**Fastest MVP Build Order + Copy‑Paste Prompts**

**0) Foundation (Day 0)**

**Goal:** Create solution, wire logging, EF Core, auth skeleton.

**Deliverables:** Solution structure, Serilog, HealthChecks, DbContext, initial migration, OIDC sign‑in/out.

**Prompts to use:**

1. **Solution skeleton**

Create an ASP.NET Core .NET 8 solution with projects Domain, Application, Infrastructure, Web (Blazor Server + MVC). Add Serilog, HealthChecks, and a shared Directory.Build.props with nullable enabled. Show the .sln, project references, Program.cs wiring, and appsettings stubs.

1. **OpenIddict OIDC integration**

Wire OIDC auth using my OpenIddict server at https://user-auth.unece.org. Use cookie + AddOpenIdConnect (code flow). Map email and roles, enrich from DB on OnTokenValidated, and implement proper single logout (end-session + post\_logout\_redirect\_uri). Include secure cookie options and environment-specific appsettings.

1. **EF Core + baseline DbContext**

Add EF Core (SQL Server) with a single EoDbContext. Create the base tables: ApplicationUser, ApplicationRole, ApplicationUserRole, ActivityLog. Include rowversion on auditable entities, configure snake\_case table names, and generate the first migration + instructions to apply.

**What we shipped**

* **Solution structure**
  + Projects: EORequests.Domain, EORequests.Application, EORequests.Infrastructure, EORequests.Web.
  + Directory.Build.props with nullable enabled and common analyzers.
* **Logging**
  + Serilog bootstrapped in Program.cs (reads from appsettings\*, request logging middleware).
* **Health checks**
  + /health/live (self), /health/ready (DB) with JSON writer.
* **EF Core + DbContext**
  + EoDbContext (SQL Server), base entities: ApplicationUser, ApplicationRole, ApplicationUserRole, ActivityLog.
  + AuditableEntity with CreatedOn/By, ModifiedOn/By, and RowVersion (globally mapped to row\_version).
  + NEWSEQUENTIALID() for Guid PKs.
* **Auth skeleton (OIDC)**
  + Cookie scheme + AddOpenIdConnect (code flow).
  + Claims mapping resilient to OpenIddict/Azure AD.
  + Proper end‑session redirect (post\_logout\_redirect\_uri).
* **App composition**
  + Blazor Server + MVC controllers + Razor Pages enabled.

**Notable files**

* src/EORequests.Web/Program.cs
* src/EORequests.Infrastructure/Data/EoDbContext.cs
* src/EORequests.Domain/Audit/AuditableEntity.cs
* src/EORequests.Domain/Security/\* (User/Role/UserRole)
* appsettings.\* scaffolding for Serilog, OIDC, ConnectionStrings.

**1) Domain Model (Core Entities)**

**Goal:** Get core entities and enums ready for compilation.

**Deliverables:** Entities + value objects + configurations.

**Prompt:**

Define the Domain entities/enums for MVP: Request, RequestType, WorkflowTemplate, WorkflowStepTemplate, WorkflowInstance, WorkflowState, TaskItem, Attachment, SlaRule, EscalationRule, CommentThread, Comment, CommentReaction, Mention. Include key properties, relationships, and rowversion where needed. Provide EF Core configurations (Fluent API) in Infrastructure. Include enums: EntityType, AssignmentMode, WorkflowStateCode, TaskProgressStatus, CommentVisibility.

**What we shipped**

* Entities & relationships for MVP:
  + **Requests**: Request, RequestType
  + **Workflow**: WorkflowTemplate, WorkflowStepTemplate, WorkflowInstance, WorkflowState
  + **Work items**: TaskItem
  + **Files**: Attachment
  + **SLA & Escalation**: SlaRule, EscalationRule
  + **Collaboration**: CommentThread, Comment, CommentReaction, Mention
* Enums:
  + EntityType, AssignmentMode (RoleBased, SelectedByPreviousStep, AutoAssign), WorkflowStateCode, TaskProgressStatus, CommentVisibility.
* Fluent configs in Infrastructure (singular snake\_case tables, indexes, FKs, concurrency via RowVersion).

**Notable files**

* src/EORequests.Domain/Entities/\*.cs
* src/EORequests.Domain/Enums/\*.cs
* src/EORequests.Infrastructure/Configurations/\* (entity type configurations)

**2) Database Migrations & Seed**

**Goal:** DB is scaffolded with seed data.

**Prompt:**

Create EF Core migrations for all entities and seed initial data: 1 RequestType records (Consultant Request), one simple WorkflowTemplate each with 2–3 WorkflowStepTemplates, and a couple of roles (Requester, Reviewer, ProcessOwner, Admin). Provide the migration, seeder, and commands to update the DB.

**What we shipped**

* **Migrations** covering all entities above.
* **Seeder** that inserts:
  + RequestType: **Consultant Request** (CONSULT).
  + **WorkflowTemplate** CONSULT\_V1 with 3 steps: SUBMIT → REVIEW → APPROVE.
  + **Roles**: Reviewer, ProcessOwner, Admin (we purposely did **not** add Requester as a role; the requester is the creator/preparer).
  + **Sample data**: one Request + WorkflowInstance + initial WorkflowState so the UI opens with real data.
* Program.cs development bootstrap:
  + db.Database.MigrateAsync() + DbSeeder.SeedAsync(db).
* **Hangfire** wired (see Step 5) but seeding is independent.

**Commands**

dotnet ef migrations add Step1\_DomainModel \

--project src/EORequests.Infrastructure --startup-project src/EORequests.Web

dotnet ef database update \

--project src/EORequests.Infrastructure --startup-project src/EORequests.Web

**Notable files**

* src/EORequests.Infrastructure/Data/DbSeeder.cs
* src/EORequests.Web/Program.cs (dev: migrate + seed)

**3) Access Control Service + Policies**

**Goal:** Centralize per‑step permissions.

**Prompt:**

Implement IAccessControlService that evaluates permissions for actions: by role, by specific assignee (SelectedByPreviousStep), and flags (AllowCreatorOrPreparer). Add authorization policies (named) and show how a Blazor page or controller checks them.

**What we shipped**

* **Centralized permissions**:
  + IAccessControlService with CanPerformAsync(workflowStateId, userId, roles, action) and GetAllowedActionsAsync(...).
  + Rules: **Admin override**, **role‑based** via WorkflowStepTemplate.AllowedRolesCsv, **specific assignee** via WorkflowState.AssigneeUserId, **creator/preparer** via AllowCreatorOrPreparer, and **no Act** when step IsComplete.
* **ASP.NET Authorization integration**:
  + Custom requirement: StepActionRequirement.
  + Resource: WorkflowStateResource { WorkflowStateId, UserId, Roles }.
  + Handler: StepActionAuthorizationHandler calls IAccessControlService.
  + **Named policies**: Step\_View, Step\_Act, Step\_Upload, Step\_Comment, Step\_CreateTask, Step\_Assign.
* **Current user accessor**:
  + ICurrentUser + CurrentUser to read app\_user\_id (and email/roles) from the cookie.
  + OIDC OnTokenValidated stamps "app\_user\_id" claim by ensuring the user exists in ApplicationUsers at sign‑in.
* **Controller example**:
  + WorkflowActionsController.Act(stateId) demonstrates policy usage, try/catch, logging, and ProblemDetails pattern via a base controller.

**Notable files**

* src/EORequests.Application/Interfaces/IAccessControlService.cs
* src/EORequests.Infrastructure/Services/AccessControlService.cs
* src/EORequests.Web/Security/StepActionRequirement.cs
* src/EORequests.Web/Security/StepActionAuthorizationHandler.cs
* src/EORequests.Web/Security/ICurrentUser.cs, CurrentUser.cs
* src/EORequests.Web/Controllers/WorkflowActionsController.cs
* src/EORequests.Web/Infrastructure/BaseApiController.cs (ProblemDetails helpers)
* src/EORequests.Web/Program.cs registrations:
  + AddScoped<IAccessControlService, AccessControlService>()
  + AddSingleton<IAuthorizationHandler, StepActionAuthorizationHandler>()
  + Named policies via builder.Services.AddAuthorization(...).
  + OIDC OnTokenValidated enrichment (stamps app\_user\_id).

**4) Workflow Engine v1 (Transitions & Assignment)**

**Goal:** Minimal engine to move steps, compute assignees, and emit events.

**Prompts:**

1. **Engine core**

Implement WorkflowEngine with APIs: StartInstance(requestId), CanAdvance(instanceId), Advance(instanceId, byUserId), SkipOrBranch(instanceId, ruleKey). Support assignment modes: RoleBased, SelectedByPreviousStep, AutoAssign. Emit domain events for StepActivated, StepCompleted, AssignmentChanged.

1. **Unit tests**

Provide xUnit tests covering transitions, assignment modes, and a simple branching rule.

**What we shipped**

* **Engine APIs**
  + StartInstanceAsync(requestId, startedByUserId) → creates WorkflowInstance, first WorkflowState, computes assignee per step’s AssignmentMode, emits StepActivated.
  + CanAdvanceAsync(instanceId) → guards completion (and checks for gating tasks, ready for Step 10).
  + AdvanceAsync(instanceId, byUserId) → completes current step, activates next step (or completes workflow), emits StepCompleted, StepActivated, and AssignmentChanged (if assignee changed).
  + SkipOrBranchAsync(instanceId, ruleKey, byUserId) → completes current as “Skipped”, jumps to a target order (goto:n) or end, same events.
* **Assignment modes**
  + AutoAssign → request creator.
  + SelectedByPreviousStep → the acting user.
  + RoleBased → no explicit AssigneeUserId (RBAC enforced by AccessControl policies).
* **Branching**
  + Minimal IBranchRuleEvaluator + BranchRuleEvaluator:
    - goto:n → jump to step order n.
    - end → finish the workflow.
    - default → next order.
* **Domain events**
  + Records: StepActivated, StepCompleted, AssignmentChanged.
  + IDomainEventDispatcher + LoggingEventDispatcher (logs now; can swap to outbox later).
* **Unit tests**
  + WorkflowEngineTests (xUnit + InMemory EF): start, advance through all steps, branch to specific order.

**Notable files**

* src/EORequests.Application/Interfaces/IWorkflowEngine.cs
* src/EORequests.Infrastructure/Services/WorkflowEngine.cs
* src/EORequests.Application/Interfaces/IBranchRuleEvaluator.cs
* src/EORequests.Infrastructure/Services/BranchRuleEvaluator.cs
* src/EORequests.Application/Interfaces/IDomainEventDispatcher.cs
* src/EORequests.Infrastructure/Services/LoggingEventDispatcher.cs
* Tests: tests/EORequests.Tests/WorkflowEngineTests.cs

**5) SLA Service + Hangfire Jobs**

**Goal:** Due dates, reminders, single escalation.

**Prompt:**

Implement SlaService that computes DueOn on step activation using SlaRule. Create Hangfire jobs: reminders (T-3, T-1) and a single escalation (T+1). Ensure idempotent job keys and outbox-friendly design.

**What we shipped**

* **SLA service**
  + ISlaService with:
    - ComputeAndSetDueDateAsync(state) → sets DueOn from SlaRule.DueDays (defaults to 17:00 UTC on due date).
    - ScheduleReminderAndEscalationJobsAsync(state) → schedules **T‑3**, **T‑1** reminders and **T+1** escalation.
    - CancelJobsForStateAsync(stateId) → deletes pending jobs for a step completed/skimmed early.
* **Hangfire jobs**
  + **Idempotent keys** (no duplicates):  
    sla:reminder:T-3:{stateIdN}, sla:reminder:T-1:{stateIdN}, sla:escalation:T+1:{stateIdN}.
  + **Instance job runner (DI‑friendly)**: ISlaJobRunner, SlaJobRunner with SendReminder and Escalate (currently log; Step 9 will notify).
  + Scheduling uses generic overloads so Hangfire resolves the job from DI:
    - BackgroundJob.Schedule<ISlaJobRunner>(r => r.SendReminder(...), when)
    - BackgroundJob.Schedule<ISlaJobRunner>(r => r.Escalate(...), when)
* **Engine integration**
  + On **activation** (start/advance/branch): SLA **compute + schedule** for the new state.
  + On **completion/skip**: SLA **cancel** for the old state.
* **Recurring job**
  + (From earlier) Daily personnel sync @ **04:00** is already in Hangfire; SLA jobs are per‑state schedules.

**Notable files**

* src/EORequests.Application/Interfaces/ISlaService.cs
* src/EORequests.Infrastructure/Services/SlaService.cs
* src/EORequests.Infrastructure/Services/ISlaJobRunner.cs
* src/EORequests.Infrastructure/Services/SlaJobRunner.cs
* src/EORequests.Infrastructure/Services/WorkflowEngine.cs (SLA hooks added)
* Tests:
  + tests/EORequests.Tests/SlaIntegrationTests.cs (Moq): verifies **compute+schedule** on start, **cancel+schedule** on advance/branch.
  + Updated WorkflowEngineTests to pass an ISlaService mock.

**Program.cs additions**

builder.Services.AddScoped<ISlaService, SlaService>();

builder.Services.AddScoped<ISlaJobRunner, SlaJobRunner>();

**6) Dynamic Form Wizard (JSON Schema)**

**Goal:** Render forms per step with conditional fields.

**Prompt:**

Build a Blazor component JsonFormWizard that renders step forms from a JSON schema with conditional visibility rules (simple expressions on prior field values). Show schema examples for Consultant Request and ICT Support, including attachments.

**7) Comments & @Mentions (2‑level threads)**

**Goal:** Threaded comments with internal/requester‑visible modes and mentions.

**Prompt:**

Implement CommentService with APIs: AddComment(thread, body, visibility, parentId?), ListThread(entityType, entityId), AddReaction(commentId, emoji), ResolveMentions. Enforce 2-level reply depth. Provide a Blazor CommentsPane with @mention picker, unread counters, and SignalR live updates.

**8) Attachments Module**

**Goal:** Upload, inline view, soft‑delete purge.

**Prompt:**

Implement AttachmentService with allowed file types, size limit, AV scan hook interface, and inline streaming (range support). Provide UI for drag/drop and an inline viewer. Add a Hangfire purge job for soft-deleted files after 90 days.

**9) Notifications (SignalR + Email)**

**Goal:** Real‑time toasts + email templates.

**Prompt:**

Add a SignalR hub /hubs/notifications and a NotificationService that pushes events for submission, assignment, mentions, and overdue. Provide Razor email templates for the same events and a user notification preferences store.

**10) Task Module (Gating)**

**Goal:** Step can’t complete until tasks done.

**Prompt:**

Implement TaskService with task/subtask dependencies and a gating flag that prevents step completion while open tasks exist. Provide a Blazor Task tab to create/update tasks and attach files.

**11) Dashboards (My Requests / Actions / SLA)**

**Goal:** Basic insights for MVP.

**Prompt:**

Create read models and Blazor pages for **My Requests**, **My Actions**, and **SLA Status** (on-time, due soon, overdue). Add CSV export actions and fast filters (type, status, assignee).

**12) Admin UI (Workflow & Form Designer — basic)**

**Goal:** Configure templates & steps without code.

**Prompt:**

Scaffold Admin pages to CRUD WorkflowTemplate/WorkflowStepTemplate, define assignment mode, AllowedRoles, SLA rules, and upload a JSON schema for step forms. Include a Permission Matrix view (Step × Role/User).

**13) Audit Timeline**

**Goal:** Transparent history.

**Prompt:**

Implement ActivityLog write helpers for create/update/status/assignment/comment events and show a timeline tab on the Request details page. Support filtering by event type.

**14) UAT Hardening & DoD**

**Goal:** MVP done.

**Prompt:**

Provide a UAT checklist and scripts: seed both workflows, run SLA jobs, test notifications, verify RBAC, restore drill, and performance counters. Include p95 latency checks (<200ms read, <500ms write) and a security scan checklist.

**🧠 How to use prompts during the project**

* Use **focused prompts per feature** (like the ones above), not a single giant prompt.
* When you hit an edge case, ask a **micro‑prompt**:

In WorkflowEngine.Advance, show how to enforce Task gating and return a friendly error if tasks exist.

* For fixes/refactors, include **the file names** and I’ll provide patches/diffs.

**🗺️ Suggested weekly flow (solo dev)**

**Week 1:** 0–3 (foundation, domain, migrations, access control)  
**Week 2:** 4–7 (workflow v1, SLA jobs, JSON forms, comments)  
**Week 3:** 8–11 (attachments, notifications, tasks, dashboards)  
**Week 4:** 12–14 (admin basics, audit timeline, UAT & DoD)

**Bonus: External personnel sync (separate, but landed during this work)**

* IExternalUserSyncService + ExternalUserSyncService (typed HttpClient with BaseUrl = https://eoportal.unece.org/Personnel/ and header X-Api-Key).
* Sync logic: upsert active users, deactivate missing, idempotent; Hangfire recurring job @ 04:00.
* Program.cs: typed client configuration, secrets via dotnet user-secrets, dashboard /hangfire protected by AdminOnly.