually.
REQ-7	Low - Future Task	Provide an admin panel for event approval, editing, and removal.

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 - Future Task	Admin panel should have simple controls for event approval and edits.

Use Cases:

Use Case ID	Name	Description	Related Requirements
UC-1	Browse Events	User browses events by date, time, location, or department.	REQ-3, REQ-5, REQ-14, REQ-15
UC-2	Filter Events	User applies filters to view only specific types of events.	REQ-3, REQ-5, REQ-15
UC-3	Add to Calendar	User selects an event and adds it to the in-built calendar.	REQ-16

UC-4	Scrape Social Media	Automatically fetches events from Instagram/Facebook.	REQ-1
UC-5	Submit Event	Event organizers manually input event data.	REQ-6
UC-6	Admin Approval	Admin reviews and approves/rejects event submissions.	REQ-7, REQ-17
UC-7	Bookmark Event	Users can bookmark events to view later.	REQ-4

Use Case Example

UC-1 Browse Events

Primary Actor: Student
Goal in Context: Discover relevant UTD events using the event hub.
Scope: UTD Event Hub System
Level: User goal
Stakeholders and Interests:

- Student: Wants an easy way to find events relevant to them.
- UTD Admin: Wants students to participate more in campus life.

Preconditions:

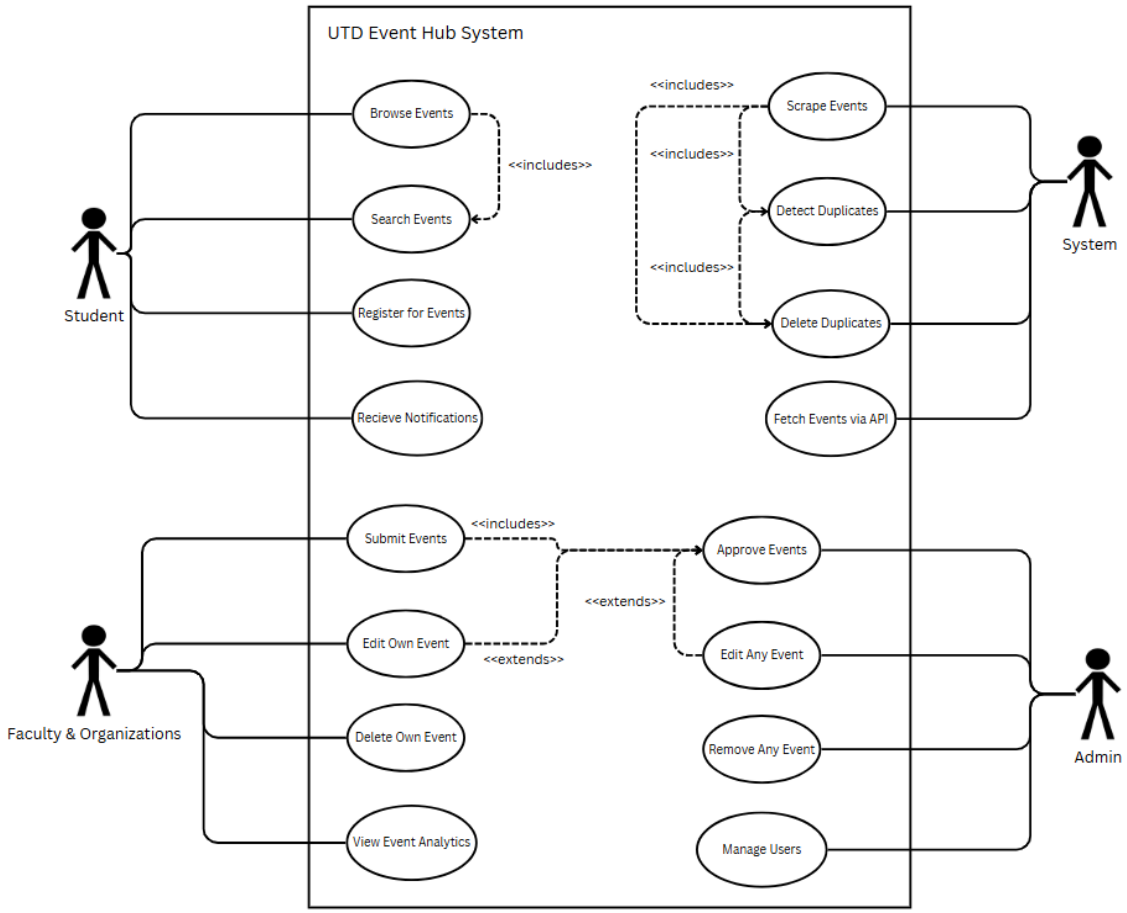
- The student is on the website or app.
- Events have already been scraped and stored.

Postconditions:

- The student browses events filtered by preference.

Main Success Scenario:

1. Student opens the UTD Event Hub.
2. System loads available events.
3. Student selects filters: date, time, department.
4. System returns matching events.
5. Student clicks an event to view more details.
6. Student optionally bookmarks or adds event to calendar.



7. Effort Estimation

Task	Estimated Hours	Actual Hours	Notes
Harsha:			
Instagram Scraper (Selenium and Instaloader)	10	14	Extra time due to anti-bot challenges
OCR + Text Extraction	6	8	Needed more time to improve text accuracy
Gemini API Integration	8	11	Needed more time to improve accuracy and classification of text
Frontend Integration	0.5	0.5	Coordination with Ebrahim

Swarna Sre:			
Scraping from Comet Calendar	10	15	Extra time spent to learn concepts and to parse html properly and to format output
Dynamic URL Implementation	2	3	This feature takes how many days events are supposed to load and outputs that many
JSON Export	1	1	Exports the information as a JSON file
Team Management	3	3	Delegate tasks; plan for MVP and check-in on updates
Webscraping - Frontend Integration	0.5	0.5	Coordination with Ebrahim
Ebrahim:			
In-build Calender	4	6	Using React Big Calendar
Basic Website & Functions	9	12	Built using React and Node.js
Updated UI	4	5	UI was improved and made more user-friendly
Integrate Instagram Data	1	1	Website made to display data procured from Instagram posts too

8. Domain Analysis

Domain Concept	**Description**
Data Gathering	Refers to collecting event-related information from online sources. In this project, it includes scraping posts from Instagram and academic calendars to obtain potential event content (images and text).
Text Extraction and Analysis	Involves converting images (e.g., Instagram posts) to text using Optical Character Recognition (OCR), then analyzing and classifying the extracted text to identify relevant event details like

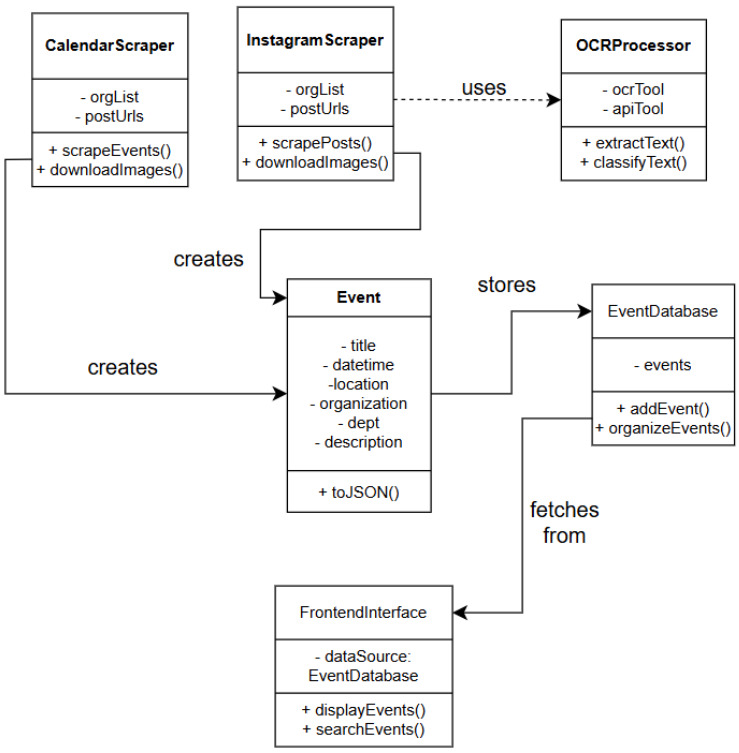
	title, date, location, and description.
Structure Event Data	The process of organizing extracted data into a well-defined format (such as the Event object) so it can be stored, retrieved, and displayed consistently.
Event Storage	Refers to the persistent storage and indexing of structured event data. It includes storing, retrieving, and querying events, enabling search and filter operations.
Frontend Interface	The user-facing part of the system that presents the stored events in a readable format. It includes functionality for displaying event lists and allowing users to search or filter by keywords.

Traceability Matrix:

1.

System Requirement	Use Cases	Domain Concepts	Priority Weight
R1: Users must be able to browse events.	Browse Events	Frontend Interface, Event Storage	4
R2: Users must be able to search for events.	Search Events	Frontend Interface, Event Storage	5
R3: Users must be able to register for events.	Register for Events	Frontend Interface, Event Storage	4
R4: System must detect and remove duplicate events.	Scrape Events, Detect Duplicates, Delete Duplicates	Data Gathering, Text Extraction and Analysis, Structure Event Data, Event Storage	3

9. Class Diagram and Interface Specification



Traceability Matrix:

Domain Concepts	Class
Data Gathering	InstragramScraper, CalendarScraper
Text Extraction and Analysis	OCRProcessor
Structure Event Data	Event
Event Storage	EventDatabase
Frontend Interface	FrontendInterface

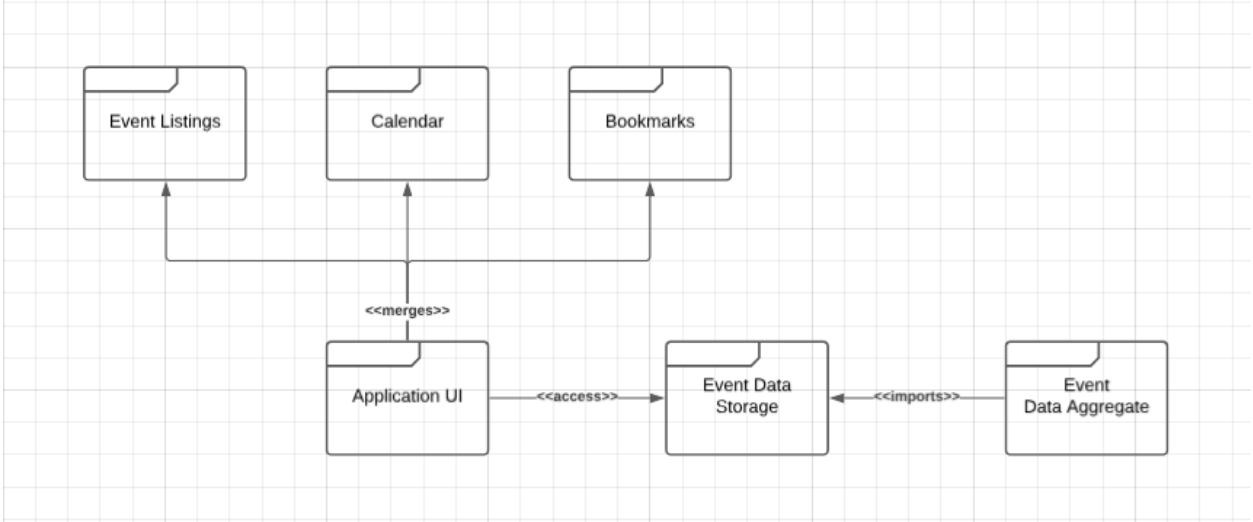
Text Description:

Concept-to-Class:	Description:
Data Gathering → InstagramScraper, CalendarScraper	InstagramScraper for social media scraping and CalendarScraper for Comet Calendar scraping. Each scraper encapsulates methods to access and collect raw data from distinct sources.
Text Extraction and Analysis → OCRProcessor	We needed a mechanism to read event data embedded in images. We encapsulated this functionality into the OCRProcessor, responsible for extracting and classifying image text using tools like Tesseract and Gemini API.
Structure Event Data → Event	The raw output from scrapers and processors needed to be structured into a consistent format. The Event class emerged to represent the finalized event object with fields like title, date, location, and description. It provides serialization methods (e.g., toJSON) for frontend integration.
Event Storage → EventDatabase	To manage and retrieve structured events, we created the EventDatabase class. It abstracts data storage, lookup, and filtering functionality for all parsed events, supporting both frontend and backend operations.
Frontend Interface → FrontendInterface	The user-facing part of the application is managed by the FrontendInterface, which communicates with the EventDatabase to display events, support search, and render outputs. It allows interaction and filtering features.

10. System Architecture and System Design

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 Gemini API, UTD Comet Calendar, Instagram, and web scraping tools if API access is limited like Tesseract and Selenium



Mapping Subsystems to Hardware

The system is a web-based application accessed by students through modern desktop or laptop browsers. It uses a cloud-based backend for background data scraping and a cloud database for event storage and retrieval.

- **Web devices (frontend):** Runs in student browsers and retrieve event data directly from the database through a secure API.
- **Server (backend):** A lightweight server periodically runs scraping and processing tasks, updating the database with new event information.
- **Database:** A cloud database stores event data.
- **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
- **Web Scrapers to External Data Sources:**
  Scrapers fetch events using HTTP/HTTPS requests or third-party APIs (e.g., Instagram Graph API if available).
- **Frontend to Backend:**
  (Optional or limited)—If any registration logic or admin features are backend-dependent, communication is via RESTful API using HTTPS and JWT for authentication.

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**
  - Event updates happen periodically via backend scraping jobs. Student-facing features don't require real-time updates but benefit from regular background refreshes. Bookmarks can be managed client-side.

Hardware Requirements

- **Device Requirements**
  - Desktop or laptop with a modern browser (e.g., Chrome, Firefox, Edge). Minimum 4GB RAM and a stable internet connection via Wi-Fi.
- **Database Requirements**
  - A scalable cloud database (relational or NoSQL) with indexing to support fast event searches and concurrent reads from frontend users.
- **Network Requirements**

- Stable broadband internet for smooth loading and registration operations. HTTPS is required for secure API communication.

11. Algorithms and Data Structures

The UTD Events Hub uses practical filtering, scraping, and text extraction algorithms to manage and present event data from multiple sources efficiently.

1. Filtering and Sorting

Implemented front-end filtering based on department, date, time, and event name using JavaScript's native .filter() and .sort() functions on event arrays. These provide responsive, client-side filtering for users to view relevant events with minimal delay.

2. Data Collection and Parsing

Scraped structured data from UTD's Comet Calendar using BeautifulSoup and Requests. Events are stored as Python dictionaries and converted to JSON and CSV formats. Filters based on query parameters (e.g., event count, department, date) allow dynamic control of data collected.

3. Instagram Data Extraction

Used Instaloader, Tesseract, and Selenium to retrieve posts' images from UTD-affiliated Instagram accounts. Applied OCR (via Gemini API) to extract text from event flyer images, then parsed it using regex and keyword matching to identify date, time, and event title.

4. Duplicate Detection (Current Approach)

Duplicate events are identified by comparing event name, date, and time fields directly. Matching entries are excluded during data finalization to avoid redundancy.

12. User Interface Design and Implementation

The user interface features a simplistic design for smooth app navigation and clarity of application content. It was implemented using React.js along with additional libraries such as Framer Motion, React Router, and React Big Calendar. At the top of the application is a green navigation bar that matches the official university colors and displays the app title and a logo of the university mascot, Temoc. To the right, users can find navigation buttons for the three main

pages: Event List, Calendar, and Saved Events. React Router handles the routing, allowing seamless transitions between these pages.

Below the navigation bar are the filter boxes, and these were implemented using React, which has pre-installed functionality for this. The events are listed in boxes called event cards. The event listings are displayed as individual event cards, each containing the department, an image, title, time, location, a link to the event, and a "Save Event" button. These cards were styled using inline CSS to adjust visual elements like size and color. The calendar page was implemented using React Big Calendar, which is a library that features a functional and interactive calendar. Finally, the saved events page, which follows the same format as the Event List page.

13. Design of Tests

The testing of UTD Event Hub was carried out in a test.js file, which contained the core logic functions of the application. The tests were designed to validate essential features such as time extraction, time tagging, keyword-based search, department filtering, and time filtering. The file uses a small set of mock event data that resembles the structure of the actual event data in the app, which ensures consistent behavior between the test and the actual application. To simulate different user inputs, editable variables are provided at the top of the file, allowing testers to modify values and observe how the tests respond. Running the file is done using the terminal, which then prints pass or fail along with the relevant output details.

14. History of Work, Current Status, and Future Work

History of Work

- The UTD Events Hub project began with the goal of consolidating event listings from multiple platforms (Comet Calendar, Instagram) into a single, filterable web interface.
- Initial work focused on frontend development—building a responsive UI using React.js and InlineCSS to display events and enable filtering by department, time, and date.
- Parallelly, a backend scraping system was built using Python and BeautifulSoup to collect structured event data from UTD’s Comet Calendar.

- A secondary pipeline was introduced to collect event data from Instagram using Instaloader and Selenium, along with OCR (Gemini API) to parse Instagram posts' text.

Current Status

- The website is functional with:
    - Filter-based search for events (by name, time, date, and department).
    - Integrated calendar view and export functionality.
    - Scraping from Comet Calendar with CSV/JSON output.
    - A working Instagram scraper with basic duplicate detection based on exact matching of event name, time, and date.
- Modal redirection to external registration pages is enabled for smoother sign-ups.

Future Work

- Expand filtering options to include tag-based categories (e.g., social, academic, networking).
- Enable full integration with Firebase or MongoDB for real-time updates and scalable storage.
- Enhance OCR and keyword recognition accuracy for Instagram posts.
- Develop a secure, role-based admin dashboard enabling approved users to moderate events (edit/verify/remove), manage submissions via structured forms, and track actions with detailed audit logs, ensuring accountability and compliance.
- Add swipe-based event browsing and one-tap calendar registration for mobile users to boost ease of access and engagement.

15. References

Web Scraping

- https://www.youtube.com/watch?v=8dTpNajxaH0&t=23s
- https://medium.com/@joerosborne/intro-to-web-scraping-build-your-first-scraper-in-5-minutes-1c36b5c4b110
- https://pypi.org/project/beautifulsoup4/

Instagram Data Extraction

- https://scrapfly.io/blog/how-to-scrape-instagram/
- https://medium.com/analytics-vidhya/web-scraping-instagram-with-selenium-python-b8e77af32ad4

- https://www.youtube.com/watch?v=iJGvYBH9mcY
- https://www.youtube.com/watch?v=sbpHUu38BdY
- https://aistudio.google.com/welcome?utm_source=google&utm_medium=cpc&utm_campaign=FY25-global-DR-gsem-BKWS-1710442&utm_content=text-ad-none-any-DEV_c-CRE_726057516129-ADGP_Hybrid%20%7C%20BKWS%20-%20EXA%20%7C%20Txt-Gemini-Gemini%20API-KWID_43700081668042204-kwd-927524447508&utm_term=KW_gemini%20api-ST_gemini%20api&gad_source=1&gad_campaignid=20866959509&gbraid=0AAAACn9t64Gsb33Dr4SzNSAlGGRgfUOZ&gclid=Cj0KCQjwoNzABhDbARIsALfY8VNTQO8UNRPnfLisP5ZECdQuGS_ZwTB-Cmx6k1PGSNCVx1VPe45M5JIaAivTEALw_wcB&gclsrc=aw.ds

Front End

- https://youtu.be/SqcY0GlETPk?si=e_q1DQSrKZOHnAod
- https://www.youtube.com/watch?v=78MjySzYnk4
- https://www.w3schools.com/nodejs/

16. Reflective Essays

Swarna Sre Ganesh Shankar

This course met my academic and personal learning needs in many ways. It gave me a structured understanding of software engineering principles, especially about software processes and documentation. I gained a much clearer picture of how software systems are planned in real-life projects. The course material and textbook covered essential areas, although I believe that the use of simpler, metaphorical examples in class would have helped me internalize the concepts more easily. Despite that, the exposure to hands-on project work helped solidify these concepts through application.

For our team project, I was primarily responsible for developing and maintaining the backend scraping pipeline, especially for the Comet Calendar. I implemented parsing logic to extract and clean event data, structured it into usable formats, and helped with integrating dynamic requirements. Working on this helped me understand how documentation, testing, and modularity play critical roles in real-world software systems. I also saw how iterative changes to our MVP scope reflected an Agile development mindset—this was particularly insightful and directly tied into the course concepts.

The most difficult part of the course for me was organizing all the required documentation and ensuring every section clearly presented the necessary information. It required a lot of context-switching and detailed thinking. If I had another semester to work on the project, I would establish fixed weekly team syncs with shared sprint boards to improve communication, planning, and live collaboration. Technically, one recurring challenge was running each other's code across different machines—we often lacked dependencies or needed more setup time than expected. Understanding each script's requirements and usage took effort, especially when integration was involved. The course prerequisites definitely helped—especially with scripting, technical improvement accommodation, and understanding code.

Learning about software engineering techniques such as **Agile methodology**, use-case modeling, and design thinking helped me appreciate the value of iterative improvement and stakeholder-focused planning. Among these, modular design and **iterative MVP planning** were most helpful for our team. They allowed us to keep the scope manageable while making meaningful progress. On the other hand, techniques like formal specification or in-depth UML modeling felt less useful in our fast-paced, prototype-heavy project setting.

Working as part of a team presented its own challenges—coordination without real-time discussions sometimes caused redundant work or delay in understandings. However, I also experienced the benefit of shared responsibility and the creative freedom that comes from working with motivated peers. Seeing everyone adjust their roles and support one another when needed was genuinely inspiring and made the **experience enjoyable.**

In conclusion, this course helped bridge the gap between theory and practice. It enhanced both my technical and soft skills, especially in communication, documentation, and modular

thinking. The course made me more confident in participating in software development projects and better prepared to understand how real engineering teams work.

Harsha Gundavelli

This essay reflects on my experience in this Software Engineering course, including the challenges and lessons I learned through my group's design project. This course was about introducing us to the full software development lifecycle, from planning and requirements gathering to testing and implementation. As a member of developing this event application that displays events from UTD, I was responsible for scraping UTD student organizations for relevant event information that would not be found on the Comet Calendar.

The course provided a strong foundation in software engineering practices, including Agile methodology, system design, and testing strategies. I have gained practical exposure to important topics like UML diagrams and iterative development. These topics helped structure our team's workflow and make real-world development. However, the course could benefit from the addition of more backend integration topics like API usage, web scraping, authentication, and data ethics. As these topics are important for modern software projects. The most difficult aspect of the course was applying theoretical knowledge into practice. Applying concepts like modular design to our real project required trial and error. For our team project, the biggest challenge was integrating team members' work together. Different programming styles and testing approaches made coordination harder than expected. Maintaining consistent goals and communication took time to improve.

In our project, my role was to scrape Instagram pages of various UTD organizations to extract event information for our application. This task was crucial to make our application useful and up-to-date. I used tools like Selenium to automate browser interactions and Tesseract OCR with Gemini API to extract and classify text from images. A major challenge was avoiding violations of privacy and anti-bot policies. I initially tried using the Instagram Graph API but ran into access issues, so I switched to browser automation with Selenium. Post image formats were also inconsistent, requiring me to design logic to identify and extract key event details like title, date, and location. For this I used Gemini API to extract text from images and classify the text for relevant fields (like title, date, and location), and standardize the output for frontend use. If I had another semester, I'd prioritize earlier integration testing and weekly internal demos. That would help detect broken features or misalignments early on. I would also encourage more structured task boards and use of GitHub to consistently share code. With more time, I'd build a more robust event-scraping pipeline and integrate a better AI-based image recognition to detect flyers and structure data for the frontend.

Some of the major technical challenges I face included bypassing anti-bot measures on Instagram, parsing unstructured text and images with inconsistent formatting, and handling rate limits and dynamic content loading. Tools like Selenium, Tesseract, and Gemini API were

essential in managing these. I learned a lot about adapting to technical obstacles and iterating on solutions quickly. The most useful techniques were version control and agile stand-ups. They helped keep the team on task and helped resolve any blockers early on. The least useful technique was formal UML diagramming. As it seemed too rigid for our evolving scope. Lightweight planning tools like flowcharts or whiteboard sessions would have been more practical and adaptive.

Working on a team had many benefits. I learned how to collaborate across disciplines, navigate frontend/backend integration, and adapt to others' feedback. Our time together has taught me the value of communication, shared goals, and flexibility. Prerequisite courses covering web scraping, basic APIs, and cloud platforms would make future students better prepared. Overall, this course has helped make the connections from theory to application and gave me a solid start towards building scalable, real-world software solutions.

Ebrahim Khan

Throughout this course, I learned important software engineering principles, more specifically about software processes and the different types of software engineering, which include component-based, service-oriented, and systems engineering, and more. Overall, I gained a much clearer understanding of the software engineering process and how real-life projects are carried out. I believe the most difficult aspect of this course was the team projects. In general, learning about software engineering was fairly straightforward, but working in a team was more difficult, as you have to take into account your teammates in your work and learn to distribute and communicate correctly to make progress.

For the group project, my role was to create the user interface and experience. I chose to create this using JavaScript and React.js, as this is the most popular option within frontend development and many resources are available to use. The biggest technical challenge for me was learning JavaScript and React. I did not have significant experience with the language before the start of the project, so it took some time getting used to everything the language has to offer. I don't think there should be any prerequisite course for this, as there are already several that teach students how to program, and going into this course is an opportunity to use what they have already learned or to learn something new.

In conclusion, I think the course was sufficient in making me understand the different concepts required to do the project, from the beginning to the end.