perplexity

Overview

This micro-service specification turns the workflow in Goal.pdf into an independently deployable "Workflow Service" that powers two parallel streams:

- Internal product development for a small web team (Product Designer, FE, BE, QA).
- A public-facing coding academy that mimics the annotator submit → review → approve loop[1].

The service exposes a JSON/REST API, publishes domain events, and persists state in PostgreSQL. All role checks follow the RBAC matrix defined in the original blueprint[1].

Service Boundaries

Micro-service	Core Responsibility	Tech/Storage
AuthN/AuthZ	JWT issuance, role claims, impersonation for Super Admin	Clerk/Auth0 + Redis
Task Service	CRUD epics, tickets, academy exercises; Kanban ordering	Node.js (Fastify), PostgreSQL
Review Service	Human/auto reviews, outcomes, comments	Node.js, PostgreSQL
CI Orchestrator	Webhooks from GitHub → test runners → result callbacks	Go, RabbitMQ
Notification	Email, Slack, in-app toasts on task/review events	Node.js, Postmark
Metrics	Prometheus scrapers → Grafana dashboards	Go, TimescaleDB
API Gateway	Single entry, rate-limit, CORS, OpenAPI docs	NGINX, Kong

Workflow, Review and CI logic live in the Task & Review services; everything else is orthogonal infrastructure.

Domain Model (Workflow DB)

```
model User {
  id         String   @id @default(cuid())
  email      String   @unique
  role       Role                    // SUPER_ADMIN, DESIGNER, FE_DEV, BE_DEV, QA, STUDE
  xp         Int      @default(0)
  tasks      Task[]   @relation("Assignee")
  reviews    Review[] @relation("Reviewer")
}

model Task {
  id         String   @id @default(cuid())
  title      String
  type       TaskType // FEATURE, BUG, RESEARCH, ACADEMY_EXERCISE
  status     Status   // TODO, IN_PROGRESS, SUBMITTED, APPROVED, REJECTED
```

```
  repoUrl    String?
  assignee   User?    @relation("Assignee", fields:[assigneeId], references:[id])
  assigneeId String?
  reviews    Review[]
  createdAt  DateTime @default(now())
  updatedAt  DateTime @updatedAt
}

model Review {
  id         String   @id @default(cuid())
  outcome    ReviewOutcome // APPROVED, REJECTED
  comment    String?
  task       Task     @relation(fields:[taskId], references:[id])
  taskId     String
  reviewer   User     @relation("Reviewer", fields:[reviewerId], references:[id])
  reviewerId String
  createdAt  DateTime @default(now())
}
```

These tables mirror the PDF's RBAC and review life-cycle[1].

Core API (Task & Review Service)

Method	Route	Description	Roles
`POST /tasks`	Create task/epic/exercise	**Designer**, **BE_DEV**, **FE_DEV**, **SUPER_ADMIN**	
`PATCH /tasks/:id/status`	Move card on Kanban board	Assignee or **SUPER_ADMIN**	
`POST /tasks/:id/submit`	Mark coding work ready for review; triggers `Task.SUBMITTED` event	Any assignee	
`POST /reviews`	Human review with outcome + comment	**QA**, **REVIEWER**, **SUPER_ADMIN**	
`GET /queue/reviewer`	Paginated list of pending reviews	**QA**, **REVIEWER**	
`POST /academy/enroll`	Student enrolls and receives starter tasks	**STUDENT**	
`POST /admin/impersonate`	Generate token for any user (audit logged)	**SUPER_ADMIN**	

OpenAPI 3 schema is generated at build time and served at `/docs`.

Event Contract (RabbitMQ)

```
Task.CREATED
Task.SUBMITTED
Task.APPROVED
Task.REJECTED
Review.CREATED
CI.RESULT       // { taskId, passed: boolean, logsUrl }
```

The CI Orchestrator publishes `CI.RESULT`; Review Service listens and auto-creates a Review with outcome *REJECTED* if tests fail, otherwise sets status to *SUBMITTED*.

Workflow Scenarios

Internal Feature Development

1. Designer `POST /tasks` (type=FEATURE).
2. FE & BE claim subtasks → status **IN_PROGRESS**.
3. Push to GitHub triggers CI Orchestrator → results returned as events.
4. Dev calls `.../submit` once unit tests pass.
5. QA picks from `/queue/reviewer`, sets `outcome=APPROVED/REJECTED`.
6. Approved → auto-deploy pipeline; Metrics Service logs cycle time for dashboards[1].

Coding Academy Loop

1. Student enrolls → Task Service seeds starter exercise (status **TODO**).
2. Student forks repo, codes, pushes → CI runs linter/unit tests.
3. Passing CI = webhook → `submit` endpoint.
4. Reviewer grabs task, leaves feedback.
5. On **APPROVED**, XP +10; trigger next harder exercise. Super Admin may override at any point[1].

Security & Observability

- JWT claims (`role`, `sub`) checked via middleware; Super Admin bypass through impersonation endpoint.
- Loki + Grafana dashboards expose: average review turnaround, CI pass-rate, student progression funnels[1].
- Audit table records all `task.status` transitions and impersonations.

Deployment Topology

```
graph TD
subgraph k8s-cluster
    gateway(API Gateway) --> task(Task Svc)
    gateway --> review(Review Svc)
    gateway --> auth(Auth Svc)
    ci(CI Orchestrator) -->|CI.RESULT| review
    review -->|Task.APPROVED| task
    task -->|Task.SUBMITTED| review
    notif(Notification)
    metrics(Prom/Grafana)
end
```

- Each service ships as a Docker image, deployed via Helm.
- GitHub Actions pushes images → ArgoCD syncs environments.

Quick-start for Developers

```
# Docker compose excerpt
version: "3.9"
services:
  task:
    build: ./services/task
    ports: ["4000:4000"]
    env_file: .env
  review:
    build: ./services/review
    ports: ["4001:4001"]
  postgres:
    image: postgres:16
    volumes: ["pg:/var/lib/postgresql/data"]
  rabbit:
    image: rabbitmq:3-management
volumes: { pg: {} }
```

`pnpm i && pnpm dev` in each service folder spins up Hot-Reloaded Fastify servers with Prisma migrations.

Roadmap (6 Sprints)

Sprint	Output
1	Auth service, RBAC tables, Super Admin impersonation
2	Task micro-service with Kanban endpoints
3	Review service + event bus wiring
4	CI Orchestrator, GitHub webhook integration
5	Academy enrollment, XP gates, notification hooks
6	Grafana dashboards, penetration test, GA launch

By encapsulating the PDF's workflow logic in this dedicated micro-service, the platform gains clean boundaries, event-driven extensibility, and a single API for both internal product development and the academy track—all governed by Super Admin oversight as mandated in the source document[1].

⁂

1. Goal.pdf