CSC 431 – Spring 2025

MeetSync

System Architecture Specification (SAS)

Team 2

Jonathan Bergbaum	Member
Salifu Fuseini	Member
Alden Sahi	Member

Version History

Version	Date	Author(s)	Change Comments
1.1	April 11	Salifu	Created document and outline
1.2	April 14	All	Filled in outline with relevant information
1.3	April 21	All	Edited the document with critique from Professor

Table of Contents

Table of Figures

1. System Analysis

1.1 System Overview

The Group Scheduling Application is designed to facilitate seamless scheduling and collaboration among group members. The system allows users to create profiles, schedule events, vote on preferred meeting times, chat within the app, and receive timely notifications. Integration with external calendar services and mobile accessibility are key features.

The application follows a **3-tier architecture** (Presentation, Logic, and Data tiers) to maintain modularity, enhance scalability, and promote maintainability.

1. Logical View

Main Concepts & Relationships:

Entity	Description
User	Authenticated individual, creates and joins events
Event	Meeting with date, time, description, attendees
Poll	Poll attached to event for selecting times or platforms
Vote	Each user's response to a poll
Reminder	Notification scheduled for an event
Message	Chat messages tied to an event group
CalendarLink	Integration reference to external calendars (Google, etc.)

2. Development View

Frontend (React):

- Organized by components and tabs
- Communicates with Django REST API via fetch or axios

- Pages: Profile, Event, Polling, Chat, Calendar Sync

Backend (Django):

Django app structure:

```
scheduler/
├── models.py  # Event, Poll, User, Message, Reminder
├── views.py   # API views using DRF
├── signals.py
├── urls.py
└── admin.py
```

- RESTful endpoints (e.g., /api/events/, /api/polls/, /api/messages/)

Modularity:

- Django REST Framework (DRF) separates views, models, and serializers cleanly
- Frontend and backend are decoupled — clean API contracts

---

3. Process View (How it behaves at runtime — for ops/performance)

Runtime interactions:

- React frontend makes asynchronous requests to Django REST API
- Each user action (e.g., create event, vote, send message) triggers a backend API call
- Backend handles validation, storage, and response serialization
- Celery for background tasks like scheduled reminders

Concurrency & Scalability:

- Django with Gunicorn or ASGI (e.g., with Daphne) supports concurrent users
- Stateless API design allows for easy horizontal scaling

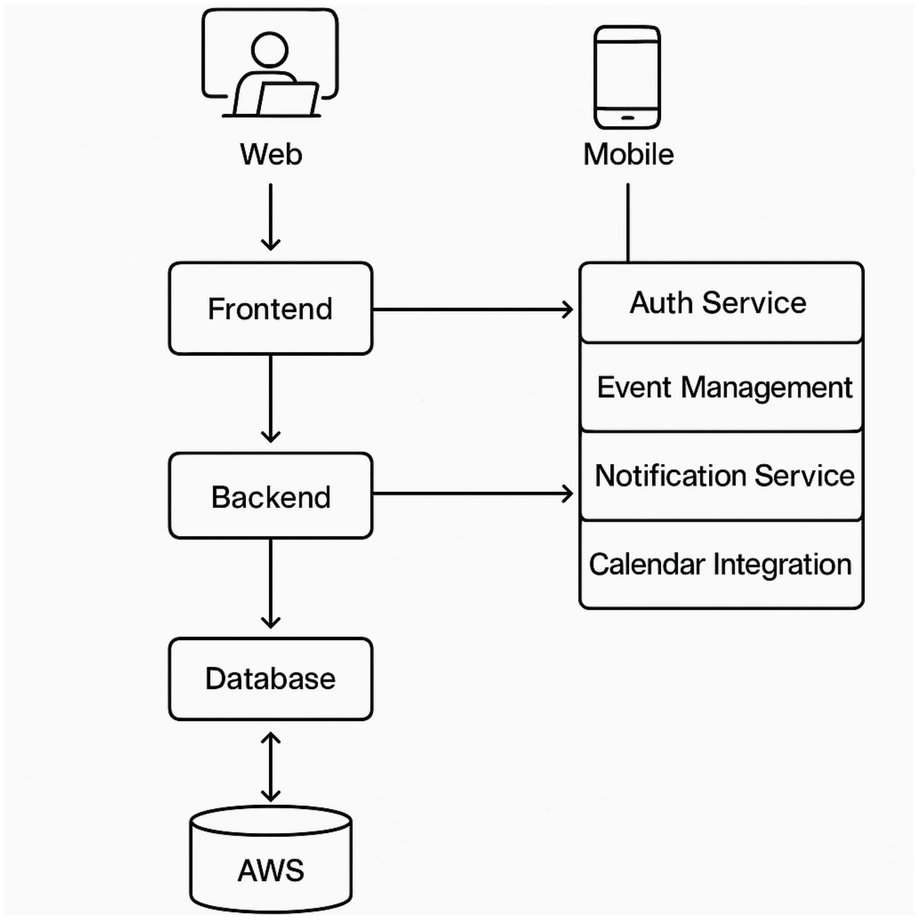
- PostgreSQL handles relational data

---

4. Physical View (How it's deployed — for infrastructure engineers)

- **Frontend**: Deployed via Vercel or Netlify
- **Backend**: Django REST deployed on Render or Railway
- **Database**: Hosted PostgreSQL
- **Reminders**: Can be powered by Celery workers and Redis

1.2 System Diagram



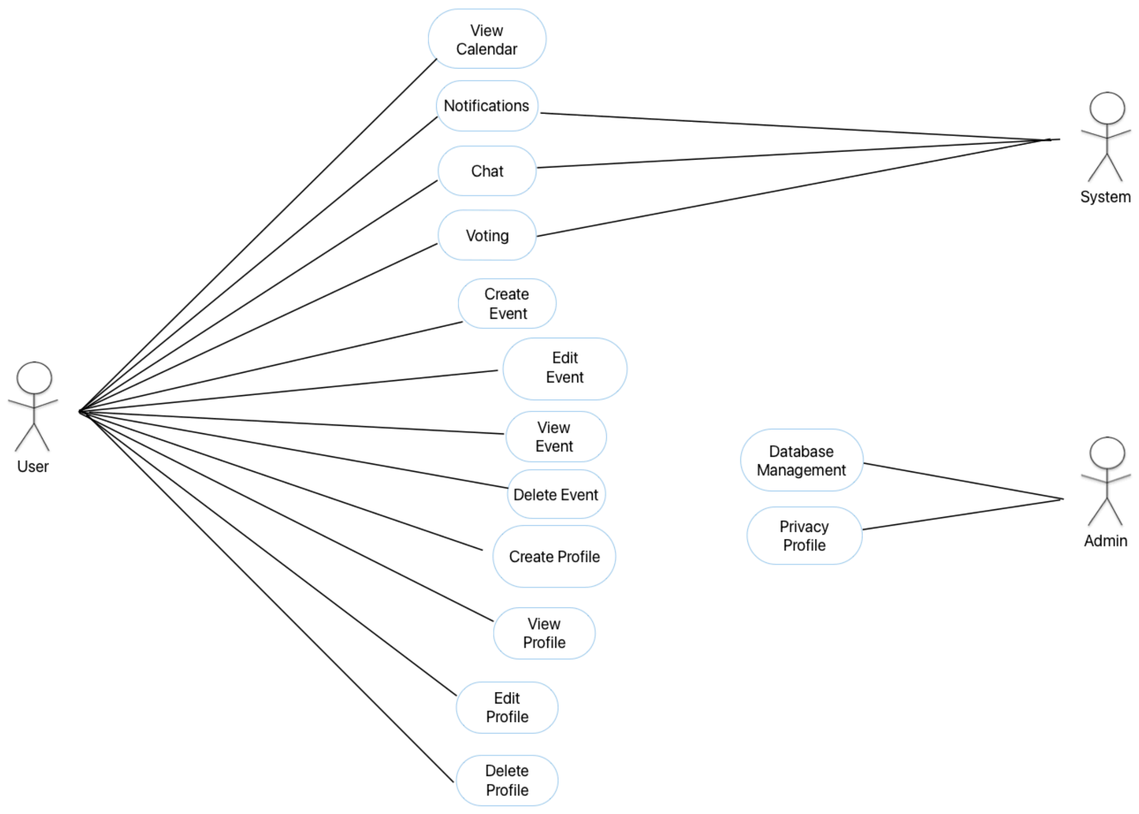
[1]

[1] Figure 1

1.3 Actor Identification

Actor	Description
Registered user	A person who has created a profile and interacts with features like event creation, polling, and chat
Guest/User Invitee	A user who is invited to events and may participate in polls without full account
System Scheduler	Background job handler for sending notifications and executing reminders
External Calendar API	Google/Outlook/Apple calendar services used for syncing events

1.3.1 Use Case Diagram



2

[2] Figure 2

1.4 Design Rationale

1.4.1 Architectural Style

Three-Tier Architecture: Separation of concerns between the user interface, application logic, and data storage layers allows scalability and maintainability.

Presentation Layer: React.js frontend for an intuitive UI

Application Layer: Python REST API (Django)

Data Layer: PostgreSQL

1.4.2 Design Pattern(s)

Name	**Observer Pattern**
Description	The Observer pattern is used in the Group Scheduling Application to automatically update user interfaces and notifications.
Example	When an event is created or modified, observers (such as notification services, calendars, or active user interfaces) receive real-time updates to reflect the latest state, keeping all relevant views synchronized.
Advantages	Provides immediate synchronization across different system components, enhancing user experience through quick updates and interactions.

Name	**Model-View-Controller (MVC)**

Description	MVC pattern separates data management (Django models), user interface (React.js views), and interaction logic (REST API controllers).
Example	Django models handle data operations and storage, React views present information and user interactions, while controllers manage requests and responses between views and models.
Advantages	Facilitates parallel development, clear separation of responsibilities, easier maintenance, and better scalability by isolating components.

Name	**Client-Server**
Description	The application employs a client-server architecture, with React frontend clients sending requests to the Django backend server.
Example	The backend manages centralized resources, event data, user authentication, and other critical services accessed through REST APIs.
Advantages	Centralized data management simplifies security and maintenance, enables load balancing and scalability, and enhances maintainability.

Name	**Singleton Pattern**
Description	The Singleton pattern ensures certain system components exist only as one instance throughout the application's lifecycle.
Example	The scheduling service managing reminders and notifications is instantiated as a singleton to ensure consistent state and coordinated access to shared resources.
Advantages	Provides global access to shared resources, maintaining state consistency and controlling access.

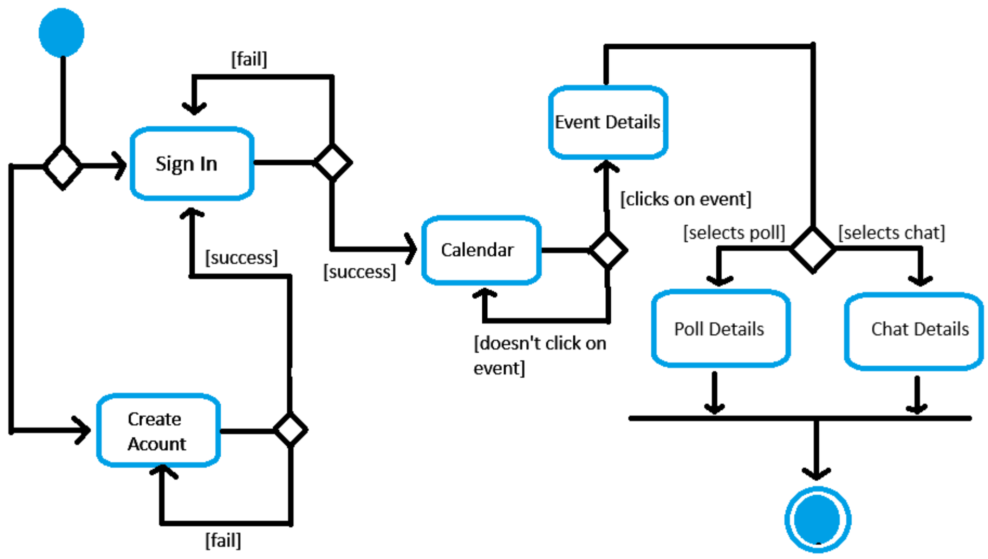
Name	**Factory Method**
Description	The Factory Method pattern dynamically creates objects.
Example	Different types of notification messages (email, SMS, push notifications) are generated through a factory method, allowing new notification types to be easily integrated without modifying existing code
Advantages	Increases flexibility, promotes extensibility, and simplifies object creation management.

1.4.3 Framework

The Group Scheduler application will utilize three frameworks in order to implement the frontend and backend. React will be used to create modular and reproducible and modern front end components. These components will query the backend through API’s using the REST Framework. Django will be used as the backend because it allows for ease of migration from an SQLite database for development to a postgreSQL database better suited for production. An SQLite database is better for development because it's easier to share, living in a single file, and also requires no setup while PostgreSQL is better for production because it supports horizontal scalability allowing replication, partitioning, and many more functions across many servers.

2. Functional Design

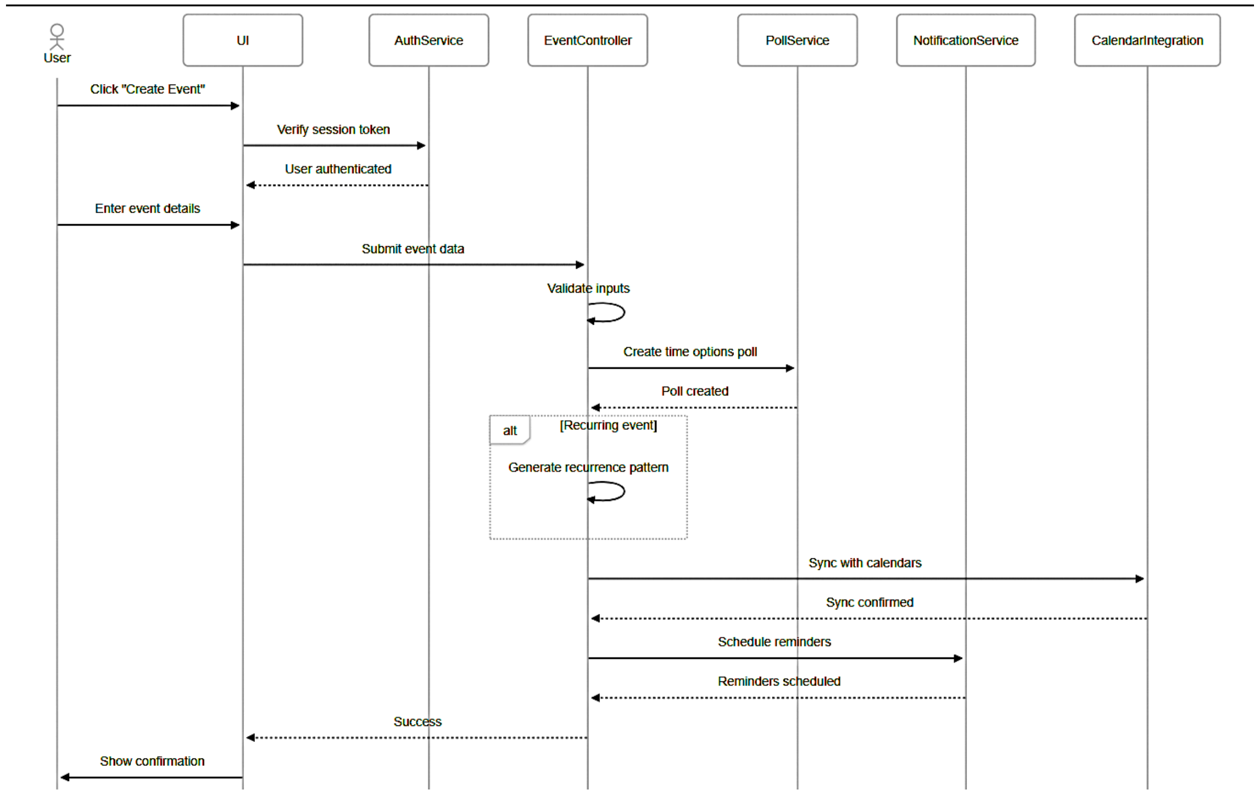
2.1 Activity Diagram



[3]

[3] Figure 3

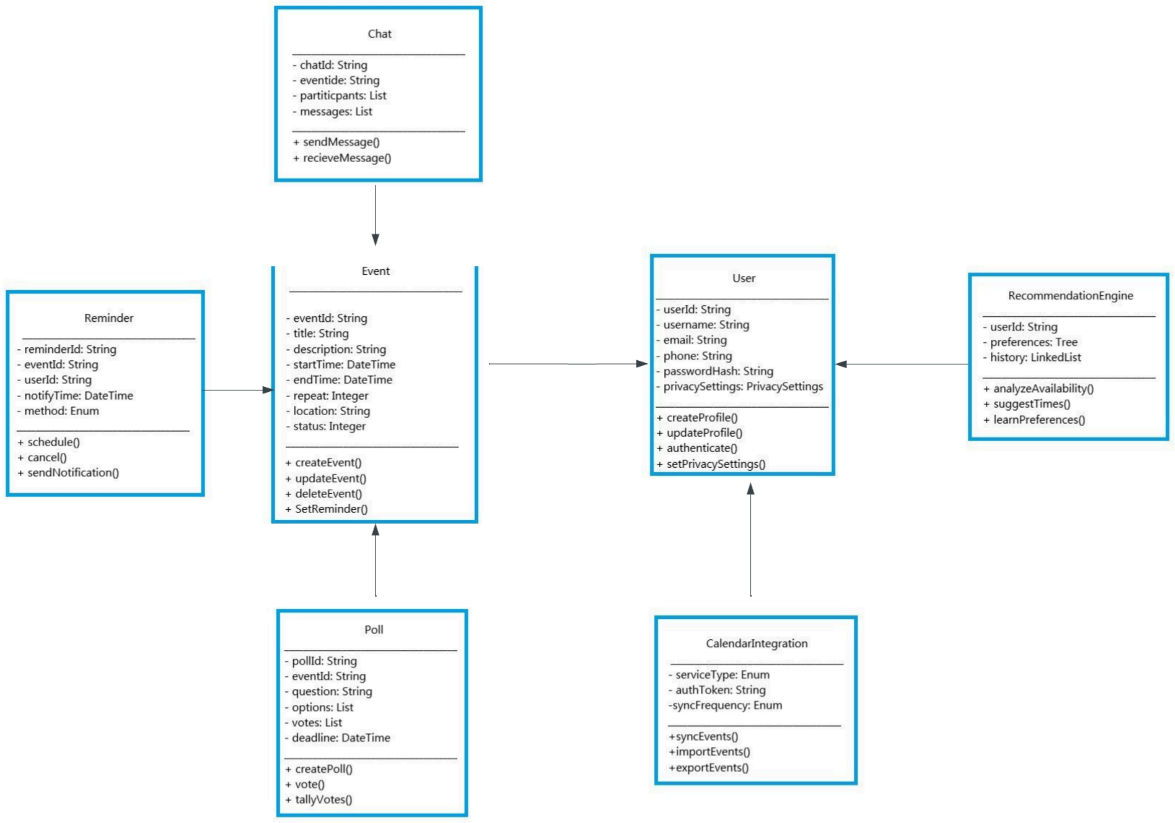
2.2 Sequence Diagram



[4]

[4] Figure 4

3. Structural Design



[5]

[5] Class Diagram