00_root_and_monorepo_...
Document

01_api_bootstrap_and_per...
Document

02_api_backbone_identity...
Document

03_api_backbone_organiz...
Document

04_api_config_profiles_an...
Document

05_api_core_and_supporti...
Document

06_api_domain_insights_s...
Document

07_web_app_shell_pages_...
Document

08_web_orgo_frontend_ty...
Document

index those files

Thought for 27 seconds

Here is a clear index of all files you uploaded, grouped in the order they appeared.

Index of Uploaded Files

00 – Root & Monorepo Meta

1. **00_root_and_monorepo_meta.txt** 00_root_and_monorepo_meta

01 – API Bootstrap & Persistence

2. **01_api_bootstrap_and_persistence.txt** 01_api_bootstrap_and_persistence

02 – API Backbone: Identity & Persons

3. **02_api_backbone_identity_and_persons.txt** 02_api_backbone_identity_and_pe...

03 – API Backbone: Organizations & RBAC

4. **03_api_backbone_organizations_and_rbac.txt** 03_api_backbone_organizations_a...

04 – API Config, Profiles & Feature Flags

5. **04_api_config_profiles_and_feature_flags.txt** 04_api_config_profiles_and_feat...

05 – API Core & Supporting Core

6. **05_api_core_and_supporting_core.txt** 05_api_core_and_supporting_core

06 – API Domain: Insights, Security & Orgo Module

7. **06_api_domain_insights_security_and_orgo_module.txt** 06_api_domain_insights_security...

07 – Web App Shell, Pages & Store

8. **07_web_app_shell_pages_and_store.txt** 07_web_app_shell_pages_and_store

08 – Web Orgo Frontend Types, Screens & Shared UI

9. **08_web_orgo_frontend_types_screens_and_shared_ui.txt** 08_web_orgo_frontend_types_scre...

If you want a *hierarchical index*, a *cross-reference map*, or a *functional inventory* (e.g., list all modules, controllers, services, DTOs), specify the format and I will generate it.

Sources

8-orgo-cyclic-overview-la...
Fichier

7-orgo-organization-prof...
Fichier

6-orgo-insights-module-...
Fichier

5-orgo-Core-Services-Sp...
Fichier

4-orgo-functional-code-n...
Fichier

3-orgo-full-stack-technic...
Fichier

2-orgo-documentation-in...
Fichier

1-orgo-database-schema-...
Fichier

index those files

Thought for 34 seconds >

Below is a complete, direct **index of all files you uploaded**, in the order they appeared. Each entry includes its citation token so you can reference it precisely.

Index of Uploaded Files

Earlier batch (Docs 00–08)

1. **00_root_and_monorepo_meta.txt** 8-orgo-cyclic-overview-labels-a...
2. **01_api_bootstrap_and_persistence.txt** 7-orgo-organization-profiles-an...
3. **02_api_backbone_identity_and_persons.txt** 6-orgo-insights-module-config-p...
4. **03_api_backbone_organizations_and_rbac.txt** 5-orgo-Core-Services-Specificat...
5. **04_api_config_profiles_and_feature_flags.txt** 4-orgo-functional-code-name-inv...
6. **05_api_core_and_supporting_core.txt** 3-orgo-full-stack-technical-spec
7. **06_api_domain_insights_security_and_orgo_module.txt** 2-orgo-documentation-index
8. **07_web_app_shell_pages_and_store.txt** 1-orgo-database-schema-reference
9. **08_web_orgo_frontend_types_screens_and_shared_ui.txt** 00_root_and_monorepo_meta

---

Newly uploaded Orgo v3 documentation (Docs 1–8)

10. **1-orgo-database-schema-reference.md** 03_api_backbone_organizations_a...
11. **2-orgo-documentation-index.md** 6-orgo-insights-module-config-p...
12. **3-orgo-full-stack-technical-spec.md** 5-orgo-Core-Services-Specificat...
13. **4-orgo-functional-code-name-inventory.md** 4-orgo-functional-code-name-inv...
14. **5-orgo-Core-Services-Specification.md** 3-orgo-full-stack-technical-spec
15. **6-orgo-insights-module-config-parameters.md** 6-orgo-insights-module-config-p...
16. **7-orgo-organization-profiles-and-cyclic-overview.md** 7-orgo-organization-profiles-an...
17. **8-orgo-cyclic-overview-labels-and-flow-rules.md** 8-orgo-cyclic-overview-labels-a...

If you'd like:

I can also generate any of the following:

- A hierarchical doc tree
- A dependency map showing how all docs reference one another
- A cross-reference index (e.g., “where each enum is defined”)
- A unified spec index across backend + frontend + configs
- A table that maps each file to module, purpose, and schema sections

Tell me the format you prefer.

Sources

Analyse the code. whichs files remains to be coded, or updated, or fixed?

To help determine which files still need to be coded, updated, or fixed, could you clarify what you mean by "analyse the code"? Specifically:

1. Are you asking for a **feature coverage audit** (i.e., which specification items have not been implemented)?
2. Are you looking for **missing or outdated code** relative to the spec documents (Docs 1–8)?
3. Should I focus on **backend (NestJS/Prisma)**, **frontend (NextJS/RTK Query)**, or both?
4. Do you want a **file-level checklist**, **module-level summary**, or detailed line-item analysis?

This will help me give you a useful, actionable report.

all. or focus on biggest problems.

Understood. I'll review the full codebase and documentation to identify major gaps, outdated components, or unimplemented parts across both frontend and backend. This includes incomplete features, missing handlers or configurations, or inconsistency with spec documents.

I'll get back to you with a structured breakdown of the key problem areas, what's missing or incomplete, and recommended fixes. You can continue working or asking questions while I dig in.

Orgo Codebase Gap Analysis – Missing & Incomplete Features

Overview of Critical Gaps and Issues

Domain Modules – Maintenance Missing: The *Maintenance* domain module is not implemented in the codebase, even though it's defined in the schema (tables like `maintenance_assets` etc.) and expected by the spec. Unlike HR and Education, there is no `MaintenanceModule` or service/controller in the API (only `HrModule` and `EducationModule` are imported in the app module). This means maintenance-related tasks (type `"maintenance"`) are not handled by any domain service or API endpoints – a major feature gap.

Domain Module Config & Handlers – Spec vs Implementation: According to the spec, each domain should have a YAML config and handler with hooks (e.g. `maintenance_module.yaml`, `maintenance_handler.py`). In code, however, domain logic is hard-coded in TypeScript services (e.g. `HrModuleService`, `EducationModuleService`) and there's no dynamic loading of `<domain>_module.yaml`. For example, the HR module code defines a constant `HR_DOMAIN_TYPE = 'hr_case'` with a note to "stay aligned" with *hr_case_module.yaml*, but there is no actual YAML parsing or handler interface. This indicates the YAML-driven configuration system is not actually implemented – domain rules (allowed categories, subtypes, etc.) are embedded in code instead of external config. The lack of a `DomainRegistry` or config loader (mentioned in Doc 3) means domain modules are not discoverable or pluggable as intended.

Workflow Engine & Rules – Partial Implementation: Core workflow services exist in name (there is a `WorkflowEngineService` and controller in the code) but appear only partially implemented. The Workflow controller provides an endpoint to execute workflows and supports dry-run, and the service likely stubs out `executeWorkflow`, `simulate`, etc. However, there is no evidence of robust rule parsing or a library of workflow definitions. The spec's features like rule validation and support for actions (CREATE_TASK, ROUTE, NOTIFY, etc.) are not fully realized in code – likely placeholders without the full rule evaluation engine. Similarly, escalation policies (periodic check for `reactivity_deadline_at`) are defined in the spec but no `EscalationService` or scheduled job was found in the code. This suggests that automatic task escalation and workflow-driven task creation from signals may not be functioning yet.

Email Ingestion & Routing – Incomplete Integration: The **Email Gateway** exists (`EmailModule` with services for parsing, validation, etc.) and can send outbound mail. However, connecting inbound emails to the workflow engine and domain routing is likely only partly done. The `EmailRouterService` is provided (to hand off parsed emails to the workflow engine), but since domain pattern configs are not loaded from YAML, any logic to match an email to a domain (e.g. maintenance vs. HR) is limited. In other words, the code doesn't have the email pattern rules from `email_patterns` in domain configs, so routing incoming emails to the correct domain workflow may not work until those rules are coded. This is a gap between the documented email-to-task flow and the current implementation.

Notification Service – Lacking Features: The spec calls for a dedicated notification service (`notifier_service`) with multi-channel (email, SMS, in-app) support. The codebase does not have a distinct NotificationModule – notifications are partially covered by the EmailService (for email channel) but there's no implementation of in-app or SMS notifications, and no use of the `notifications` table. There is also no code for "send_task_notification" logic that respects visibility or profile settings (as outlined in Doc 5). This means features like in-app alerts or role-based notification filtering are not yet built – a notable omission for production use.

Logging, Audit & Security – Partial: A `LogService` (likely as part of `LoggerModule`) exists for writing log events, and the schema has `activity_logs` and `security_events`. However, enforcement of guardrails (PII masking, visibility-based restrictions on logs/exports) is not evident. For instance, *visibility* levels (`PUBLIC/INTERNAL/RESTRICTED/ANONYMISED`) are defined and used on tasks, but we did not find code that filters audit logs or query results based on the viewer's role or the data's visibility. Similarly, retention policies for logs (per profile) are defined in docs but no code references to log retention or rotation. Thus, the groundwork for audit logging is there, but important safeguards and data-retention features remain to be coded.

Configuration & Profiles – Defined but Unused: The code has an `OrgoConfigModule/Service` for loading configuration, and the spec details YAML config files (database, email, logging configs, etc.). In practice, the config service likely only loads environment variables (e.g. `DATABASE_URL`). There's no indication that it reads the profile YAML or any `*.yaml` at runtime. Organization profiles are stored in the DB (`organization_profiles` table) and a set of profile "codes" (like `hospital`, `retail_chain`) are mentioned in docs, but the **Org Profile** settings are not actively applied in code logic. For example, the *reactivity and SLA* parameters in a profile should influence task deadlines and escalation. The TaskService currently uses a fixed default for `reactivity_deadline_at` and notes that in a "later iteration" this should derive from OrgProfile and rules. This iteration has not happened – tasks don't yet adjust SLA or escalation based on an org's profile. Likewise, features like pattern sensitivity or transparency levels per profile are not hooked up. This is an inconsistency: the schema and docs support rich org-specific behavior, but the running code still behaves in a one-size-fits-all manner.

Insights & Cyclic Overview – Mostly Not Implemented: The **Insights** module (analytics star schema) is present in the database and a TypeORM connection for `insights` is set up. However, there is no service code to populate or query the `insights.fact_*` tables. The spec envisions ETL jobs and pattern detection feeding a cyclic review process (weekly/monthly audits), but we found no `CaseReviewService` or scheduled job for cyclic reviews in the code. The absence of these pieces means Orgo's feedback loop – detecting recurring issues and opening review cases – is not functional. The **insights config** (thresholds, cache settings, etc.) defined in Doc 6 is likely not used at all. This is a major feature gap relative to the spec's promise of "patterns turned into work items."

Test Coverage & Tooling – Gaps: There is essentially no test coverage visible. The repository has a Jest config and empty test placeholders, but we did not find any `.spec.ts` files or meaningful unit tests. Core services like `createTask` or `workflowEngine.executeWorkflow` have no automated tests, despite the spec's non-functional requirements emphasizing testing (e.g. unit tests for each core function, integration tests for email→task→notification flows). This lack of tests is risky and indicates technical debt. Additionally, CLI or developer tools (e.g. a bootstrap script to create an initial admin user, or a CLI to run maintenance jobs) are either absent or very minimal. The `package-scripts.js` defines some yarn scripts for building, but nothing like a CLI command to manually trigger workflows or ingest offline data. Teams will likely need such tools, so their absence is notable.

File/Module Status Checklist

Below is a checklist of key files/modules and their implementation status, highlighting what is OK, missing, incomplete, or inconsistent with the specifications:

Module / Feature	File(s) or Area	Status
Maintenance Domain Module	*No code present* (DB tables exist)	**Missing** – No `MaintenanceModule` or service (schema defined but no implementation).
HR Domain Module	`apps/api/src/orgo/domain/hr/*` (HrModule, Service, Controller)	**Partially Implemented** – Supports creating HR cases ar tasks with proper schema links. However, logic is embedded via raw SQL and custom code (no YAML

Module / Feature	File(s) or Area	Status
		handler). Some HR-specific rules (e.g. auto-anonymizin severe cases) may not be fully enforced.
Education Domain Module	`apps/api/src/orgo/domain/education/*` (Module, Service, Controller)	**Implemented** – Provides endpoints to register and list incidents with learning group/person context. Aligns w spec, though also hard-coded logic.
Maintenance Domain Module	*Not present*	**Missing** – No endpoints or services for maintenance ta Should be prioritized to implement (domain type `"maintenance"` is unused currently).
Domain Module Config Loading	`domain_modules/` (expected directory)	**Not implemented** – The code does not load YAML con for domains. Domain settings are hard-coded, violating intended modularity (e.g. HR module expects `hr_case_module.yaml` alignment but uses constants instead).
Workflow Engine	`apps/api/src/orgo/core/workflow/*` (WorkflowEngineService, WorkflowController)	**Incomplete** – Basic structure exists (execute/validate endpoints) but full rule evaluation engine is likely stubb No evidence of comprehensive workflow rule parsing c storage of workflow definitions.
Escalation & SLA Policies	(Part of Task/Workflow services)	**Incomplete** – Escalation is partly handled (TaskService c set status to ESCALATED), but no scheduled job or EscalationService checks deadlines. `reactivity_deadli` is set with a default formula; integration with org profil or escalation policies is not done.
Email Gateway	`apps/api/src/orgo/core/email/*` (EmailModule, EmailService, Parser, etc.)	**Partially Implemented** – Outbound email sending via SMTP (using nodemailer) is implemented. Inbound em polling/parsing classes exist, but how far they go in creating tasks is uncertain. Likely needs further integrat testing, especially linking parsed emails into the workfl engine for task creation.
Notification Service	*No dedicated module* (email covers some notifications)	**Outdated/Partial** – There's no `NotifierService` for in app/SMS channels as in spec. Email notifications on tas events might be sent via EmailService, but other chann and the `notifications` table are unused. This should

Module / Feature	File(s) or Area	Status
		developed or toggled via feature flags (e.g. `orgo.insights.enabled`) but currently is a gap.
Logging & Audit	`apps/api/src/orgo/core/logging/*` (LogService, LoggerModule)	**Partially Implemented** – The LogService is used to log events (e.g. queue metrics, errors). However, the finer points like log filtering by visibility or automated log retention (per profile's retention_profile) are not coded Also, the `security_events` (for sensitive actions) are n emitted anywhere we can see.
Configuration System	`apps/api/src/orgo/core/config/*` (OrgoConfigService, ConfigModule)	**Inconsistent/Incomplete** – The structure for config files present and the code uses NestJS ConfigModule for en vars. But YAML files (database_connection.yaml, email_config.yaml, etc.) are not actually being read, des the module design. For example, EmailModule expects OrgoConfigModule for email settings, yet likely falls ba to env defaults. This config system needs to be comple (e.g. load YAML per environment).
Feature Flags	`apps/api/src/orgo/config/feature-flag/*` (FeatureFlagService, FeatureFlagController)	**Implemented (API)** – CRUD APIs for feature flags at org/env scope are provided and they persist to the `feature_flags` table. However, usage of these flags in code is minimal. They are not yet woven into condition logic (meaning toggling a flag currently has no effect o workflows or features until code reads it).
Organization Profiles	`apps/api/src/orgo/backbone/organizations/*` (OrgService, OrgProfile logic)	**Partially Implemented** – Org profiles can be created/re (the `OrganizationModule` and profile DTOs exist), but t **behavior tuning** they contain is not utilized. E.g. an org profile code (like "hospital") doesn't automatically chan SLA or visibility rules in the running system. The code comment in TaskService confirms SLA is using a fixed default pending profile integration. This inconsistency should be resolved by applying profile settings to Task creation, escalation, and Insights thresholds.
Insights & Analytics	`apps/api/src/orgo/insights/*` (InsightsModule) and DB `insights.*` tables	**Mostly Missing** – Aside from establishing a DB connec for the `insights` schema, there's no code to populate fact/dimension tables or to expose analytics via API. Fo

Module / Feature	File(s) or Area	Status
		example, no `InsightsController` or services for patter detection exist. Thus, the Insights module remains large theoretical – the foundation is there, but the ETL jobs a reporting endpoints have not been built.
Cyclic Review Process	*No explicit service (expected CaseReviewService)*	**Missing** – The weekly/monthly/annual case review logi described in Doc 8 is not implemented. No scheduled j creates review cases for patterns like "≥N incidents in 3 days". This is an important oversight, as the "oversight loop" is a key Orgo value proposition. Adding a `CaseReviewService.runCyclicReview()` job (and corresponding workflow rules) should be a priority.
Offline Sync	DB tables (`offline_nodes` , `sync_*`) and some code in PersistenceService	**Incomplete** – The schema supports offline node sync, b we didn't find client sync logic or conflict resolution implemented in code. The PersistenceService has hook SQLite, but an actual offline mode ("ENVIRONMENT = offline") is likely untested. This may not be critical for M but it's a feature flagged in the spec and needs attentic offline usage is intended.
Testing & QA	*Test files, CI pipeline*	**Gap** – Little to no automated tests were found. Core lo (task lifecycles, email parsing, RBAC) lacks unit tests, wh is risky. Additionally, end-to-end scenario tests (email - task → escalation) are not present. This gap means the current code may have undiscovered bugs or regressio Investing in test development and scenario simulation (perhaps using the spec's checklists as a guide) is highl recommended.
Developer Tooling	*CLI scripts, seed data*	**Gap** – There is no CLI or admin console to perform common tasks (seeding initial data, running maintenan workflows, backfilling analytics). Developers must manu call APIs or craft SQL for these tasks. Over time, adding commands (e.g. to create an org, trigger the cyclic revi run one-time migrations) would improve productivity a ops.

Recommendations and Next Steps

Given the above gaps, here are the **priority areas to address**:

- **Implement the Maintenance Module:** Develop the Maintenance domain module to parity with HR/Education. This includes a `MaintenanceModuleService` (with methods like `registerIncident`, `listIncidents` similar to those in HR/Education), and hooking up maintenance asset linking (use the `maintenance_task_links` table). This will fulfill a major missing piece and allow end-to-end testing of a "generic" domain module as envisioned.
- **Finish Workflow Engine Features:** Build out the workflow rule engine so that it can evaluate configured rules (likely to be stored in YAML or DB). Ensure that email ingestion triggers the engine: e.g. if an incoming email matches HR patterns, the engine runs the HR workflow (perhaps auto-creating a task or case). Also implement escalation rule checks – perhaps as a scheduled job that invokes `taskService.escalateTask` for overdue tasks. This will involve using OrgProfile's `reactivity_profile` (SLA timings) to set deadlines and then comparing against `reactivity_deadline_at`. Essentially, the logic in **Doc 5 §5.3 (SLA & escalation)** needs coding.
- **Integrate YAML Config & Profiles:** Make the `OrgoConfigService` actually load YAML files for core configs (email, notifications, logging) and domain configs. This might involve adding a file loader (perhaps reading from a `config/` directory) and validating against the schema in Doc 2. For example, load `email_config.yaml` to configure email polling intervals, or load `profiles.yaml` to pre-populate profile templates. At a minimum, ensure that the environment-specific settings (e.g. dev vs prod in config) are honored. Similarly, use the organization's profile at runtime: for instance, when creating a task, apply the org's `transparency_profile` to default the Task's visibility, or use the `pattern_sensitivity_profile` to determine if an incident triggers an alert. This will close the loop between the **configuration data** and application behavior.

- **Develop Notification and Audit Features:** Introduce a NotificationService that can handle in-app notifications and possibly SMS/webhooks in the future. This service should read the `notification_config.yaml` (channels enabled, templates for emails, etc.) and send notifications on relevant task events. Even a basic in-app notification (writing to a `notifications` table and WebSocket gateway) would satisfy much of the spec. On the audit side, implement writing **security_events** for actions like export or profile changes, and ensure PII is masked in logs for restricted items (perhaps using the `visibility` field to decide). These enhancements will align the implementation with the guardrails specified (privacy by design).
- **Cyclic Overview & Insights:** Plan and implement the cyclic review job. This could be a cron-triggered service that queries for patterns (for example, use the analytics tables or even the operational tables to find "N tasks of category X in the last Y days") and then creates a summary Case if thresholds are exceeded. Since the analytics ETL isn't done, an interim approach might query the `tasks` table directly for patterns. Over time, build the ETL to populate `insights.fact_tasks` and use those for reporting. Also, implement at least one Insights API endpoint – e.g. an endpoint to get basic metrics (counts of tasks by status, etc.) to verify the star schema is being filled. These steps will begin to deliver on Orgo's promise of turning operational data into oversight.
- **Testing & Quality:** Backfill unit tests for critical services. Start with TaskService (cover createTask with various inputs, updateStatus transitions ensuring illegal transitions are rejected, etc.), then WorkflowEngineService (test that a sample workflow definition leads to expected calls), and Domain services (test that HR `registerReport` creates case, task, links correctly). Use the **spec's checklists** (like Doc 3's Domain Module checklist) as test criteria. Additionally, test multi-tenancy enforcement (ensuring one org cannot access another org's data via the APIs). Setting up these tests will catch regressions early and increase confidence as the team fills in the remaining features.

By addressing these areas, the Orgo codebase will become much more aligned with Docs 1–8. In summary, the foundation is strong – multi-tenant core, canonical schemas, and some core services are in place – but significant portions (maintenance domain, dynamic configs, full workflow rules, insights feedback loop, etc.) remain **incomplete**. Tackling the missing domain module first would deliver immediate functional breadth, while workflow/insights improvements will deliver the depth and automation promised in the design. Each implemented fix or feature should be validated against the spec (ensuring enums, invariants, and flows remain consistent) so that documentation and code converge. With these gaps filled and thorough testing, Orgo can progress from a partially implemented prototype to a robust platform meeting its ambitious cross-domain, cross-organization goals.

Sources

from the report you gave, which files needs update?
Simple list, with numbers in the path

Thought for 7m 57s

1. apps/api/src/orgo/domain/hr/hr.service.ts
2. apps/api/src/orgo/domain/education/education.service.ts
3. apps/api/src/orgo/core/tasks/task.service.ts
4. apps/api/src/orgo/core/workflow/workflow-engine.service.ts
5. apps/api/src/orgo/core/workflow/escalation.service.ts
6. apps/api/src/orgo/core/email/email-router.service.ts
7. apps/api/src/orgo/core/signals/signal-ingest.service.ts
8. apps/api/src/orgo/core/notifications/notification.service.ts
9. apps/api/src/orgo/core/logging/log.service.ts
10. apps/api/src/orgo/config/config.service.ts
11. apps/api/src/orgo/config/org-profile.service.ts

12. apps/api/src/orgo/config/feature-flag.service.ts
13. apps/api/src/orgo/backbone/organizations/organization.service.ts
14. apps/api/src/orgo/insights/reports/reports.service.ts
15. apps/api/src/orgo/insights/insights-cache-warmup.service.ts
16. apps/api/src/orgo/insights/pattern-detection.service.ts
17. apps/api/src/orgo/core/offline/sync.service.ts

Next step, you will make me 17 prompts, each prompt in "copy window". Those prompts will give instructions for each of the 17 files you listed. Use a number for each prompt. In those prompt, mention: Fetch original code from the dump I gave you in a txt. Give full final version of code. Validate for alignment on the docs I provided in this conversation.

Thought for 5m 54s

text

Copier le code

```
1. apps/api/src/orgo/domain/hr/hr.service.ts

Fetch the original code for `apps/api/src/orgo/domain/hr/hr.service.ts` from the r

Then:
- Analyse the current implementation of the HR domain service against the Orgo v3
- Refactor/extend the service so that:
  - It uses the canonical Task and Case lifecycles and enums.
  - It correctly models the `hr_case` domain type, participants, confidentiality
  - It is multi-tenant safe (always scoped by `organization_id`).
  - It aligns with the documented workflows for registering HR reports and listing
- Produce the **full final version** of `apps/api/src/orgo/domain/hr/hr.service.t
- Validate that the final code is fully aligned with the docs I provided in this
```

text

Copier le code

```
2. apps/api/src/orgo/domain/education/education.service.ts

Fetch the original code for `apps/api/src/orgo/domain/education/education.service

Then:
- Compare the current implementation to the Education domain specification in the
- Update the service so that:
  - It uses the canonical Task enums and lifecycle for `education_support` incide
  - It correctly creates and links tasks to learning groups and persons as per th
  - It provides robust, typed list/query functions that match the expected filter
  - All SQL/Prisma usage is safe, multi-tenant, and consistent with other domain
- Output the **full final version** of `apps/api/src/orgo/domain/education/educat
- Validate that the final code is aligned with all relevant docs I provided (data
```

text Copier le code

```
3. apps/api/src/orgo/core/tasks/task.service.ts

Fetch the original code for `apps/api/src/orgo/core/tasks/task.service.ts` from t

Then:
- Analyse how this service currently implements Task creation, listing, status up
- Refine/extend it so that:
  - Task lifecycle and transitions strictly follow the Orgo v3 spec (allowed tran
  - All canonical enums and JSON shapes match the docs and the web `Task` types.
  - It integrates correctly with organization profiles for reactivity/SLA default
  - It is fully multi-tenant and uses the standard result/error patterns used els
- Produce the **full final version** of `apps/api/src/orgo/core/tasks/task.servic
- Validate alignment with the docs I provided (DB schema, API core & supporting c
```

text Copier le code

```
4. apps/api/src/orgo/core/workflow/workflow-engine.service.ts

Fetch the original code for `apps/api/src/orgo/core/workflow/workflow-engine.serv

Then:
- Compare the current workflow engine implementation to the workflow/rule engine
```

```
- Implement or complete:
  - Parsing and execution of workflow definitions as described in the docs.
  - Safe, idempotent execution of workflows over Cases and Tasks, including dry-r
  - Proper logging, feature-flag hooks, and error handling using the standard res
- Output the **full final version** of `apps/api/src/orgo/core/workflow/workflow-
- Validate that the final code matches the workflow-related sections of the docs
```

text Copier le code

```
5. apps/api/src/orgo/core/workflow/escalation.service.ts

Fetch the original code for `apps/api/src/orgo/core/workflow/escalation.service.t

Then:
- Align the implementation with the escalation design in the docs:
  - Use `reactivity_deadline_at`, escalation levels, and organization profile SLA
  - Support periodic evaluation of overdue Tasks/Cases and triggering appropriate
- Implement/complete:
  - Clear public methods (e.g. `evaluateEscalationsForOrg`, `escalateTask`, etc.)
  - Integration points with AlertingService, notifications, and insights where de
  - Logging and feature-flag based enable/disable behaviour as per functional IDs
- Produce the **full final version** of `apps/api/src/orgo/core/workflow/escalatio
- Validate that the final code is fully aligned with the docs I provided (task li
```

text Copier le code

```
6. apps/api/src/orgo/core/email/email-router.service.ts

Fetch the original code for `apps/api/src/orgo/core/email/email-router.service.ts

Then:
- Review email routing requirements in the docs (mapping emails/threads to Cases/
- Update the router so that:
  - It accepts parsed email payloads and uses configuration/rules to decide whetho
  - It can hand off to specific domain modules or generic workflow definitions ba
  - It logs classification success/failure in a structured way and records proces
```

```
- Output the **full final version** of `apps/api/src/orgo/core/email/email-router
- Validate that the final code is consistent with the email, signal/ingest, workf
```

text Copier le code

```
7. apps/api/src/orgo/core/signals/signal-ingest.service.ts

Fetch the original code for `apps/api/src/orgo/core/signals/signal-ingest.service

Then:
- Compare the implementation to the “Signals & Ingestion” specification in the do
- Implement or refine:
  - Entry points for ingesting generic signals (email, webhooks, offline sync) int
  - Mapping signals to labels, profiles, workflows, and domain modules using conf
  - Multi-tenant safety, idempotency, and robust error logging with the standard
- Produce the **full final version** of `apps/api/src/orgo/core/signals/signal-ing
- Validate alignment with the ingest-related portions of the docs I provided (cor
```

text Copier le code

```
8. apps/api/src/orgo/core/notifications/notification.service.ts

Fetch the original code for `apps/api/src/orgo/core/notifications/notification.se

Then:
- Map current behaviour to the notification spec in the docs (channels: email, in
- Implement or complete:
  - A central NotificationService abstraction that can queue and send notificatio
  - Integration with task/case/workflow events and feature flags (e.g. enabling o
  - Proper persistence into the `notifications` table and safe multi-tenant acces
- Output the **full final version** of `apps/api/src/orgo/core/notifications/noti
- Validate that the final code is aligned with the notification and security/priv
```

text Copier le code

```
9. apps/api/src/orgo/core/logging/log.service.ts

Fetch the original code for `apps/api/src/orgo/core/logging/log.service.ts` from
```

```
Then:
- Review the logging/audit requirements from the docs (functional IDs, categories
- Update the service so that:
  - It exposes a clear, strongly-typed API for system/security/logging events.
  - It consistently attaches functional IDs, tenant IDs, and relevant metadata.
  - It supports privacy/visibility rules where specified (e.g. avoiding sensitive
- Produce the **full final version** of `apps/api/src/orgo/core/logging/log.servi
- Validate that the final logging behaviour matches the docs I provided (core logg
```

text Copier le code

```
10. apps/api/src/orgo/config/config.service.ts

Fetch the original code for `apps/api/src/orgo/config/config.service.ts` from the

Then:
- Align this service with the configuration model in the docs:
  - Environment-level config, module-level config (email, workflows, org_profiles
  - YAML-based config loading where specified, with validation.
- Implement or complete:
  - Helpers to fetch and merge config from base + environment + org scopes.
  - Types for key config slices that are referenced in other services (email, not
- Output the **full final version** of `apps/api/src/orgo/config/config.service.t
- Validate that the final implementation is consistent with the configuration and
```

text Copier le code

```
11. apps/api/src/orgo/config/org-profile.service.ts

Fetch the original code for `apps/api/src/orgo/config/org-profile.service.ts` fro

Then:
- Use the org profile and behaviour-profile docs (profile codes, behaviour knobs,
- Implement or refine:
  - Loading and resolving an organization's active profile (DB + YAML where appli
  - Convenience methods for reading SLA, visibility defaults, pattern sensitivity
  - Integration points to be used by TaskService, WorkflowEngine, Alerts/Insights
```

```
- Produce the **full final version** of `apps/api/src/orgo/config/org-profile.serv
- Validate that the final code matches the organization/profile-related docs I pr
```

text Copier le code

```
12. apps/api/src/orgo/config/feature-flag.service.ts

Fetch the original code for `apps/api/src/orgo/config/feature-flag.service.ts` fr

Then:
- Compare current behaviour to the feature-flag spec in the docs (scopes: global/
- Implement/complete:
  - Methods to evaluate a feature flag for a given functional ID, org, environmen
  - CRUD/helpers that align with the `feature_flags` schema and the REST endpoint
  - Clear typing and documentation so other services can easily consume feature-f
- Output the **full final version** of `apps/api/src/orgo/config/feature-flag.serv
- Validate that the final code is aligned with the feature-flag sections of the d
```

text Copier le code

```
13. apps/api/src/orgo/backbone/organizations/organization.service.ts

Fetch the original code for `apps/api/src/orgo/backbone/organizations/organizatio

Then:
- Align the implementation with the backbone/organization spec:
  - Creation, update, and retrieval of organizations.
  - Status changes (active/suspended/archived) and their implications.
  - Initial profile/feature-flag setup for new organizations as described in the
- Ensure:
  - Strong multi-tenant semantics and safe lookups.
  - Integration with OrgProfileService, feature flags, and insights where needed.
- Produce the **full final version** of `apps/api/src/orgo/backbone/organizations/
- Validate that the final code is consistent with the backbone organizations/RBAC
```

text Copier le code

```
14. apps/api/src/orgo/insights/reports/reports.service.ts

Fetch the original code for `apps/api/src/orgo/insights/reports/reports.service.t

Then:
- Use the Insights/analytics spec (star schema, fact/dim tables, report types lik
- Implement or complete:
  - Methods to query the insights warehouse (Postgres schema or other configured
  - Support for filter parameters (org, time window, domain/type, profile code) e
  - Proper performance considerations and use of any caching layer defined in the
- Output the **full final version** of `apps/api/src/orgo/insights/reports/report
- Validate that the final reporting behaviour is aligned with the Insights and cy
```

text Copier le code

```
15. apps/api/src/orgo/insights/insights-cache-warmup.service.ts

Fetch the original code for `apps/api/src/orgo/insights/insights-cache-warmup.ser

Then:
- Review the caching and “warm dashboard” requirements in the Insights config doc
- Implement or refine:
  - Logic to precompute and cache Insights dashboards for key organizations/time
  - Config-driven scheduling/selection of which dashboards to warm.
  - Robust logging and error handling for cache warm-up runs.
- Produce the **full final version** of `apps/api/src/orgo/insights/insights-cach
- Validate that the final code matches the Insights caching and performance secti
```

text Copier le code

```
16. apps/api/src/orgo/insights/pattern-detection.service.ts

Fetch the original code for `apps/api/src/orgo/insights/pattern-detection.service

Then:
- Use the pattern-detection and cyclic overview docs (weekly/monthly/yearly patte
- Implement or complete:
  - Scans over the insights fact tables (or operational tables, if specified) to
```

```
  - Respect of per-profile thresholds and window sizes.
  - Emission of results in a form usable by cyclic review (e.g. creating or updat
- Output the **full final version** of `apps/api/src/orgo/insights/pattern-detect
- Validate that the final behaviour is aligned with the pattern/Insights sections
```

text Copier le code

```
17. apps/api/src/orgo/core/offline/sync.service.ts

Fetch the original code for `apps/api/src/orgo/core/offline/sync.service.ts` from

Then:
- Review the offline/sync design in the docs (offline nodes, sync journals, confl
- Implement or refine:
  - APIs to sync changes between offline nodes and the central system, respecting
  - Handling of `offline_nodes`, `sync_*` tables as per the schema (upload/downlo
  - Clear separation between ONLINE and OFFLINE modes, consistent with DatabaseSe
- Produce the **full final version** of `apps/api/src/orgo/core/offline/sync.serv
- Validate that the final code is fully aligned with the offline/sync sections of
```

which files you have in memory

Thought for 34 seconds >

Here are **all the files you uploaded in this conversation** — the ones I have available "in memory" and can search/parse:

Original 00–08 backend & frontend architecture dumps

1. **00_root_and_monorepo_meta.txt**
2. **01_api_bootstrap_and_persistence.txt**
3. **02_api_backbone_identity_and_persons.txt**
4. **03_api_backbone_organizations_and_rbac.txt**
5. **04_api_config_profiles_and_feature_flags.txt**

6. 05_api_core_and_supporting_core.txt
7. 06_api_domain_insights_security_and_orgo_module.txt
8. 07_web_app_shell_pages_and_store.txt
9. 08_web_orgo_frontend_types_screens_and_shared_ui.txt
