## Objectives

To keep verification and validation focused on Schedulite's main goals, we will tie each objective to product capabilities in our RVTM and SDP. Here are the capabilities and their requirements.

- CAP-1: Accounts and groups (FR-1, FR-2)
- CAP-2: Event creation (FR-3)
- CAP-3: Push notifications and RSVP actions (FR-4, FR-5)
- CAP-4: Availability dashboard (FR-6)
- CAP-5: Event details and notes (FR-5)
- Additional SDP capabilities: calendar integration and offline handling.

We intend to verify and validate at least the following measurable objectives:

Objective 1: One-tap RSVP latency and interaction cost
Verify that a registered member can submit an In / Out / Maybe response from a push notification on iOS and Android in ≤3 taps and with median end-to-end time ≤3 seconds when ~20 users respond to the same event.

- Linked capabilities: CAP-2 (FR-3, event creation feeding events), CAP-3 (FR-4, RSVP actions).

Objective 2: Notification content and cross-platform consistency
Verify that when an organizer creates or updates an event, notifications on iOS and Android show the correct title, start time, location, and RSVP state for at least 95% of a 50-event sample, and that the web details view reflects the same information within 5 seconds under normal network conditions.

- Linked capabilities: CAP-2 (FR-3), CAP-3 (FR-4), CAP-5 (FR-5, details).

Objective 3: Headcount accuracy on the availability dashboard
Verify that the dashboard's In / Out / Maybe counts match stored responses exactly (0

discrepancies) for events with 5–100 members, across at least 30 events that include response edits and late changes.

- Linked capabilities: CAP-3 (FR-4, FR-5, RSVP data), CAP-4 (FR-6, dashboard).

Objective 4: Role-based permissions for event management
Verify that only Owners and Organizers can create, edit, or delete events and send RSVP notifications, and that Members can only view event information and manage their own responses. Across at least 30 permission test cases, no unauthorized action should succeed (0 false positives) and ≥95% of valid actions should succeed.

- Linked capabilities: CAP-1 (FR-2, roles), CAP-2 (FR-3), CAP-3 (FR-4).

Objective 5: Reliable offline RSVP queueing and sync
Verify that if a member responds while offline, the app stores the choice locally and syncs it within 30 seconds of reconnection, without duplicates or lost responses, in at least 20 offline scenarios across iOS and Android.

- Linked capabilities: CAP-3 (RSVP actions), SDP offline handling.

Objective 6: Calendar integration correctness
Verify that "Add to calendar" from the event details view creates events in Google Calendar or Apple Calendar with the correct title, date, start time (and end time if set) and location for at least 40 test events, with 0 critical mismatches (wrong date or time).

- Linked capabilities: CAP-5 (details view), SDP calendar integration.

These objectives are specific enough that we can attach concrete test cases and keep them traceable back to CAP and FR entries.

**Prioritization**

| **Product Capability** | **Priority (High, Med, Low)** | **Brief Justification** |
|---|---|---|
| CAP-3: Push notifications and RSVP actions (FR-4, FR-5) | High | This is the core value of Schedulite. If notifications or one-tap RSVP are slow, missing, or confusing, users |

| | | |
|---|---|---|
| | | will not adopt the app. It also spans mobile clients, push infrastructure, and backend APIs, so both risk and impact are high. |
| CAP-4: Availability dashboard (FR-6) | High | Organizers rely on accurate headcounts to book space and make decisions. Wrong tallies directly affect real events. The dashboard aggregates many responses in near real time, which raises complexity and makes it a high-priority target for V&V. |
| CAP-2: Event creation (FR-3) | High | Every notification and RSVP flow depends on events being created correctly. Errors here propagate everywhere else. We need to verify that required fields are stored and surfaced properly across clients. |
| CAP-1: Groups and roles (FR-2) | Medium | Role mistakes can send the wrong notifications or give the wrong users control, which is serious but less visible than RSVP failure. We will test permissions thoroughly, but after we stabilize the core RSVP and dashboard flows. |
| CAP-5: Event details and notes (FR-5) | Medium | Details and “Maybe” notes help organizers understand context and edge cases, but the app still works if this is slightly imperfect. Risk is moderate because it touches multiple fields and state changes, so we keep it Medium. |
| Reminders (CAP-7, FR-7) and calendar/offline features | Medium | Reminders, calendar integration, and offline |

| | | |
|---|---|---|
| | | behavior make the product stickier and more reliable, but they extend the core flows rather than define them. We will verify them with focused tests, but not at the same depth as core RSVP and dashboard. |

**Full RVTM**

| Product Capability | Requirement ID | Requirement (Shall Statement) | Supporting Context | Test Case ID | Test Description | Impacted Components | Additional Comments |
|---|---|---|---|---|---|---|---|
| CAP - 1 | FR -1 | The system shall allow a user to register and sign in using OAuth 2.0 authentication | 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. | T - 1 | Configure Google OAuth 2.0 client ID and test 5 sign in attempts from unique accounts. | Database | |
| CAP - 1 | FR-2.1 | The system shall let users create groups | Users shall be able to create groups in order to send members RSVP notifications and event info. The creator of the group will be assigned the “owner” role. | T - 2.1 | Create 3 groups and verify users are able to view the groups they have created. | Database | |
| CAP - 1 | FR-2.2 | The system shall let users join groups | Users shall be able to join groups created by others users in order to receive RSVP notifications and information about | T - 2.2 | Create 3 different groups and verify at least 5 users are able to join and receive | Database | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | upcoming events. | | notifications. | | |
| CAP - 1 | FR-2.3 | The system shall let users assign Owner, Organizer, and Member roles with defined permissions. | Role information is stored in the database within the groups schema. There can only be one owner who owns the group and has all permissions, they can appoint organizers who can also add members and also create/edit events. Members can only RSVP and view event information. | T 2.3 | Create a group of at least 5 people and assign Owner, Organizer, and Member roles to each member, then make sure role switching is possible for the Owner to apply on members. | Database | |
| CAP - 2 | FR -3 | The system shall let users create an Event with title, start time, location, and optional notes. | An owner or organizer can create these events with all necessary details and this event will be stored in the database with RSVP data being tied to each event. | T - 3 | Create 10 events, testing different types of times, locations, and different fields. All information should be in the database with common CRUD operations working correctly. | Database | |
| CAP - 3 | FR - 4 | The system | Use the Apple push | T - 4 | Send 10 pushes to | Server | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | shall send actionable push notifications containing In/Out/Maybe actions and shortened event context and time/location on expanding the notification. | notification service and Android notifications to send these notifications on event creation. | | test devices on iOS and Android. Verify 100% of the notifications arrive on event creation and are functional with the RSVP and expand to show time/location. Make sure each role is able to receive the notification properly. | | |
| CAP - 3 & CAP - 5 | FR - 5 | The system shall allow a responder to add an optional note when selecting “Maybe” and display that note to Organizers. | Note ≤300 characters and stored with the RSVP data in the database. | T - 5 | Submit 5 “Maybe” RSVPs with notes, should be 100% visible to Organizers and the note should be editable by the responder until the event starts. | UI | |
| CAP - 4 | FR - 6 | The system | We can use web | T - 6 | With ≥20 | UI | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | shall display an Availability Dashboard with live tallies by response (In/Out/Maybe) and a timestamped response list to the organizers. | sockets and event hooks to capture the RSVP's and update the dashboard on responses in real time. | | members and ≥20 RSVPs, verify dashboard updates within 2 seconds of server acknowledgment. The response list should be ordered by time of response. | | |
| CAP - 7 | FR - 7 | The system shall let Organizers send reminders to non responders and "Maybe" users with rate limiting. | Default per user per event, max notifications is 1 every 12 hours. | T - 7 | Configure reminder rules, and ensure that no user receives >1 reminder per 12 hour window. | Server | |

**Test Approach**

Our team needs to be able to have both manual and automatic tests in order to ensure our project meets our established standards. On the backend, we'll write unit tests for smaller pieces like our data models and event logic, and integration tests that actually call the Express routes and talk to the database, which lets us catch issues with real requests and responses. On the frontend side, which will be web, iOS, and Android, we'll do more system-level testing by walking through core user flows such as creating events, sending RSVPs, and checking availability; treating it the same way a real user might. Most of the backend tests will be automated and run through npm, while a lot of the UI testing will be manual, since we want to see how it actually feels to use the app in a browser or on a phone. For non-functional testing, we plan on doing basic checks on things like how fast key endpoints respond, how the app behaves with bad input or a spotty network connection, and whether the main screens are easily understandable and reasonably accessible to new users. As we develop, we'll probably reuse the same automated tests plus a small manual regression checklist of our main user flows, so whenever we add a feature or change something, we can quickly see if we accidentally broke something.

**Validation Plan**

As we progress through creating Schedulite, we will have two main approaches to our testing procedures. For our User Acceptance Testing (UAT), we will have potential users test different parts of our app as we complete them (for example the event making process or account making process) to ensure that they are intuitive (able to be used with little to no instruction), give feedback quickly (lets say 5-10 seconds at *most*), and is visually pleasing to the user (not feeling cluttered or too compacted). For our Usability and Performance Evaluations we will be using information from both the UAT and from the tools listed in the next section below. Using these we will be sure that our app is functioning on a level that will not be inconvenient for users.

**Tools, Environments & Test Data Sets**

| **Verif/Val Tools** | **How they will be used** |
|---|---|
| Postman/Bruno | These will be used to test the API routes and ensure that they are able to send back 201 codes successfully, as well as the time it takes for those responses to be sent. |
| MongoDB | In addition to simply being our database, we will make sure to check it in tandem with postman/bruno API calls to ensure that the database is populating as intended. |

| | |
|---|---|
| Web Console | The web console will be used to ensure the frontend of our app looks to the end user as it should, as well as identifying possible areas of error within that frontend. |
| Expo Go | This will be used to ensure that our app functions correctly and looks well on an IOS or Android phone. |