

Schedulite

Initial Software Development Plan

ECE 49595-046 SD Fall 2025

TEAM 09

Luca Ricci - riccil

Aryan Banerjee - banerj61

Jacob Jackson - jacks921

1: Project Overview

The purpose of the Schedulite project is to create an application that declutters the act of scheduling an event with others. The application will feature actionable notifications for users to RSVP for events in groups they are members of. Group owners can also create those events with a date, time, location and a brief description.

2: Scope

In Scope

CAP-1 Groups & Roles: Users can create new groups, join existing groups using a code/link, and Owners can assign Organizer or Member permissions to control event creation and modification rights.

CAP-2 Event Creation & Metadata: creation of events including title, start time, location, and notes. Includes full CRUD (Create, Read, Update, Delete) capabilities.

CAP-3 Actionable Notifications & RSVP: Push notifications that allow users to select "In," "Out," or "Maybe" directly from the lock screen without opening the full app.

CAP-4 Availability Dashboard: live tallies of responses from users and a timeline of when responses were recorded.

CAP-5 Event Detail View: A detailed in app view showing the full description of the event, and the ability for a user to edit their previous RSVP status.

Under Evaluation:

CAP-6 Calendar Integration: Integration with Google/Microsoft/Apple to automatically add events to calendar.

CAP-7 Reminders & Rate Limiting: automated reminder notifications to non responders with strict rate limits to avoid spamming.

CAP-8 Offline Operation & Sync: Local storage of RSVP actions taken while offline, with an automated sync when the user restores internet.

Out of Scope:

- Comprehensive chat features: The app focuses on scheduling, so communication is limited to event notes to avoid spam and clutter.
- Detailed event planning: Complex logistics planning requires back and forth communication which again creates unnecessary clutter for attendees.
- Monetization (paid subscriptions): no paid subscriptions or models are planned
- Machine learning integration: Machine Learning and LLM's are too unreliable for an app that necessitates strict headcounts and prioritizes accuracy.

Assumptions:

- **User Hardware:** We assume the user possesses an IOS or Android phone capable of receiving push notifications, or a computer for the web view.
- **Internet Access:** We assume the user has intermittent access to the internet. While offline queuing is a feature, the core feature of receiving invitations relies on an internet connection at some point.
- **Third party service stability:** We assume that MongoDB, Apple Push Notifications, and Google OAuth services maintain stable and consistent service through development and deployment.
- **Users Consent:** We assume users are willing to grant permission for push notifications as this is the primary interaction vector for the app.

Constraints:

- **Fixed Academic Timeline:** The project must be completed and deployed by the end of the Fall 2025 semester. This necessitates a prioritization of core RSVP flows over extra features.
- **Team skill set:** The team must operate within our current proficiency in React Native and Node.js/TypeScript, or factor in time to learn new skills for the development phase.
- **Security requirements:** We must ensure user data, specifically contact info and private event data, is encrypted in transit and storage to meet standard privacy expectations.
- **Cross-platform limitations:** The UI/UX must remain somewhat consistent across iOS and Android, as we are serving to multiple platforms and all users deserve equal service.
- **Budget limitations:** The project currently has a \$0 budget, restricting us to free tiers of APIs and hosting services for development, and a minimal deployment cost from our own pockets.

3: Requirements

3.1 Functional Requirements

- **ID:** FR-1
- **Statement:** The system shall allow a user to register and sign in using OAuth 2.0 authentication
- **Rationale:** OAuth 2.0 provides secure authentication without requiring the team to manage passwords or pay for SMS verification, and most users already possess Google accounts.

- **Test Method:** Configure Google OAuth 2.0 client ID and perform tests on Registration, Log In, and Error conditions. We will perform 5 registration events checking whether user data and tokens are stored correctly in our database. Then we will perform 5 successful login attempts and 5 invalid login attempts with invalid tokens and incorrect passwords to verify error handling.
- **Supporting Context:** The backend calls the Google OAuth endpoint so users can do a google sign in and it provides ID tokens. The tokens expire per provider policy and are refreshed silently so the OAuth API just needs to be interfaced with correctly.
- **Trace:** CAP-1
- **Priority:** Must

- **ID:** FR-2
- **Statement:** The system shall let users create groups
- **Rationale:** Groups are the fundamental container for events, without them, invitations cannot be sent to the right people.
- **Test Method:** Create 5 unique groups via the API and verify that the database records the creator as the 'Owner' for each.
- **Supporting Context:** Groups are stored in the database and link the users and events that are part of the group. Upon creation, the user's ID is automatically added to the 'members' array with the role 'Owner'.
- **Trace:** CAP-1
- **Priority:** Must

- **ID:** FR-3
- **Statement:** The system shall let users join an existing Group using a group code
- **Rationale:** Users need a simple way to enter a group roster.
- **Test Method:** Generate join codes for 3 groups. Have 5 distinct test accounts attempt to join using valid and invalid codes. Verify the valid users appear in the member list.
- **Supporting Context:** Users can join using a 6 digit code generated by the backend and linked to the Group ID. It allows users to query and join the group without explicit email invites.
- **Trace:** CAP-1
- **Priority:** Must

- **ID:** FR-4

- **Statement:** The system shall allow the Group Owner to assign 'Organizer' and 'Member' roles.
- **Rationale:** Delegation of event creation requires a role with more permissions than a base user.
- **Test Method:** In a test group, the Owner attempts to promote a Member to Organizer. Verify the Organizer can subsequently create an event, while a standard Member cannot.
- **Supporting Context:** Role information is stored in the database within the groups schema. There can only be one Owner. Organizers can add members and create/edit events. Members can only RSVP and view event information.
- **Trace:** CAP-1
- **Priority:** Must

- **ID:** FR-5
- **Statement:** The system shall let authorized users (Owners/Organizers) create an Event with required metadata: Title, Start Time, and Location.
- **Rationale:** Standardized metadata is required so that notifications are consistent and readable.
- **Test Method:** Perform 5 Event Creation API calls. Verify that if 'Title' or 'Start Time' is missing, the system returns a 400 Bad Request error. Verify valid events are stored in the DB.
- **Supporting Context:** An owner or organizer creates these events with all necessary details. The event object is stored in the database, and the RSVP data structure is initialized and tied to this event ID.
- **Trace:** CAP-2
- **Priority:** Must

- **ID:** FR-6
- **Statement:** The system shall allow authorized users to Update and Delete events (CRUD).
- **Rationale:** Plans change; organizers must be able to correct errors or cancel events.
- **Test Method:** Create an event, then update the title, time, description or other metadata. Verify that the database reflects the change. Then delete the event and verify it is no longer retrievable.
- **Supporting Context:** Updates use the patch routes to modify specific fields without overwriting the entire event. Deletes use the delete route to delete events from the database.
- **Trace:** CAP-2

- **Priority:** Must

- **ID:** FR-7
- **Statement:** The system shall send actionable push notifications containing the Event Title and Time, accompanied by interactive 'In', 'Out', and 'Maybe' buttons.
- **Rationale:** One tap responses increase the odds of a person actually RSVP'ing and the context allows people to make the choice whether they can attend or not easily.
- **Test Method:** Send 20 notifications to both iOS and Android test devices with different event title lengths and different times. 20 notifications should give enough variability in times and title lengths to test most cases. Verify that the information is displayed correctly and that expanding the notification reveals the buttons and that tapping a button sends a response to the server correctly.
- **Supporting Context:** Use the Apple push notification service and Android notifications to send these notifications on event creation.
- **Trace:** CAP-3
- **Priority:** Must

- **ID:** FR-8
- **Statement:** The system shall allow users to respond with an "In", "Out", or "Maybe" response with an optional note.
- **Rationale:** This is the other core user flow where attendees can actually respond to created events with a RSVP. Furthermore, having a short context note improves planning accuracy and allows organizers to clarify any doubts.
- **Test Method:** Create an event and RSVP with 6 different accounts. The responses should include all possibilities: In, Out, Maybe, In + Note, Out+Note, Maybe + Note. All 6 response should be correctly recorded in the database
- **Trace:** CAP-3, CAP-5
- **Priority:** Must

- **ID:** FR-9
- **Statement:** The system shall allow a user to edit their response (e.g., change 'Maybe' to 'In') and attach an optional note.
- **Rationale:** Availability changes; users must not be locked into a preliminary response.
- **Test Method:** User A responds 'Maybe' with a note. User A then updates status to 'In'. Verify the database updates the status to 'In' and retains/updates the note history.
- **Supporting Context:** Notes are limited to ≤300 characters and are stored within the RSVP document in the database. Organizers can view these notes in the dashboard.
- **Trace:** CAP-3, CAP-5
- **Priority:** Should

- **ID:** FR-10
- **Statement:** The system shall display an Availability Dashboard with live tallies by response (In/Out/Maybe) and a timestamped response list to the organizers.
- **Rationale:** Organizers need real-time headcounts and response history to help with planning and to hold members accountable.
- **Test Method:** Create a group with 20 members and over the course of 20 minutes, create RSVP responses. At least 10 members should modify their responses, and 1 should not respond at all. Verify dashboard updates within 2 seconds of server acknowledgment and accurately represent the actual state of responses. The Dashboard should maintain a response list which logs these responses with the time they were made.
- **Supporting Context:** We can use web sockets and event hooks to capture the RSVP's and update the dashboard on responses in real time.
- **Trace:** CAP-4
- **Priority:** Must

- **ID:** FR-11
- **Statement:** The system shall let Organizers send reminders to non responders and "Maybe" users with rate limiting.
- **Rationale:** Reduces manual chasing of people by allowing organizers to schedule, or manually send reminders to non-responders and maybe users while also preventing overwhelming notifications through the rate limiting.
- **Test Method:** Create an event in a group with at least 5(1 organizer, 4 members) people, and have 1 member respond with "In", 1 with "Out", 1 with "Maybe", and 1 not respond. An organizer should set a rule to remind all non responders a day before the event. Verify that the reminder gets sent and received by the non responder and "Maybe" members. Then have the organizer try and manually send a reminder again within 10 minutes, and after 13 hours. The first attempt should get rate limited, while the second should be sent and received.
- **Supporting Context:** Default per user per event, max notifications is 1 every 12 hours.
- **Trace:** CAP-7
- **Priority:** Should

- **ID:** FR-12
- **Statement:** The system shall queue RSVP actions taken while offline and sync them automatically when connectivity is restored, preserving timestamps.
- **Rationale:** Sometimes, users might view their mobiles on unstable networks so they should still be able to respond on their end to provide the smoothest UX, and that response should be automatically synced.

- **Test Method:** In airplane mode, have 5 users perform RSVPs and restore the network, All responses should be synced correctly with the server within 2 seconds. The dashboard should show timestamps of when the person responded and when the response was registered.
- **Supporting Context:** Client side and server side queues with a web hook on the request so that the server updates on every new update.
- **Trace:** CAP-8
- **Priority:** Should

- **ID:** FR-13
- **Statement:** The system shall export attendance summaries as CSV including member, role, response, note, and timestamps.
- **Rationale:** Organizers need records and sharable reports for more professional meetings or groups.
- **Test Method:** For an event with ≥ 15 members, export CSV and verify all metadata such as event data, time of snapshot exists and that the rows are correct against the actual dashboard at the given time.
- **Supporting Context:** Will add a feature to export the event data stored in the database as a CSV and provide an option to do so from the dashboard.
- **Trace:** CAP-4, CAP-5
- **Priority:** Should

- **ID:** FR-14
- **Statement:** The system shall provide an option to link a user's calendar and add events to it.
- **Rationale:** Reduces context switching and missed events along with allowing a user to maintain their existing workflows.
- **Test Method:** Have a user join 3 groups and create 2 events in each group and the user should import to Google and Apple Calendar. This should create the event in their calendar. Event edits should show up on the calendar within 15 minutes and putting "Out" should remove it from the calendar.
- **Supporting Context:** Google Calendar, and Apple Calendar API integrations will be needed and the information will need to be synced with app CRUD operations.
- **Trace:** CAP-6
- **Priority:** Could

3.2 Non-Functional Requirements

- **ID:** NFR-1 (Performance)

- **Statement:** The system shall process a user's RSVP action from the time they tap a notification or in-app RSVP button to the time the server stores the response and the client shows the updated state in 2 seconds or less for at least 95% of requests, when at least 100 distinct users submit responses across at least 10 active events within a 60 second period under normal network conditions.
- **Rationale:** Fast and accurate RSVP processing makes the app feel responsive and reliable, which is critical because the main value of Schedulite is quick "In / Out / Maybe" answers from notifications on iOS and Android.
- **Test Method:** Simulate 100 test accounts sending RSVP actions to 10 different events over 60 seconds from iOS and Android clients. Log the time when each client sends the request and when it receives the server response. Verify that at least 95% of RSVP requests complete in 2 seconds or less, and that no major spikes occur during the test.
- **Supporting Context:** "Normal network conditions" means Wi-Fi or cellular connections with stable signal (no airplane mode or forced packet loss). This requirement applies to both iOS and Android apps, as well as the backend route that records RSVP responses.
- **Trace:** CAP-3
- **Priority:** Must

- **ID:** NFR-2 (Maintainability)
- **Statement:** The backend services shall maintain at least 80% line coverage from automated tests, and these tests shall run automatically for every pull request before changes are merged into the main branch.
- **Rationale:** High automated test coverage makes it easier to refactor code, fix bugs, and add new features without breaking existing behavior.
- **Test Method:** Run a coverage tool like Jest with coverage report in the continuous integration pipeline and verify that the overall line coverage for backend services stays at or above 80%. The pipeline shall block merges when coverage falls below this threshold until additional tests are added. The 80% rule ensure the codebase does not have failing code, or useless code.
- **Supporting Context:** "Backend services" here include the authentication, group management, event creation, RSVP, and notification APIs. Documentation for these services (such as API route descriptions and setup instructions) will be maintained in the repository, but it is not part of this specific non-functional requirement.
- **Trace:** CAP-1 , CAP-2, CAP-3
- **Priority:** Should

- **ID:** NFR-3 (iOS Delivery Reliability)

- **Statement:** The system shall ensure that at least 97% of push notifications sent to iOS devices appear on the device within 60 seconds of being queued by the backend under normal network conditions.
- **Rationale:** Timely notifications are critical for one-tap RSVP. iOS uses APNs, which has its own delivery constraints and needs to be validated separately from Android.
- **Test Method:** Send 200 push notifications to iOS test devices through APNs. Use backend logs and client-side timestamps in the iOS test build to measure delivery time. Verify that at least 97% appear within 60 seconds.
- **Supporting Context:** This requirement strictly concerns Apple Push Notification Service (APNs). It excludes Android and backend constraints.
- **Trace:** CAP-3, CAP-7
Priority: Must

- **ID:** NFR-4 (Android Delivery Reliability)
- **Statement:** The system shall ensure that at least 97% of push notifications sent to Android devices appear on the device within 60 seconds of being queued by the backend under normal network conditions.
- **Rationale:** Android devices vary widely in manufacturer behavior and background limitations, so notification delivery needs its own platform-specific requirement.
- **Test Method:** Send 200 push notifications to Android test devices through FCM. Use backend logs and Android test build timestamps to measure delivery. Verify that at least 97% appear within 60 seconds.
- **Supporting Context:** This requirement strictly concerns Firebase Cloud Messaging (FCM) for Android.
- **Trace:** CAP-3, CAP-7
Priority: Must

4: Deliverables

Deliverable 1: IOS Mobile Application

Description: A functioning Schedulite iOS application published on the Apple App Store. The app shall allow users to create accounts, join and form groups, receive push notifications through APNs, and submit one-tap RSVP responses.

Relevant Requirements: FR-1 through FR-7, NFR-1, NFR-4 (iOS Delivery Reliability)

Deliverable 2: Android Mobile Application

Description: A functioning Schedulite Android application published on the Google Play Store. The app shall provide the same core functionality as the iOS app, including group creation, event creation, and one-tap RSVP, with notifications delivered through FCM.

Relevant Requirements: FR-1 through FR-7, NFR-1, NFR-5 (Android Delivery Reliability)

Deliverable 3: Backend Service Deployment

Description: A deployed backend server that implements authentication, group management, event creation, RSVP handling, notification orchestration, and persistent data storage. The backend shall be hosted on a cloud environment and accessible by both mobile applications through secure API routes.

Relevant Requirements: FR-1, FR-2, FR-3, FR-4, FR-6, NFR-1, NFR-3

Deliverable 4: Technical Documentation

Description: Developer-facing documentation containing API specifications, route definitions, database schemas, notification flow diagrams, environment setup steps, CI/testing instructions, and versioning guidelines so future developers can maintain and extend the system.

Relevant Requirements: NFR-3 (Maintainability)

Deliverable 5: User Documentation

Description: Simple onboarding documentation or user help pages that explain how to create an account, join groups, view events, and respond to notifications inside the app.

Relevant Requirements: FR-1 through FR-7

Deliverable 6: Marketing & User Adoption Plan

Description: A concise marketing plan outlining how the team will promote Schedulite to early adopters such as student clubs, sports teams, and organizations at Purdue. The plan shall describe outreach channels, value proposition messaging, and a strategy for building the initial user base.

Relevant Requirements: Supports overall system deployment but is not tied to specific FR/NFR; included to meet project expectations

5: Development Methodology

The development methodology our team chose for Schedulite's development is the Kanban methodology. This methodology seems to be a strong fit for our team because it allows for our project to have multiple parallel workstreams and gives us a lightweight way to keep the work moving without too much overhead. This methodology also works well with evolving requirements and feedback, which we anticipate will be a recurring theme throughout the development process. Not only that, but it also allows for a prototype-first workflow, where we can slice our development into small increments and continuously deliver improvements.

6: Verification and Validation Plan

<https://github.com/ECE495-Team09/Schedulite/blob/b72cdb45cebb0307995441466fa97145eb79454d/Verification%20and%20Validation.pdf>

7: Gantt Chart

https://docs.google.com/spreadsheets/d/1pWjNw_1AKV5KajuyIOCQTA8cHX_yzvnnqDEdHp-t-tM/edit?usp=drivesdk