Perfect ✅
Here’s a **two-person roadmap** for **ParkSpot TLV**, covering ~18 days.
I’ll split responsibilities clearly between **Person A (Backend/Infrastructure)** and **Person B (Frontend/Mobile)**, while pointing out **dependencies** (where one must wait for the other vs. where tasks can run in parallel).

🧑‍💻 Roles

- **Person A – Backend & Infrastructure**
  - ASP.NET Core API
  - Database design & migrations (Postgres/PostGIS + EF Core)
  - Rules engine
  - Notifications (server scheduling + FCM)
  - Logging & tracing
  - CI/CD & cloud deployment
- **Person B – Mobile App (Frontend)**
  - .NET MAUI app
  - Authentication & vehicle management screens
  - Map UI (Google Maps SDK + overlays)
  - Local storage (SQLite)
  - Push notification handling (client side)
  - UX flows (Park here, alerts, settings)

📅 18-Day Two-Person Plan

Day	Person A (Backend)	Person B (Frontend)	Parallel or Dependent?

1	Setup backend solution, init API project, add `/health` endpoint.	Setup MAUI project, repo structure, placeholder navigation.	Parallel
2	Add Serilog logging, correlation IDs, global error handler.	Build login/signup UI skeleton (no backend yet).	Parallel
3	Setup Postgres + PostGIS; write EF Core migrations for `users`, `vehicles`, `street_segments`, `zones`.	Integrate SQLite; prepare local models (`vehicles`, `sessions`).	Parallel
4	Implement JWT auth (`/auth/register`, `/auth/login`).	Wire login screen to backend auth (once available).	Dependent (B waits for A)
5	Vehicle endpoints (`/vehicles` CRUD).	Vehicle management UI (list, add, delete).	Can overlap once API exists
6	Parking rules schema & seed demo rules.	Build basic map screen (Google Maps SDK integration).	Parallel
7	Rule evaluation engine (given time + vehicle + segment → status).	Overlay colored polylines (mock data until API ready).	Parallel
8	Map API `/map/segments?bbox=…&time=…&vehicleId=…`.	Fetch map overlays from backend instead of mock.	Dependent (B consumes A)
9	Segment details API (`/segments/{id}/rules`).	Bottom sheet with rule details on tap.	Dependent
10	Parking sessions API (`/sessions` start/stop).	“Park here” button → POST session → show countdown.	Dependent
11	Add Hangfire for scheduling jobs; FCM push integration.	Integrate Firebase SDK in MAUI; register push tokens.	Parallel
12	Notification scheduler: schedule push at `(expires_at - lead_time)`.	Handle incoming push → show system notification.	Dependent
13	Add local fallback notification logic (server signals app with expiry).	Implement local timer backup for offline cases.	Parallel

14	Advanced logging: traceparent headers, Seq/Elastic sink.	Implement settings screen (alert lead time, vehicle default).	Parallel
15	Optimize PostGIS queries (indexes, ST_Simplify, bbox filters).	Optimize map rendering (simplified polylines, caching).	Parallel
16	Security hardening: Argon2id password hashing, refresh tokens.	UX polish: error messages, loading indicators, smooth navigation.	Parallel
17	Integration tests: rule evaluation, sessions, notifications.	UI tests: login, add vehicle, map view, session start.	Parallel
18	Deploy API to Azure App Service + Postgres. CI/CD pipeline.	Package MAUI app for Android (pilot build).	Parallel

---

🔑 Dependencies Explained

- **Auth & Vehicles** → Person B waits for Person A (Day 4–5).
- **Map API** → Person A must expose `/map/segments` before Person B can color overlays (Day 8).
- **Notifications** → Server scheduling must exist (A, Day 11–12) before B can show push alerts.
- Everything else (logging, SQLite, UI skeleton, maps, settings, polish) can be done **in parallel**.

⚡ Efficiency Notes

- The **critical path** is:
  Backend rule engine → Map API → Parking sessions → Notifications.

- Person B can **mock data** early (colored polylines, fake rule details) while Person A builds APIs. This avoids idle time.
- By **Day 18**, you have:
    - Fully working API in Azure.
    - Mobile app with login, vehicles, map overlays, parking sessions, and push alerts.

Here's the two-person plan with clear ownership and "Definition of Done" (DoD) for every milestone. It's 18 days (≈3 weeks). Everything is C#/.NET: ASP.NET Core API, MAUI app, EF Core, PostgreSQL/PostGIS (prod), SQLite (device), Serilog, Hangfire, Firebase Cloud Messaging (push only).

Roles (fixed for the whole project)

- Person A — Backend & Infrastructure: ASP.NET Core API, EF Core (PostgreSQL/PostGIS), rules engine, notifications scheduler, logging/tracing, CI/CD, cloud.
- Person B — Mobile App (Frontend): .NET MAUI, Google Maps overlays, auth UI, vehicles UI, local SQLite cache, push handling, UX flows.

Week 1 (Days 1–6): Foundations, Auth, Vehicles, Map Shell

Day	Person A (Backend)	DoD (what must work)	Person B (Mobile)	DoD (what must work)	Parallel/ Dependency
1	Create ASP.NET Core solution, Minimal API scaffold, `/health` liveness. Add Shared/ DTO project.	`GET /health` returns 200; solution builds in CI.	Create MAUI app skeleton, navigation shell (Login, Map, Vehicles, Settings).	App builds/runs on emulator; pages route correctly.	Parallel

2	Serilog JSON logging (console + rolling files), global exception handler, correlation IDs (W3C traceparent echo).	Logs include timestamp, level, traceId; 5xx returns RFC7807 problem+json.	Design auth and vehicles UI (mock). Wire form validation.	UI interactions validated; no backend calls yet.	Parallel
3	Provision PostgreSQL + PostGIS. EF Core setup. Migrations for `users`, `vehicles`, `zones`, `street_segments`.	`dotnet ef database update` on clean DB succeeds; GIST index exists on geometries.	Add SQLite via EF Core on device; local tables for `vehicles`, `cached_segments`, `sessions_local`.	Local DB created on first run; CRUD works for a dummy record.	Parallel
4	Auth: Argon2id hashing, `POST /auth/register`, `POST /auth/login` (JWT + refresh). Swagger/OpenAPI documented.	Register/login round-trip with test user; invalid creds yield 401; tokens expire per config.	Wire login/signup UI to backend; persist JWT securely; attach bearer token to future calls.	Successful login creates session; relaunch keeps user signed in.	B depends on A's endpoints
5	Vehicles: `GET/POST/PATCH/DELETE /vehicles` scoped to user.	CRUD passes integration tests; 403 on cross-user access; OpenAPI updated.	Vehicles screen: list + add/edit/delete + choose default vehicle.	Full UI↔API round-trip; optimistic UI updates; error banners on failures.	B consumes A
6	Seed `zones` and pilot `street_segments` (few neighborhoods). Index tuning.	Seed script loads without errors; bbox query returns segments in <150 ms (dev DB).	Integrate Google Maps in MAUI; show user live location; draw mock colored polylines.	Map centers on user; mock overlays render smoothly; location permission flow OK.	Parallel

Week 2 (Days 7–12): Rules, Map API, Sessions, Push Notifications

Day	Person A (Backend)	DoD	Person B (Mobile)	DoD	Parallel/Dependency
7	Rules model: `parking_rules` table (JSONB windows + conditions). Rules evaluator service (time + vehicle + segment → status + next_change_at). Unit tests for common cases.	20+ rule tests pass (weekday windows, resident/disabled exceptions). Deterministic outputs for fixed clock.	Map UI time selector (Now / pick future). Prepare to pass `time` and `vehicleId` to API.	UI can choose time and active vehicle; state persists while navigating.	Parallel
8	Map API: `GET /map/segments?bbox=…&time=…&vehicleId=…` → returns simplified polylines + `status` + `color` + `nextChangeAt`. Uses ST_Intersects, ST_Simplify, bbox paging.	With seed data, API returns ≤500 segments page in <200 ms; OpenAPI examples included.	Swap mock overlays for real API. Color segments per response. Handle paging while panning/zooming.	Smooth pan/zoom; throttled fetch; colors match server output in sample checks.	B consumes A
9	Segment details: `GET /segments/{id}/rules?time=…&vehicleId=…` → reasons, human text, next change.	Example: "Resident Zone 5 free until 18:00; non-residents paid until 17:00." Verified for known streets.	Bottom sheet on tap: rule summary, reason, until/next change.	Details load <500 ms; back/close behaves; accessibility labels set.	B consumes A

10	Sessions API: `POST /sessions` (vehicleId, segmentId, plannedMinutes, leadMinutes), `GET /sessions/active`, `PATCH /sessions/{id}` end.	Session lifecycle covered by integration tests; prevents 2 active sessions per user unless designed otherwise.	“Park here” flow: confirm street+vehicle, create session, show countdown chip.	End-to-end call creates session; countdown ticks; cancel ends session.	B consumes A
11	Hangfire (or Quartz.NET) for scheduling. FCM server integration. Store device tokens. Compute `fire_at = expires_at - leadMinutes`.	Background job enqueued on session create; dry-run logs include token and payload shape.	Add Firebase SDK to MAUI; request notification permission; register and send device token to backend.	Device token stored server-side; test push arrives on device in foreground/background.	Parallel, then integrate
12	Notification dispatch: only send if session still active; idempotent; retries with backoff; structured audit log.	Manual time-shift test fires push at expected minute (±60 s).	Handle push on device: show system banner; deep-link to session/map; respect user settings.	Push displays correctly; tapping opens app to session; background handling verified.	A triggers B’s validation

Week 3 (Days 13–18): Fallbacks, Settings, Performance, Hardening, QA, Deploy

Day	Person A (Backend)	DoD	Person B (Mobile)	DoD	Parallel/Dependency
13	Add "fallback reminder" in session response (timestamp for local alarm). Expose `/push/register` and `/push/test` for QA.	Fields present; contract documented; test endpoint returns 200 and triggers a harmless local notification.	Local fallback notification scheduled on device at server-suggested time (if user enabled).	If offline or push blocked, local alarm still fires within ±60 s.	Parallel
14	Observability: enrich logs (traceId, userId, sessionId), optional sink to Seq. Add `/ready` readiness (DB + Hangfire + FCM).	`/health` and `/ready` green in staging; logs searchable by traceId.	Settings screen: lead minutes, yellow threshold, mute toggles. Persist to device and server.	Settings update round-trip; yellow logic applied on map.	Parallel
15	PostGIS perf: ST_SnapToGrid/Simplify tolerance by zoom, proper GIST indexes, bbox + tile caching strategy.	P95 for `/map/segments` ≤250 ms with 1000 segments in bbox on dev data.	Map perf: request coalescing, minimal redraws, polyline simplification on client.	No jank on mid-range Android; memory stable over 10 min pan/zoom test.	Parallel
16	Security hardening: password policy, refresh tokens, revoke on logout, rate limiting on auth, CORS, minimal scopes for cloud.	Security tests pass; OWASP-style checklist items closed.	UX polish: empty states, failure toasts, retry, loading skeletons, safe area spacing.	Acceptance walkthrough free of rough edges; no blocking UI glitches.	Parallel

17	Test suite: unit tests (rules), integration (sessions, notifications), e2e (happy path). Seed data and Postman collection published.	>80% coverage for rules + controllers; e2e passes on CI.	UI tests: login → add vehicle → see colored map → park here → receive push. Crash handling and logging verified.	Automated smoke passes on CI; manual checklist complete.	Parallel
18	Deploy: Azure App Service (API), Azure Database for PostgreSQL (with PostGIS), Azure Storage for logs/artifacts. GitHub Actions CI/CD (build, test, deploy).	Blue/green or staging slot swap; `/health` green; connection strings via secrets.	Package Android APK (pilot). Versioned build; release notes; share with testers.	Installable on test devices; crash-free session count target reached (e.g., 20 sessions).	Parallel

Ownership, hand-offs, and contracts

- API contracts freeze points
    - End of Day 8: `/map/segments` response shape (segment id, name, encoded/simplified polyline or coordinate list, status, color, nextChangeAt).
    - End of Day 9: `/segments/{id}/rules` detail schema (reason strings + structured fields).
    - End of Day 10: `/sessions` create/active/end payloads.
    - End of Day 11: `/push/register` body and push payload schema.
- Shared artifacts
    - Shared/ project for DTOs used by both API and MAUI to avoid drift.
    - OpenAPI JSON committed; Postman collection exported.
    - Seed data set (zones, a few hundred segments, representative rules) for deterministic tests.

- Branching and CI
    - `backend/*` and `mobile/*` branches; protected `main`.
    - PR requires: compile, unit/integration tests, lint/format, OpenAPI diff (breaking changes flagged).

Acceptance criteria summary (what "done" means per area)

- Auth & vehicles
    - Register/login with JWT works; tokens refresh and revoke; vehicles CRUD scoped to user; 401/403 paths tested.
- Map overlays
    - Given a bbox/time/vehicle, server returns colored segments; client renders with smooth pan/zoom; paging and throttling in place.
- Rules engine
    - Deterministic status for a fixed time; unit tests for weekday windows, mixed permits, and "next change at" boundaries; holiday/exception hook prepared.
- Sessions & notifications
    - Create session sets expiry; scheduler enqueues job; push delivered within ±60 s of configured lead; cancel prevents push; local fallback works offline.
- Observability
    - Every request has traceId; errors are structured problem+json; `/health` and `/ready` cover DB + scheduler + FCM; logs searchable by user/session.
- Performance
    - `/map/segments` P95 ≤250 ms with realistic bbox; mobile memory stable; no noticeable UI stutters on mid-range device.
- Security
    - Argon2id for passwords; rate limits on auth; CORS restricted; secrets in Azure Key Vault or App Service settings; minimal DB privileges.

Who can work simultaneously vs. where one waits

- Simultaneous
    - Days 1–3 (scaffolding, logging, DB/migrations, MAUI shell, SQLite).
    - Day 6 (seed streets; map SDK with mock overlays).
    - Day 11 (server push infra vs. device push registration).
    - Days 13–18 (observability, settings, perf, hardening, testing, deploy).
- Dependencies
    - Day 4–5: Mobile auth/vehicles waits for backend endpoints.
    - Day 8–10: Mobile overlays, details, and sessions wait for map and session APIs.
    - Day 12: Validating production-like push requires backend scheduler working.

If you want, I can turn this into two printable checklists (one per person), each with daily boxes to tick, plus a minimal OpenAPI stub for `/map/segments`, `/segments/{id}/rules`, and `/sessions` so you can start wiring right away.

Here's a very detailed, day-by-day workflow for the two-person plan.

Everything is C#/.NET: ASP.NET Core API, EF Core (PostgreSQL + PostGIS), .NET MAUI, SQLite, Serilog, Hangfire, FCM. For each day you'll see: goals, what Person A (backend) and Person B (mobile) implement, tests to run that day, acceptance checks, artifacts to commit, and common pitfalls.

Day 1 — Project scaffolding & repos

- Goals
    - Working solution skeletons for API and MAUI.
    - CI builds succeed; /health responds.

- Person A (Backend)
  - Create solution ParkSpot.sln with projects: BackEnd.Api, BackEnd.Domain, BackEnd.Infrastructure, Shared.Contracts, Tests.BackEnd.
  - Add minimal Program.cs with /health and /version.
  - Add Directory.Packages.props to pin package versions.
  - Set nullable enable, warnings as errors for BackEnd.*.
  - Add GitHub Actions workflow: dotnet build + test on push.
- Person B (Mobile)
  - Create MobileApp (MAUI) with pages: LoginPage, MapPage, VehiclesPage, SettingsPage (Shell navigation).
  - Add Dependency Injection via MauiProgram.cs; create HttpClient service placeholder.
  - Enable Android emulator profile and device permissions placeholders (Location, Notifications).
- Tests today
  - Backend: super-simple xUnit test asserts /health 200 via WebApplicationFactory.
  - Mobile: launch app on emulator, navigate across pages without crashes.
- Acceptance checks
  - CI green on both solutions.
  - /health returns 200 locally.
  - App opens and routes between all pages.
- Artifacts
  - README with run instructions.
  - ADR-0001 Stack choices.
- Pitfalls
  - Forgetting to set LangVersion and TreatWarningsAsErrors.
  - MAUI Android SDK mismatch; verify with dotnet workload list.

Day 2 — Logging, error handling, tracing scaffold

- Goals
    - Structured logs and consistent error responses; W3C trace context emitted.
- Person A
    - Add Serilog (console JSON + rolling files to ./logs/api/).
    - Global exception handler producing application/problem+json.
    - Echo traceparent/tracestate headers; create middleware to attach traceId, userId(if present) to log scopes.
    - Add /ready placeholder.
- Person B
    - Add basic ILogger usage in ViewModels; central ErrorService to show toasts/snackbars.
    - Implement RetryPolicy (Polly) for HttpClient with jitter backoff.
- Tests today
    - Backend: unit tests for error middleware (400, 401, 403, 500).
    - Verify each request log has timestamp, level, traceId, path, method, status, durationMs.
    - Mobile: simulate HTTP failure; confirm toast and retry.
- Acceptance checks
    - 5xx returns problem+json with traceId.
    - Logs are single-line JSON and include correlation fields.
- Artifacts
    - Logging policy doc: fields, redaction rules (Authorization/Cookie).
- Pitfalls
    - Logging sensitive headers; add redaction.

Day 3 — Databases wired: Postgres + PostGIS + EF Core; SQLite on device

- Goals
  - EF Core connected to Postgres (dev); migrations created and applied.
  - SQLite initialized on device with local schema.
- Person A
  - Provision Postgres; enable PostGIS extension.
  - Create EF Core DbContext + entities: User, Vehicle, Zone, StreetSegment.
  - Migrations: create tables, GIST indexes on geometries; seed minimal zones + 50 sample segments.
  - Add Testcontainer setup for integration tests.
- Person B
  - Add EF Core SQLite DbContextMobile with VehiclesLocal, CachedSegments, SessionsLocal.
  - Lazy init migrations on first run; simple Repository for local CRUD.
- Tests today
  - Backend integration test spins up Postgres container, runs migrations, verifies PostGIS extension exists.
  - Mobile unit test writes/reads a VehicleLocal record.
- Acceptance checks
  - dotnet ef database update succeeds on clean DB.
  - Mobile creates sqlite db file and persists local data between launches.
- Artifacts
  - db/Seed/segments.geojson (pilot area).
  - Migration scripts in Database/Migrations.
- Pitfalls
  - Missing SRID: store geometries with a consistent SRID (e.g., 4326) and document it.

Day 4 — Authentication (JWT + refresh) and secure client session

- Goals
  - Register/login is functional; client stores tokens securely.
- Person A
  - Implement Argon2id password hashing, password policy.
  - Endpoints: POST /auth/register, POST /auth/login, POST /auth/refresh, POST /auth/logout.
  - Issue short-lived access token + refresh token; revoke on logout/rotation.
  - Swagger security scheme; authorize attribute boilerplate.
- Person B
  - Login/Signup forms wired to backend; secure token storage using SecureStorage.
  - HTTP pipeline adds Authorization header automatically.
  - Persist auth state; logout clears tokens.
- Tests today
  - Backend: auth integration tests (register/login/refresh/revoke); invalid creds → 401; token expiry simulation.
  - Mobile: happy path login, relaunch persists session; bad creds shows inline validation.
- Acceptance checks
  - Tokens rotate; refresh invalidated after use; blacklist enforced.
- Artifacts
  - OpenAPI auth docs; Postman collection auth flows.
- Pitfalls
  - Returning too much info on auth errors; keep generic.

Day 5 — Vehicles domain end-to-end

- Goals
  - Vehicles CRUD (server + client) scoped to user.
- Person A
  - Endpoints: GET/POST/PATCH/DELETE /vehicles; fields: plate, type, residentZoneCode, disabledPermit.
  - Add OwnerId checks in handlers; optimistic concurrency with rowversion.
- Person B
  - VehiclesPage: list, add, edit, delete; set default vehicle; validation (plate format).
  - Local cache sync of vehicles for offline view.
- Tests today
  - Backend: authz tests deny cross-user access; validation tests.
  - Mobile: UI tests add/edit/delete; offline view falls back to cache.
- Acceptance checks
  - Full round-trip CRUD; errors surfaced meaningfully to user.
- Artifacts
  - DTOs placed in Shared.Contracts to avoid drift.
- Pitfalls
  - Not normalizing plates; decide uppercase normalization.

Day 6 — Map shell & location services (client) + seed more segments (server)

- Goals
  - Map displays user location; mock overlays ready; more seed data loaded.
- Person A
  - Expand seed for StreetSegments (1–2 full neighborhoods).
  - Verify GIST index and bbox query performance.

- Person B
  - Integrate Google Maps in MAUI; request location permissions; show blue dot.
  - Implement viewport change events with throttled callback and bbox calculation.
  - Render mock polylines with colors (temp).
- Tests today
  - Backend: bbox query returns segments under threshold time.
  - Mobile: permissions flows (grant/deny); map pans/zooms smoothly; throttling verified via logs.
- Acceptance checks
  - Usable map screen; user position accurate; mock overlays visible.
- Artifacts
  - MapService interface with GetSegmentsAsync(bbox, time, vehicleId).
- Pitfalls
  - Excessive fetches on pan; ensure debounce + distinct until changed.

Day 7 — Parking rules model + evaluator core

- Goals
  - Server can evaluate legality for a segment at a given time and vehicle context.
- Person A
  - Table parking_rules: baseType(FORBIDDEN/FREE/PAID), timeWindows(jsonb), conditions(jsonb: residentOnly, disabledAllowed, vehicleTypes).
  - Implement RulesEvaluator with NodaTime; resolve nextChangeAt; cover boundary cases.
- Person B
  - Time selector UI (Now / choose date/time); vehicle selector quick switch; state stored centrally.

    - Wire MapService to pass time + vehicleId (still using mock server response for now).
- Tests today
    - Backend: 25+ unit tests across typical Tel-Aviv windows (e.g., residents free nights, Saturday exceptions).
    - Mobile: UI state persists across tabs; validation prevents past date.
- Acceptance checks
    - Deterministic evaluator outputs for fixed test clock.
- Artifacts
    - Rules JSON schema examples committed.
- Pitfalls
    - DST changes; test on days clocks shift.

Day 8 — Map API for segments with legality

- Goals
    - Real server data powers map overlays.
- Person A
    - Endpoint GET /map/segments?bbox=…&time=…&vehicleId=… returns: segmentId, name, simplified polyline (encoded or list), status, color, nextChangeAt.
    - Use ST_Intersects + ST_Simplify(zoom-aware tolerance); page results.
- Person B
    - Replace mock with real API; implement paging over viewport; draw per color.
    - Display yellow when $0 < \text{nextChangeAt} - \text{time} \le \text{userThreshold}$.
- Tests today
    - Backend: contract tests on response shape; performance P95 ≤250 ms for typical bbox.
    - Mobile: visual spot checks; colors match known ground truth in pilot area.

- Acceptance checks
    - Smooth pan/zoom with updates; no UI freezes; correct colors.
- Artifacts
    - OpenAPI with concrete examples; sample bbox curl.
- Pitfalls
    - Oversized payloads; check gzip enabled and geometry simplification.

Day 9 — Segment details API + bottom sheet

- Goals
    - Tap a street → see human-readable rules and structured details.
- Person A
    - Endpoint GET /segments/{id}/rules?time=…&vehicleId=… returns status, reasons[], windows[], nextChangeAt, paymentInfo(if PAID).
- Person B
    - Bottom sheet UI: rule summary, reason bullets, “Park here” CTA, optional link “Pay (Pango)” placeholder.
    - Graceful loading state; retry on failure.
- Tests today
    - Backend: reason text correctness; handles unknown segments with 404 problem+json.
    - Mobile: UI test opens sheet, scrolls, dismisses; state restored on rotate.
- Acceptance checks
    - Details appear <500 ms in pilot data; status matches overlay color.
- Artifacts
    - Reason text templates documented.
- Pitfalls
    - Duplicated work between evaluator and detail endpoint; share service logic.

Day 10 — Parking sessions lifecycle

- Goals
  - Start/stop an active parking session server-side; client shows countdown.
- Person A
  - Endpoints: POST /sessions (vehicleId, segmentId, plannedMinutes?, leadMinutes?), GET /sessions/active, PATCH /sessions/{id} {status=ended}.
  - Validate no multiple active sessions unless deliberately allowed; compute expiresAt from rules or planned duration.
- Person B
  - “Park here” flow: sheet → confirm dialog → create session → show countdown chip on map header; “End session” action.
  - Persist session locally; recover countdown after app relaunch.
- Tests today
  - Backend: lifecycle integration tests; race conditions (double start/end) handled idempotently.
  - Mobile: cancel confirms; countdown ticks; survives minimize/restore.
- Acceptance checks
  - Full round-trip works; session visible from GET /sessions/active; UI reflects ended state promptly.
- Artifacts
  - Session state machine diagram.
- Pitfalls
  - Time zone confusion; always use UTC server-side, convert in UI.

Day 11 — Push infrastructure: Hangfire + FCM

- Goals
  - Backend schedules push; device can receive a test push.
- Person A
  - Add Hangfire with persistent storage; dashboard secured.
  - /push/register (stores deviceToken per user); /push/test sends dummy message.
- Person B
  - Integrate Firebase Messaging in MAUI; request permission; post deviceToken to server; handle foreground/background notification display.
- Tests today
  - Backend: enqueue test sends to token stub; log payloads.
  - Mobile: receive test push when app foreground/background; tapping opens MapPage.
- Acceptance checks
  - Test push arrives within a few seconds; deep-link works.
- Artifacts
  - Push payload schema documented (title, body, sessionId?, deeplink).
- Pitfalls
  - Not persisting platform info (Android/iOS); store platform with token.

Day 12 — Scheduled notifications for sessions

- Goals
  - Real alert at expiresAt − leadMinutes, only if session still active.
- Person A
  - On session create/update, enqueue job for (expiresAt − leadMinutes); job checks active status; retries on transient FCM errors.
  - Audit table for sent pushes with traceId and outcome.
- Person B
  - Notification handler routes to active session or map; respects user muted settings.
  - In-app banner if app is open; system notification otherwise.
- Tests today
  - Backend: time-shifted unit/integration test validates schedule accuracy ±60 s; canceling session prevents push.
  - Mobile: receive scheduled push; tapping brings user to correct screen.
- Acceptance checks
  - At least one full "create → scheduled push → receive → open → end session" works.
- Artifacts
  - Notification lead time default and override behavior spec.
- Pitfalls
  - Duplicate scheduling on session edits; ensure you cancel/reschedule previous job.

Day 13 — Local fallback reminders

- Goals
    - If push fails or device offline, a local reminder still fires.
- Person A
    - Include fallbackReminderAt in session create response; optional “remindAtUtc” endpoint if schedule changes.
- Person B
    - Schedule OS-level local notification for fallbackReminderAt; cancel when session ends or push received.
- Tests today
    - Mobile: airplane mode scenario still produces local alert at correct time.
- Acceptance checks
    - Either push or local alert fires reliably; double alerts avoided.
- Artifacts
    - Fallback decision tree documented.
- Pitfalls
    - Device doze modes; use exact alarms only when necessary and document battery impact.

Day 14 — Settings & “yellow” threshold; observability polish

- Goals
    - User can set leadMinutes, yellowThreshold; logs enriched; /ready checks all dependencies.
- Person A
    - /ready probes DB, Hangfire, FCM connectivity; add Seq sink (optional).
    - Log enrichers add userId and sessionId consistently.
- Person B

    - SettingsPage: lead minutes, yellow threshold, notification toggles; apply to rendering (yellow if nextChangeAt within threshold).
- Tests today
    - Backend: /ready returns green only when all dependencies ok.
    - Mobile: adjusting yellow threshold changes colors without app restart.
- Acceptance checks
    - Logs filterable by userId and sessionId.
- Artifacts
    - Runbook for /ready failures; KQL/Seq queries examples.
- Pitfalls
    - Forgetting to persist settings to server if needed for scheduling defaults.

Day 15 — Performance tuning (server and client)

- Goals
    - Meet latency and smoothness targets.
- Person A
    - Index review; add ST_SnapToGrid for simplified geometry materialization; cache common zone queries; compress responses.
    - Load tests for /map/segments with realistic bbox/zoom.
- Person B
    - Reduce redraws; batch polyline updates; avoid overdraw; cache responses per tile key; throttle viewport requests.
- Tests today
    - Backend: k6/Bombardier run; P95 ≤250 ms; payload sizes monitored.
    - Mobile: 10-minute pan/zoom without GC thrash; FPS stable.
- Acceptance checks
    - App remains responsive; server meets latency SLOs.

- Artifacts
    - Perf dashboard snapshots; tuning notes.
- Pitfalls
    - Returning overly detailed polylines at low zoom; ensure zoom-aware tolerance.
    -

Day 16 — Security hardening

- Goals
    - Close common holes; finalize auth flows.
- Person A
    - Rate limit auth endpoints; lockout on brute force; CORS allowlist; HTTPS only; secrets via KeyVault/App Settings; DB least privilege.
    - Refresh token rotation with reuse detection; revoke on logout/all devices option.
- Person B
    - SecureStorage audit; explicit logout; token refresh retry; force re-auth on 401 with user-friendly flow.
- Tests today
    - Backend: ZAP/Burp light scan in staging; auth abuse tests; CORS preflight.
    - Mobile: 401/403 handling UX.
- Acceptance checks
    - Security checklist items closed; minimal scopes validated.
- Artifacts
    - Threat model doc; checklist in repo.
- Pitfalls
    - Logging secrets; re-validate redaction rules.

Day 17 — QA, test suite consolidation, pilot dataset

- Goals
    - High confidence through tests; pilot area verified.
- Person A
    - Expand rule unit tests; add end-to-end test: register → add vehicle → map request → session → scheduled push.
    - Produce Postman collection + seed database script for pilot.
- Person B
    - Automated UI smoke (login, vehicles, map, park here, receive push); manual checklist with edge cases (deny location, offline, rotate).
- Tests today
    - Full CI pipeline: unit + integration + e2e; artifacts uploaded.
- Acceptance checks
    - 80% coverage on rules and controllers; e2e passes; manual pilot checks (10 random streets) match expected statuses.
- Artifacts
    - Test plans; known-issues file.
- Pitfalls
    - Flaky tests due to time; use fixed clock abstractions.

Day 18 — Deploy & pilot build

- Goals
    - Staging/prod environment live; Android pilot APK distributed.
- Person A
    - Deploy API to Azure App Service (staging slot); Azure Database for PostgreSQL with PostGIS; connection strings via secrets; set HEALTHCHECK path; slot swap.
    - Set up Hangfire dashboard protected behind auth.
- Person B
    - Build signed Android package; internal distribution; capture logs/crashes via AppCenter or Firebase Crashlytics (optional).
- Tests today
    - /health and /ready green in cloud; smoke test endpoints; scheduled push test to a test device.
    - App installs and runs on 2–3 physical devices.
- Acceptance checks
    - Live pilot across a chosen neighborhood; at least 20 successful “park→notify” runs without crashes.
- Artifacts
    - Deployment guide; release notes; pilot feedback form.
- Pitfalls
    - Timezone misconfig in cloud; verify server uses UTC and client converts.

Weekly “Definition of Done” summary

- Week 1 done when
  - CI builds pass; auth + vehicles fully functional; MAUI app logs in and manages vehicles; map shows user and mock overlays.
- Week 2 done when
  - Map overlays come from real API with correct colors; details sheet works; sessions lifecycle complete; scheduled push arrives; deep-link works.
- Week 3 done when
  - Local fallback reminders work; settings influence rendering and notifications; performance and security targets met; tests green; cloud deployed; pilot APK delivered.

Who can work in parallel vs. dependencies (quick recap)

- Parallel most days except
  - Day 4–5: B needs A’s auth/vehicles endpoints.
  - Day 8–10: B needs A’s map/segment/session endpoints.
  - Day 12: B’s validation needs A’s scheduler live.