Scheduling Decision Guide

Companion to the Scheduling docs.

Use this guide to decide what can be booked, who can book it, and how appointments move through your workflow. Section 1 establishes the use case, Section 2 scopes additional requirements, and Section 3 maps those requirements to modeling and implementation choices.

Medplum provides availability calculation and atomic booking operations. Your application supplies business rules such as licensure checks, patient preferences, approval workflows, and integrations. The scheduling APIs are in beta; check the beta changelog when upgrading an existing implementation.

Section 1: Use Case & Participants

1.1 Who books, and on whose behalf?

- Patient self-service
- Staff booking for a patient
- Provider managing their own appointments
- System or API-driven booking
- A combination of these

Why: determines authentication, delegated access, and which users can change availability or make exceptions (3.4, 3.9).

1.2 What must be available for the appointment?

- A clinician only
- Several concurrent resources, such as a clinician, room, and device

- A room or device without a clinician
- A shared resource with capacity greater than one
- A group session with several patients

Why: distinguishes multi-resource coordination from capacity and patient-specific workflow decisions (3.2, 3.6, 3.7).

1.3 Does the patient choose a person, or an eligible pool?

- A named practitioner or care team
- Any practitioner qualified for the service
- First available, subject to preferences
- Staff assignment after a request

Why: determines candidate selection, hard eligibility rules, preference ranking, and what "first available" promises (3.5, 3.6).

1.4 When is the appointment considered booked?

- Immediately after the time is selected
- After the patient completes another step
- After staff approval or authorization review
- After required participants respond

Why: determines whether to book immediately, reserve time with a hold, or keep a request without consuming capacity (3.8).

1.5 Is Medplum replacing a scheduler, or working alongside one?

Identify the system of record for appointments, availability, cancellations, and external identifiers. Which existing appointments and blocks must be migrated? Which system can change them after launch?

Why: coexistence needs an explicit write authority and reconciliation policy. Existing scheduling data may also require a beta migration (3.15).

1.6 Who maintains the scheduling configuration?

Who defines visit types, provider hours, resource assignments, holidays, and exceptions? Can providers edit their own hours? Who approves changes that affect appointments already booked?

Why: identifies configuration owners, administrative permissions, and change-management requirements (3.1–3.4).

Section 2: Feature Scoping

Every implementation must define its service and actor model, availability, access control, booking flow, and conflict handling. These are not optional features.

For each additional requirement below, record Yes, No, Nice-to-have, or Not sure. Follow the linked deep dive for Yes and Nice-to-have answers; assign an owner to resolve Not sure answers.

Feature	Deep dive	Answer
Site-specific hours, time zones, or provider overrides	3.1, 3.3	
Lead-time rules, rolling booking windows, or daily/weekly caps	3.3	
Licensure, service enrollment, or other eligibility requirements	3.5	
Language, practitioner, day/time, or other patient preferences	3.5	
First available or automatic actor assignment	3.5, 3.6	
Multiple concurrent actors, with named resources or alternatives	3.6	
Shared capacity, deliberate overbooking, or group visits	3.7	
Holds, staff approval, or authorization review	3.8	
Live availability refresh or privileged scheduling overrides	3.9	

Feature	Deep dive	Answer
Patient cancellation/rescheduling, participant responses, or encounter creation	3.10	
Recurring appointment series	3.11	
Waitlists and offers when capacity opens	3.12	
Reminders and two-way responses	3.13	
Telehealth provisioning	3.14	
External calendar or scheduler integration	3.15	

Section 3: Feature Deep Dives

Each section should end with a chosen approach or an open question with an owner. Use the technical docs for resource examples and operation payloads.

Lane	Subsections	When to read
Foundations	3.1–3.4	Always
Finding an appointment	3.5–3.6	Cover selection always; combinations when multiple resources are needed
Booking and managing appointments	3.7–3.10	Cover booking, conflicts, and lifecycle always; capacity when needed
Conditional workflows	3.11–3.15	When flagged in Section 2

Foundations

3.1 Visit Types and Parameter Ownership

Decide what each bookable service means and which settings are shared versus actor-specific. Use a HealthcareService for each bookable appointment

type. Its SchedulingParameters supply defaults; matching parameters on a Schedule override individual fields. See Defining Availability.

Questions:

- What distinguishes one visit type from another?
- Which duration, start-time grid, buffers, and capacity settings are shared?
- Which actors or sites need different settings, and who can change them?

Situation	Approach
Shared visit configuration	Put common parameters on HealthcareService. Put shared weekly hours in its native availableTime field.
Per-actor differences	Set only the differing Schedule parameters, with a service reference to the HealthcareService. Unset fields inherit.
Different hours by site	Consider location-specific HealthcareService offerings and per-service Schedule overrides. Separate services by location only when the distinction matters to scheduling.
Multiple resources booked together	Keep effective duration, alignment interval, offset, and alignment timezone equal across the required Schedules. Buffers, hours, and capacity can vary.
Retiring a visit type	HealthcareService.active = false prevents finding, booking, holding, and confirming appointments for it. Existing appointments remain; cancellation is still allowed. Plan for outstanding holds.

Reference matching matters: `Schedule.serviceType` uses the service-reference extension within a CodeableConcept. Schedule parameter overrides separately identify their service with a `service` sub-extension. A matching code alone does not establish these links, and an override without a service pointer is ignored.

3.2 Actors and Schedule Modeling

Decide which resources consume time and which merely describe eligibility or location. Medplum requires exactly one actor on each Schedule. Consolidating a practitioner's availability on one Schedule is the recommended starting point; the server does not enforce one Schedule per actor or automatically synchronize separate calendars for the same person.

Questions:

- Which practitioners, rooms, and devices must be reserved?
- Is a Location a bookable resource, a site of service, or a jurisdiction?
- Are resource requirements simultaneous, or different phases of a procedure?

Situation	Approach
Clinician availability	Use a Practitioner actor. Use PractitionerRole for service/jurisdiction eligibility rather than splitting the person's calendar by license.
Individually bookable room or device	Give the Location or Device its own Schedule and include it in the appointment search and booking.
Facility or jurisdiction used for filtering	Model the association for candidate selection; do not assume a Location reference reserves time or checks capacity.

Situation	Approach
Record the appointment's site	Store a Location reference in `Appointment.supportingInformation` using `toAppointmentSiteReference` from `@medplum/core`; read it with `getAppointmentSite`. This identifies the site separately from a booked room and does not reserve capacity.
Same person at several sites	Keep the actor consistent and vary service-specific availability. Separate calendars can otherwise permit conflicting bookings.
Resources needed for different phases	Define a separate orchestration design. A concurrent multi-Schedule appointment uses a common booking interval, not independently timed resource phases.

3.3 Availability, Time Zones, and Blocks

Decide when booking is allowed and how exceptions are maintained. Use implicit availability, not pre-generated open Slots. Persist Slots for booked, held, or blocked time.

Questions:

- Are weekly hours shared, actor-specific, or service-specific?
- Which time zone defines local hours and which defines the start-time grid?
- What are the maximum booking horizon, minimum notice, and exception rules?
- Who enters closures, and what happens to appointments already booked?

Situation	Approach
Shared hours with actor overrides	HealthcareService availableTime supplies defaults. A Schedule availability override replaces those hours, rather than intersecting with them.
Prep or cleanup time	Set bufferBefore/bufferAfter; booking reserves exclusive buffer Slots. Include buffers when assessing usable appointment times.
Fixed appointment grid	Set alignment interval and offset deliberately. The default is 60 minutes with offset zero, not automatic back-to-back placement.
Local hours and daylight saving time	Resolve availability timezone from Schedule parameters, then service parameters, then the actor extension. Configure alignmentTimezone separately if the grid must follow local midnight; its default is Etc/UTC. Test relevant DST transitions.
One-off absence or holiday	Create busy-unavailable Slots on affected Schedules. Omit Slot serviceType to block all services; set it only for an intentionally service-specific block.
Facility closure	Include the facility as a required schedulable resource, or apply blocks to each affected Schedule. A facility block does not automatically propagate to practitioners.

Situation	Approach
Maximum booking window	Use `Schedule.planningHorizon`. A rolling horizon requires application maintenance.
Minimum lead time or daily/weekly caps	The scheduling server does not enforce these policies. Implement and validate them in the application's controlled booking path, not just search filters.

Defaults need explicit review: without configured availability, time defaults to always available. Without a resolvable availability timezone, scheduling fails. `availability.notAvailableTime` is not currently processed; use blocked Slots for exceptions. A new block does not itself resolve existing appointments, so define the notification and rescheduling workflow.

3.4 Access and Delegated Administration

Decide who may see schedules and patient details, book for someone else, change availability, and make exceptions. Use AccessPolicy and ProjectMembership for data access; use the Access Control Decision Guide for the broader authorization model.

Questions:

- Can patients discover availability without reading other patients' appointments?
- Which staff can book or cancel for which patients and organizations?
- Who can change shared defaults, provider hours, or blocked time?

Situation	Approach
Patient self-service	Grant only the resource access required by the supported operations and the patient's workflow. Test with patient credentials, including denied access cases.

Situation	Approach
Delegated staff booking	Scope access by role and organization. Preserve the initiating user and reason for delegated or exceptional actions in the audit design.
Rescheduling permission	In addition to booking access, `$reschedule` needs Appointment read/update and Slot read/delete access. The existing Slots and their Schedules must be readable. Test these permissions explicitly; permission to book does not imply permission to move an appointment.
Required business checks beyond access rules	Route booking through a controlled server-side workflow, and restrict alternate write paths that could bypass those checks.
Sensitive group or cross-team appointments	Review resource-level visibility before putting multiple patients or teams on a shared Appointment.

Candidate filtering and hidden UI controls are not authorization. Do not assume permissions on a Schedule automatically confer the intended permissions on its Slots or Appointments. A post-write Subscription Bot can trigger follow-up work, but cannot serve as a pre-commit rejection of the original write.

Finding an Appointment

3.5 Eligibility, Preferences, and First Available

Decide which actors are eligible before searching, then how to rank acceptable results. Scheduling operations do not infer licensure, enrollment,

patient preferences, or clinical suitability. See Matching Patients to Providers by State-by-State Licensure.

Questions:

- Is the request for a specific practitioner, a class of practitioners, or any eligible practitioner?
- Which requirements are hard constraints, and which preferences can be relaxed with patient agreement?
- Are language, practitioner gender, continuity, location, or day/time preferences relevant?
- Does "first available" mean earliest overall, earliest preferred, or earliest within a bounded search?
- How are equally suitable options selected, and what happens when some searches fail?

Situation	Approach
Licensure or service eligibility	Resolve allowed actors through the application's rules before `$find`, and validate eligibility again in the booking workflow. Model a practitioner's service role with its jurisdiction Locations; multiple jurisdictions may share one role.
Hard patient requirement	Remove incompatible candidates or times before presenting an offer. Do not silently relax it when no result exists.
Soft preference	Rank eligible results and make alternatives explicit. Keep preference ranking separate from access and licensure rules.
Named practitioner	Search that practitioner's Schedule plus any other required resources.

Situation	Approach
First available from a pool	Search eligible candidates or combinations over a defined horizon, merge results, and apply a stable tie-break rule. Expand the horizon explicitly when needed.
Partial results	Distinguish “no availability” from an incomplete or failed search. Do not claim globally earliest availability when only part of the candidate set was searched.

`$find` accepts explicit Schedules, not a patient-preference or ranking specification. Its search window is limited to 31 days per request; larger horizons require multiple requests. Result limits and candidate-combination limits must also be considered when defining first-available behavior.

3.6 Multi-Resource Combinations

Decide which resources are required together and where alternatives are allowed. Repeating `schedule` in `$find` requests the intersection of all those Schedules, not a choice among them.

Questions:

- Does the appointment need one practitioner or several?
- Are the room and device fixed, or can any eligible one be used?
- How large can the candidate pool become?

Situation	Approach
Named clinician and named room	Send both Schedules in one `$find` call. Book the resulting proposal with one `$book` or `$hold` operation.

Situation	Approach
Either clinician A or B, plus one room	Search each eligible clinician/room combination, then merge options. Passing both clinicians together would reserve both.
Two clinicians required	Include both Schedules in the same search and booking. Each must support the service and compatible common parameters.
Large pools	Narrow eligibility before enumerating combinations, bound the search, and define how partial results are displayed.

Use the server's multi-Schedule booking operation for the selected combination. Sequential independent bookings can leave only part of the required appointment reserved.

Booking and Managing Appointments

3.7 Capacity and Group Visits

Decide whether concurrency represents deliberate overbooking, interchangeable resource capacity, or patients attending a shared session. These are different modeling decisions. See Overbooking.

Questions:

- Must each chair or room be individually assigned, or are they interchangeable?
- Does each patient need independent cancellation, access, clinical documentation, and billing?
- Does a provider participate in every concurrent booking?

Situation	Approach
Exclusive resource	Keep the default `slotCapacity` of 1.

Situation	Approach
Deliberate overlapping appointments	Set slotCapacity on the service or Schedule and continue using $book/$hold. All required resources must have capacity.
Interchangeable stations	A shared Schedule with capacity can represent the pool. Use individual actors when assignment, equipment differences, or station-specific blocks matter.
Several patients in a session	Prefer patient-specific Appointments when privacy and per-patient lifecycle matter; define explicit session linkage and capacity. A single multi-patient Appointment requires a separate participant-count policy.

Capacity counts overlapping booking Slots, not the number of Patient participants. The strictest capacity among overlapping bookings applies, and existing Slots retain the capacity stamped when booked. Changing configuration does not rewrite those Slots. Buffers remain exclusive and can prevent otherwise permitted overlaps.

Costly modeling choice: a shared multi-patient Appointment couples access and lifecycle. FHIR R4 has no Appointment.partOf; define any session grouping deliberately rather than inventing that field.

3.8 Immediate Booking, Holds, and Approval

Decide whether to consume capacity immediately, reserve it temporarily, or keep an unreserved request. Approval and prior-authorization workflows are application responsibilities, not automatic effects of an Appointment status.

Questions:

- Does the time need protection while details or approvals are collected?

- Who approves, what evidence is required, and how long may a hold remain?
- Who releases abandoned or rejected requests?

Situation	Approach
Immediate booking	`$find` → `$book`. Creates a booked Appointment and its busy/buffer Slots atomically.
Reserve while awaiting approval	`$find` → `$hold` → `$confirm`. A hold creates a pending Appointment and busy-tentative Slots that consume capacity.
Rejected or expired hold	Use `$cancel` to release it. Define expiry tracking and cleanup in the application; `$hold` has no built-in TTL.
Request without reserving time	Track the request separately, such as in a Task. Search and book when ready; availability may have changed.
Participant acceptance required	Capture responses and implement the rule that permits confirmation. An AppointmentResponse does not automatically confirm a hold.

For a patient appointment, add the selected Patient to the proposal's participants before booking or holding; `$find` supplies the scheduled actors, not the patient. Do not call `$book` to confirm an existing hold. Do not use `tentative` as an Appointment status; it is a participant status. Make confirmation and expiry processing coordinate so an approval cannot race an independent cleanup workflow unnoticed.

3.9 Conflicts, Refresh, and Controlled Overrides

Decide how the user experiences a changed result and who can bypass ordinary scheduling rules. `$find` is a snapshot, not a reservation. `$book`, `$hold`, and `$reschedule` recheck availability transactionally, including capacity and buffers.

Questions:

- Is refresh-on-action sufficient, or does the workflow need live updates?
- What happens when another user books the last available capacity?
- Which exceptions are allowed, who authorizes them, and how are they audited?

Situation	Approach
Ordinary booking conflict	Refresh search results and ask the user to select another option. Do not present an unsuccessful booking as confirmed.
Live calendar experience	Use bounded polling or appropriately scoped Subscriptions. Refresh affected results; live updates do not replace transactional booking validation.
Ambiguous response after a network failure	Reconcile whether the appointment was created before retrying. Avoid blind retries that can create duplicate bookings.
Routine capacity greater than one	Configure capacity rather than implementing an override path.
Privileged exception	Define a controlled write path, required reason, access restrictions, consistency checks, and downstream notification. Direct FHIR writes do not inherit the scheduling operation's validation.

The React workspace's `canBypassSchedulingRules` option exposes controls; it does not enforce permission. A transaction can keep custom Appointment/Slot writes together, but does not by itself reproduce `$book`'s conflict checks. Ensure

transaction support is enabled for cross-referencing writes; do not rely on a batch fallback for atomicity.

3.10 Cancellation, Rescheduling, and Clinical Lifecycle

Decide which changes are allowed after booking, how reasons are recorded, and when an appointment becomes a clinical encounter.

Questions:

- Who can cancel or move an appointment, and with what notice?
- Must rescheduling preserve Appointment identity and downstream references?
- Which arrival, no-show, and completion states are needed?
- When is the Encounter created, and who owns that automation?

Situation	Approach
Cancel a booked or pending appointment	Use `$cancel`, optionally with a structured `cancelationReason`. It preserves the cancelled Appointment and deletes its linked Slots, including buffers, atomically.
Find a replacement time or resource	Pass `ignore-appointment=Appointment/{id}` to `$find` to exclude this appointment's Slots, including buffers, from availability calculations. Other appointments still block. Commit the move with `$reschedule`; `$book` and `$hold` do not ignore the original Slots.
Move time or resources	Use `$reschedule` for a booked or pending Appointment. Supply the new `start` and the complete set of destination Schedules. It preserves Appointment identity and status, reconciles scheduled participants, and replaces Slots atomically. If the new time is unavailable, the original booking remains intact.

Situation	Approach
Change visit type	$reschedule retains the original service type and requires exactly one HealthcareService reference on the stored Appointment. Changing visit type requires cancel-and-book, with explicit failure/recovery handling and a new Appointment identity.
Link a replacement appointment	Use an explicit application linkage or Provenance model. R4 Appointment.basedOn references ServiceRequest, not another Appointment.
Arrival and encounter creation	Define application transitions and create the Encounter at the chosen event, linking through Encounter.appointment. Make automation safe to retry.
Participant accept/decline	Use Appointment.participant.status and, where needed, AppointmentResponse. Keep response handling distinct from the overall Appointment status.
No-show or completion	Define the operational rule and accountable actor. $book does not automatically manage later clinical states.

Confirm that your server version supports $reschedule before choosing this workflow. It derives duration and buffers from the scheduling configuration; omitted Schedules are removed from the appointment. Patients and other non-scheduled participants are preserved. Slot IDs change, so integrations must handle Slot deletion and creation. Site metadata in supportingInformation stays unchanged; a move to another site needs an explicit metadata update in the application's workflow.

Do not equate an SMS acknowledgment with confirmation of a held booking, or mutate only Appointment.start/end to reschedule while leaving its Slots behind. Notify dependent systems whenever the time, actors, or status changes.

Conditional Workflows

3.11 Recurring Appointment Series

Decide how recurrence is grouped and what happens when only some occurrences can be booked. Current scheduling operations handle individual appointments; recurring weekly availability is not a recurring appointment series.

Questions:

- Is the series finite, or maintained over a rolling horizon?
- Must every occurrence be available before the series is accepted?
- Can a user change one occurrence, all future occurrences, or the entire series?
- Does recurrence follow local wall-clock time across DST changes?

Situation	Approach
Series driven by an actual service order	Each occurrence may reference its ServiceRequest through Appointment.basedOn. Define separate series identity if one order can generate several series.
Series without an appropriate order	Define a shared series identifier or documented extension. Do not invent a clinical order solely to obtain a grouping field.
Booking several occurrences	Check each occurrence and handle partial success explicitly. Multiple $book calls are not one all-or-nothing series transaction.

Situation	Approach
Series-wide changes	Track membership and exceptions, then orchestrate per-occurrence changes with recovery and notification.

FHIR R4 has no native Appointment recurrence template. Do not use `Appointment.partOf` or a CarePlan reference in `Appointment.basedOn`. Recheck the scheduling API's supported series behavior before implementing a custom model, since this area is evolving.

3.12 Waitlists

Decide whether a vacancy triggers an offer, a hold, or an automatic booking. Waitlist ranking and promotion require application orchestration.

Questions:

- What service, candidates, time range, and preferences does the patient accept?
- How are priority and fairness determined?
- How long is an offer valid, and does it reserve capacity?

Situation	Approach
Track unmet demand	Use a workflow record such as Task containing the request criteria and patient reference.
Notify when time opens	Re-run eligibility and availability checks, then send an offer. Notification alone does not reserve time.
Offer with a reservation	Use `$hold` and an application-managed expiry policy. Coordinate competing offers and cleanup.

Automatic promotion	Confirm the patient's consent and criteria, revalidate eligibility, and use $book; handle a competing booking as a normal conflict.

3.13 Reminders and Responses

Decide the channels, timing, and meaning of responses. Reminder acknowledgment, participant acceptance, and booking confirmation are separate events.

Questions:

- Which channels and timing rules apply, and in whose time zone?
- Can replies acknowledge, cancel, or request a change?
- How are reminders suppressed after cancellation or rescheduling?

Situation	Approach
Scheduled outbound reminder	Use application scheduling/Bots and a channel provider. Track delivery attempts and deduplicate by appointment, reminder type, and relevant appointment version/time.
Two-way response	Authenticate or otherwise validate the response context and route it to the appropriate workflow or operation.
Appointment changes	Invalidate obsolete reminders and schedule replacements. Recheck current state before sending.

Use the Messaging & Communications Decision Guide for channel, consent, and delivery design.

3.14 Telehealth Provisioning

Decide when to create a virtual meeting and how participants receive access. Video provisioning is an integration, not part of $book.

Questions:

- Is the visit virtual, in-person, or selectable per booking?
- Is the meeting created at booking, approval, or shortly before the visit?
- Who can read the join details, and what happens when the visit changes?

Situation	Approach
Provision at booking or confirmation	Trigger an idempotent integration after the chosen state is committed. Keep failures visible without implying the booking failed.
Provision just before the visit	Recheck the Appointment state before creating the meeting and sending access details.
Store join details	Use a documented R4-compatible extension or linked resource, with appropriate access restrictions.
Cancel or move the appointment	Reconcile the external meeting and invalidate obsolete links or invitations as required.

3.15 External Calendar Integration

Decide whether the integration exports appointments, imports busy time, or permits another system to change bookings. Establish one write authority per kind of data.

Questions:

- Is the destination a personal calendar, patient calendar feed, or another clinical scheduler?
- Do external events block availability, and do edits move Medplum appointments?
- How are cancellations, recurrence exceptions, outages, and reconciliation handled?

Situation	Approach
Outbound appointment mirror	Publish selected appointment details and retain external identifiers. Minimize patient information exposed to personal calendars.
External busy-time import	Maintain `busy-unavailable` Slots for relevant events in the booking horizon. Handle updates and deletion as well as creation, with stable external identifiers and conditional writes.
Bidirectional appointment editing	Specify conflict ownership, echo suppression, and reconciliation. Route accepted changes through the controlled scheduling workflow.
Patient calendar feed	Provide a patient-scoped, revocable access mechanism and a clear policy for sensitive details.

An external calendar event does not automatically create a Medplum conflict. Imports and reconciliation must maintain the blocking Slots. Use the Data Migration Decision Guide when replacing or running alongside another scheduler.

Reference Implementation

For a reference implementation of a complex scheduling interface, see `@medplum/react-scheduling`, including SchedulingWorkspace, appointment search/booking components, MultiCalendar, and ScheduleAvailabilityEditor. The package is currently Alpha and subject to breaking changes; host applications supply business rules, authorization, and integrations. Explore the components in Storybook.

Before You Implement

- Record the chosen service, actor, availability, and access models, with an owner for each.
- Separate server-supported behavior from custom workflows and unresolved product requirements.
- Test booking conflicts, capacity and buffers, time zones, expiry/cancellation races, and denied access using representative user roles.
- Define recovery for partial series bookings, failed integrations, and ambiguous responses.
- Confirm the server/package versions in use and review the scheduling beta changelog before launch or upgrade.
- Assign an owner and acceptance criteria to every Not sure answer and every custom scheduling rule.