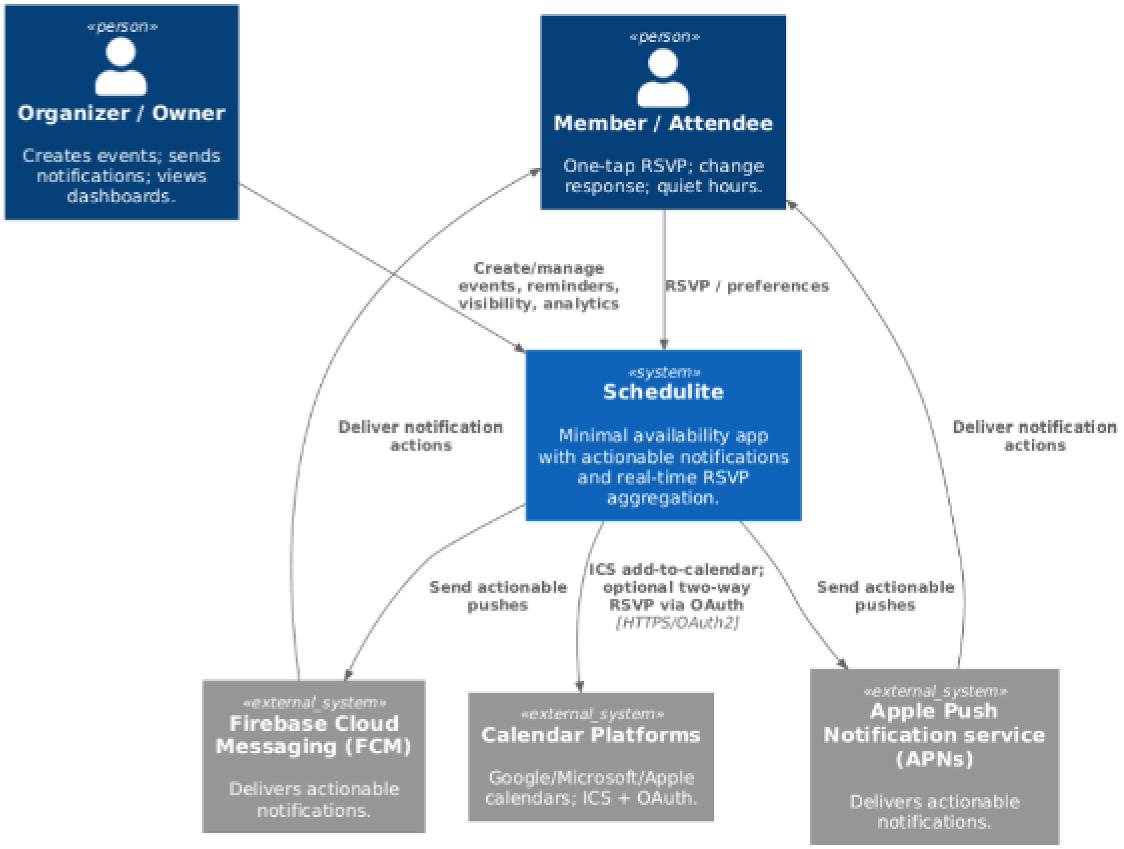
Schedulite - Design Document

ECE 49595 – Fall 2025

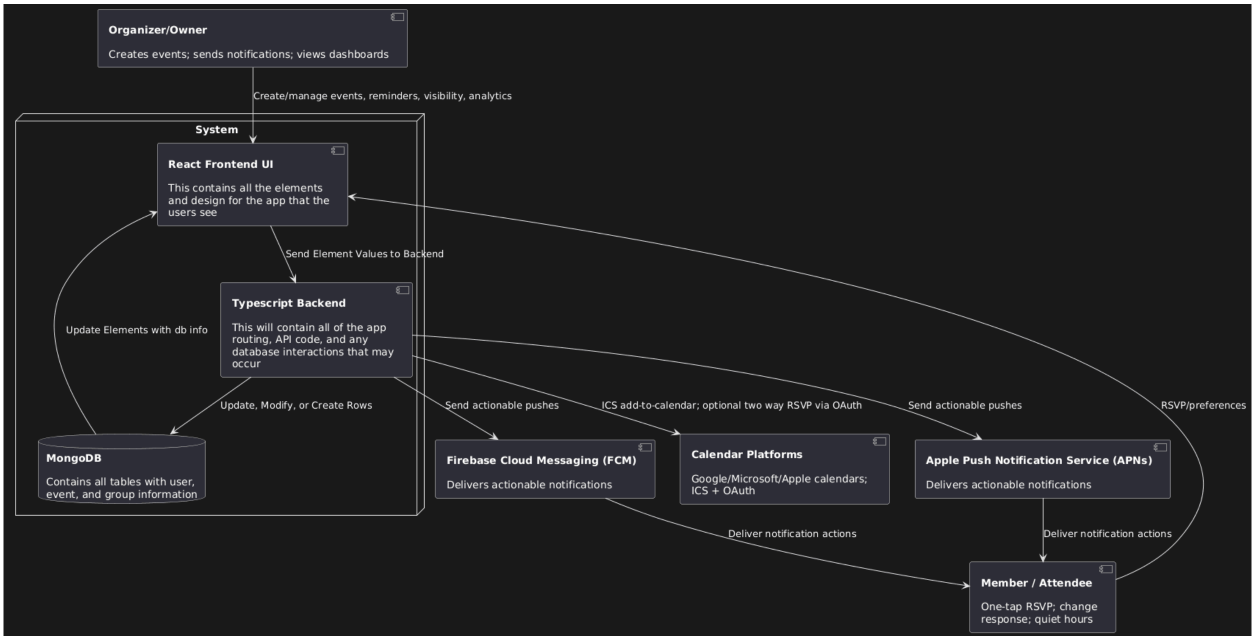
Team 09 – Luca Ricci, Aryan Banerjee, Jacob Jackson

Design Diagrams

System Context Diagram


«person»
Organizer / Owner
Creates events; sends notifications; views dashboards.
«person»
Member / Attendee
One-tap RSVP; change response; quiet hours.
Create/manage events, reminders, visibility, analytics
RSVP / preferences
«system»
Schedulite
Minimal availability app with actionable notifications and real-time RSVP aggregation.
Deliver notification actions
Deliver notification actions
Send actionable pushes
ICS add-to-calendar; optional two-way RSVP via OAuth
[HTTPS/OAuth2]
Send actionable pushes
«external_system»
Firebase Cloud Messaging (FCM)
Delivers actionable notifications.
«external_system»
Calendar Platforms
Google/Microsoft/Apple calendars; ICS + OAuth.
«external_system»
Apple Push Notification service (APNs)
Delivers actionable notifications.

Container Diagram


Organizer/Owner
Creates events; sends notifications; views dashboards
Create/manage events, reminders, visibility, analytics
System
React Frontend UI
This contains all the elements and design for the app that the users see
Send Element Values to Backend
Typescript Backend
This will contain all of the app routing, API code, and any database interactions that may occur
Update Elements with db info
Update, Modify, or Create Rows
MongoDB
Contains all tables with user, event, and group information
Send actionable pushes
ICS add-to-calendar; optional two way RSVP via OAuth
Send actionable pushes
RSVP/preferences
Firebase Cloud Messaging (FCM)
Delivers actionable notifications
Calendar Platforms
Google/Microsoft/Apple calendars; ICS + OAuth
Apple Push Notification Service (APNs)
Delivers actionable notifications
Deliver notification actions
Deliver notification actions
Member / Attendee
One-tap RSVP; change response; quiet hours

Technology Stack

Tech Name	**Description**
Typescript (nodeJS)	Will consist of most of the backend, including most if not all of the database interactions
Express.js	Webapp framework that will be used for our API's and our app routing: event route, group route, event creation route
OAuth2.0	Considering using this for our admin logins, may or may not also be used for user logins
helmet.js	Adds an additional layer of security through HTTP Protections
domPurify	Filters user input and validates it, ensuring no SQL, HTML, or JS injection
React	Will be used for most of the frontend, providing our UI elements and overall design/user experience
Expo Framework	A framework that assists in IOS and Android app development

MongoDB	NoSQL database hosting that will have our entire db structure
Google Cloud	A webhost that will host the web app portion of Schedulite

Figma

https://www.figma.com/design/EqEVKRYzQXblXNvXiscXoe/Final-Design-Demo?node-id=0-1&t=ZViW7Lx5FtrkBIRS-1

Design Trade

Database (Aryan)

Alternatives

1. **MongoDB:** NoSQL database with JSON like documents. It works well with flexible event and RSVP data. Furthermore it has a good free tier, and is easy to use from NodeJS and TypeScript.
2. **PostgreSQL:** SQL Relational database with strong constraints and clear schemas. Mature ecosystem and lots of tools for development. Efficient for long-term structure, but needs more setup, migrations, and joins in our backend code.
3. **Google Drive:** Store each group as a JSON file on Google Drive using OAuth. Very cheap and scales with users, but reads and writes are slower and integration is more complex.

Evaluation Table:

Criterion	Weight	MongoDB (DB-A)	Postgres (DB-B)	Google Drive (DB-C)
Read latency	20%	8	8	3
Write reliability	15%	8	9	5
Concurrent write behavior	15%	8	8	3
Schema flexibility	15%	9	6	7
Development complexity	15%	9	6	4
Cost total	10%	8	8	9
Scaling readiness	10%	8	8	6
Weighted Total	100%	83.0%	75.5%	49.5%

Why MongoDB is Chosen:

We chose MongoDB because it gives us an easy way to store users, groups, and events with enough flexibility to manage changing RSVP’s with minimal joins. It also has good performance for most metrics and fits our event and RSVP dashboard use cases. Furthermore the free tier fits our student project development plan smoothly. Postgres would give us stronger constraints, but it would cost us more time in schema design and migrations, and it is not as flexible for our use case. The Google Drive option is interesting for cost and data ownership, but the slower I/O and more complex OAuth flow make it risky for real time and concurrent updates. Overall, MongoDB balances speed, flexibility, and development effort the best for our current goals.

Backend Routing Language (Luca)

Alternatives

1. **TypeScript:** TypeScript compiles to JavaScript and adds static typing. It keeps the Node ecosystem but helps catch bugs at compile time and makes route logic easier to maintain.
2. **JavaScript:** Normal JavaScript is simple and familiar, with the same runtime performance as TypeScript, but it doesn’t have built in type checking, so it’s easier to accidentally add bugs and errors as the project grows.
3. **Java:** Java is fast and used a lot for production backends, but it has a lot more boilerplate code. There is a lot of setup as well, which makes it overkill for Schedulite.
4. **Python:** Python is easy to write and has many good web frameworks like Flask, but it is usually slower and less resource efficient than Node, which can matter if we scale up in the future.

Evaluation Table

Criterion	Weight	TypeScript	JavaScript	Java	Python
Speed	1.0	7	7	8	5
Resource efficiency	1.0	8	8	5	6
Ease of use	0.5	8	6	7	10
Total	N/A	19	18	16.5	16

Why TypeScript is chosen:

We chose TypeScript because it keeps the same performance and works with a lot of the same tools as JavaScript, but with types that help us write safer code. Since a lot of our logic involves passing objects between routes and the database, type checking makes refactoring and extending code safer. JavaScript alone would work, but we would lose this safety net for basically no gain. Java and Python are great languages, we love them but they do not match our existing stack and would slow us down with different frameworks and more boilerplate code to setup what we want. TypeScript gives us a good mix of speed, productivity, and maintainability for this project.

Push Notification System (Jacob)

Alternatives

1. **Direct APNs + FCM(PN-A1):** Backend talks directly to Apple APNs for iOS and FCM for Android. Gives the most control and potentially lowest latency, but doubles the integration work and maintenance.
2. **Unified FCM Gateway(PN-A2):** Backend only talks to FCM, which then relays to Android and APNs. Simpler than managing both providers directly, but still requires us to manage tokens and retries.
3. **Expo Push Service(PN-A3):** Clients register Expo push tokens, and our backend calls Expo's API. Expo then fans out to APNs and FCM, which cuts down our code and still supports actionable notifications.

Evaluation Table

Criterion	Weight (%)	PN-A1	PN-A2	PN-A3
Latency	30	9	8	7
Reliability	25	9	8	8
Integration complexity	15	5	8	10
Security and privacy	15	9	8	8
Scalability	15	8	9	7
Weighted total	100	795	800	**815**

Why Expo is chosen:

We chose Expo Push Notification Service because it lets us ship reliable, cross-platform notifications without writing a ton of provider specific code. The latency and reliability are good enough for our goal of fast RSVPs, and Expo hides most of the APNs/FCM complexity. Direct

APNs + FCM gives slightly better theoretical control, but it would cost us way more time to implement and debug. The FCM gateway is a middle ground, but still leaves us responsible for more low level details. Expo frees us up to focus on building the rest of Schedulite while still meeting our performance and reliability targets.