OCH STUDENT MANAGEMENT PORTAL (SMP) TECHNICAL SPECIFICATIONS DOCUMENT

Technical Specification Document (Developer-Ready)

Nov 8th 2025

A) Foundation Modules

User & Identity (Accounts, Roles, SSO, MFA)

Technical Specification (Developer-Ready)

1) Module Overview

Centralized identity, authentication, and authorization for all OCH SMP services. Provides:

- Account lifecycle (signup → verification → activation → deactivation → erasure).
- AuthN: email+password, **passwordless magic link/OTP**, **MFA (TOTP/SMS)**, and **SSO (OIDC/SAML)**.
- AuthZ: **RBAC/ABAC** with organization (Sponsor/Employer) scopes and cohort/track context.
- Session management, device trust, API keys (service/partner), secrets rotation, audit hooks.

Value: single source of truth for users, organizations, roles, and permissions; secure-by-default access to all modules.

2) Functional Objectives

1. Provide secure, standards-based login (OIDC/OAuth2, PKCE, short-TTL JWT + refresh).
2. Enforce **MFA** and risk-based challenges; support **passwordless** flows.
3. Manage **RBAC** roles + **ABAC** policies (cohort_id, track, org_id).
4. Support **organizations** (Sponsors/Employers) with delegated admin.
5. Issue/rotate **API keys** and **webhook signing secrets** for partners.
6. Full **audit trail**, consent checks, and privacy operations (export/erasure) aligned to Botswana DPA/GDPR.
7. Provide SDKs/middlewares for other services to validate tokens & enforce policy.

3) User Types & Permissions

Roles (global/base)

- **Admin** – full platform admin; manage roles/policies, tenants, secrets.
- **Program Director** – manage programs/cohorts/tracks; view analytics; assign mentors.
- **Mentor** – access assigned mentees; create notes; review portfolios; limited analytics.
- **Student** – access personal modules (profiling, learning, portfolio, mentorship).
- **Finance** – access billing/revenue, refunds, sponsorship wallets; no student PII beyond billing.

- **Sponsor/Employer Admin** – manage sponsored users, view permitted profiles per consent.
- **Analyst** – analytics read with RLS/CLS; no PII without scope.

ABAC Attributes

`cohort_id`, `track_key`, `org_id`, `consent_scopes[]`, `entitlements[]`, `country`, `tz`, `risk_level`.

Policy examples

- Mentor can `READ` mentee profiling **only if** `match_exists(user_id, mentor_id)` **and** `consent_scopes` includes `"share_with_mentor"`.
- Director can `LIST` cohort portfolios if `user.cohort_id == request.cohort_id`.
- Finance can `READ` invoices if `invoice.org_id == user.org_id` **or** role in {Admin, Finance}.

4) Core Functional Requirements

4.1 Account Lifecycle

- **Create:** email (or SSO), name, country, timezone; send verify link; optional invite flow.
- **Verify/Activate:** email link or OTP; set MFA policy.
- **Profile:** minimal PII (name, email, country, tz, preferred language); student may add optional fields.
- **Suspend/Deactivate:** soft-lock access; retain audit; notify owning org/cohort.
- **Delete/Erasure:** privacy workflow triggers downstream purge signals; retain legal ledger where required (billing).

4.2 Authentication

- **Email + Password** (Argon2id hashing, per-user salt, pepper via KMS).
- **Passwordless:** magic link (one-time, 10 min); OTP (email/SMS).
- **MFA:** TOTP (RFC 6238) primary; SMS/backup codes secondary; policy per role (e.g., Finance/Admin = mandatory).
- **SSO:** OIDC (Google/Microsoft), SAML 2.0 for enterprises; Just-In-Time (JIT) provisioning with role mapping.
- **Risk signals:** new device, geo-velocity, TOR/VPN list → step-up MFA.

4.3 Authorization

- **RBAC:** role assignments at **global** and **org** scope; cohort/track role grants.
- **ABAC:** policy engine evaluates attributes + environment (time, IP reputation) for every request.
- **Entitlements check:** gate features (from Billing) at middleware.

4.4 Sessions & Tokens

- **Access token:** JWT (15 min), audience per service, signed with rotating keys (JWKS).
- **Refresh token:** opaque, 30 days, rotating; bound to device; stored httpOnly, Secure, SameSite=Lax.
- **Logout/Revocation:** blacklist (hash), device-scoped revocation; global sign-out.

4.5 Organizations & Tenancy

- **Organization (org)** entity for Sponsors/Employers/Partners.
- **Delegated admin:** invite members, assign roles, manage sponsored students.
- **Scopes:** data segregation by `org_id`; employer view limited by consent.

4.6 API Keys & Integrations

- **Service accounts** (machine-to-machine) with **scoped API keys**; least privilege.
- **Partner apps**: OAuth2 Client Credentials or Authorization Code with PKCE; rate-limited; webhook signing (HMAC).

4.7 Consent & Privacy Hooks

- Capture user consent scopes (profiling visibility, public portfolio, employer share, marketing).
- All token issuance embeds `consent_scopes`. Middleware denies access when scope missing.

4.8 Audit & Security Ops

- Audit all auth events: login, MFA, SSO mapping, role changes, token revocations, API key usage.
- Security alerts: brute-force, credential stuffing, anomalous geography.

5) Data Model (PostgreSQL)

```
users(
  id UUID PK, email CITEXT unique, email_verified BOOL, password_hash TEXT
null,
  name TEXT, country CHAR(2), timezone TEXT, language TEXT,
  status ENUM[active,suspended,deactivated], created_at, updated_at,
last_login_at
)

user_identities(           -- external IdPs / SSO links
  id UUID PK, user_id UUID FK, provider
ENUM[google,microsoft,saml,apple,local],
  provider_sub TEXT, metadata JSONB, linked_at
)
```

```
mfa_factors(
  id UUID PK, user_id UUID FK, type ENUM[totp,sms,backup_codes],
  secret_encrypted TEXT, phone_e164 TEXT, enabled BOOL, created_at
)

roles( id UUID PK, key TEXT unique, description TEXT )
user_roles( user_id UUID, role_id UUID, scope ENUM[global,org,cohort,track],
scope_ref UUID null, PRIMARY KEY(user_id, role_id, scope, scope_ref) )

policies( id UUID PK, name TEXT, effect ENUM[allow,deny], resource TEXT,
action TEXT[], condition JSONB, version INT, active BOOL )

orgs( id UUID PK, name TEXT, type ENUM[sponsor,employer,partner], country
CHAR(2), status ENUM[active,inactive], created_at )
org_members( org_id UUID, user_id UUID, role_id UUID, PRIMARY KEY(org_id,
user_id, role_id) )

sessions(
  id UUID PK, user_id UUID, device_fingerprint TEXT, ip INET, ua TEXT,
  refresh_token_hash TEXT, expires_at TIMESTAMP, created_at, revoked_at
)

api_keys(
  id UUID PK, owner_type ENUM[user,org,service], owner_id UUID, key_hash
TEXT,
  scopes TEXT[], rate_limit_per_min INT, created_at, last_used_at, revoked_at
)

consents( id UUID PK, user_id UUID, scope TEXT, granted BOOL, granted_at,
expires_at )

audit_logs( id UUID PK, actor_id UUID, entity TEXT, entity_id UUID, action
TEXT, meta JSONB, at TIMESTAMP )
```

Security notes

- Email as **CITEXT** for case-insensitive uniqueness.
- All secrets encrypted at rest (KMS).
- Refresh tokens & API keys stored **hashed** (Argon2id).
- Row Level Security (RLS) on org-scoped tables.

6) Process Flows

Signup (email + passwordless)

1. POST `/auth/signup` → create `users` (status=active if invite; else pending_verify).
2. Send magic link **or** OTP.
3. Verify → optional MFA enrollment → issue tokens (JWT + refresh).
4. If invite includes `cohort_id`/`org_id`, auto-attach roles.

Login + MFA

1. POST `/auth/login` (email+password or passwordless link).
2. Risk engine evaluates → if required, prompt MFA (TOTP/SMS/backup).
3. Issue JWT (15m) + refresh (30d), set httpOnly cookie.
4. Record `sessions` + audit.

SSO (OIDC/SAML)

1. Redirect to IdP → return with code → exchange via PKCE.
2. Map IdP groups/claims → roles (config rules).
3. If first login, create user (JIT) and link identity.
4. Enforce MFA policy if IdP MFA absent and role requires.

Consent-gated Access

- Middleware checks `consent_scopes` for resource; deny with 403 + `required_scope` hint.

Logout / Revoke

- Invalidate refresh token (set `revoked_at`, add to denylist); broadcast to services.

7) Integrations & APIs

REST (prefix `/api/v1/auth`)

- `POST /signup` — create account (invite optional).
- `POST /login` — password or code (passwordless).
- `POST /login/magic-link` — send link.
- `POST /mfa/enroll` (TOTP setup) / `POST /mfa/verify` / `POST /mfa/disable`.
- `POST /token/refresh` — rotate refresh token.
- `POST /logout` — revoke session.
- `GET /me` — profile + roles + consents.
- `POST /consents` — update consent scopes.
- `POST /password/reset/request` / `POST /password/reset/confirm`.

Admin/Org

- `POST /roles` / `GET /roles`
- `POST /users/{id}/roles` (with scope)
- `POST /orgs` / `POST /orgs/{id}/members`
- `POST /api-keys` / `DELETE /api-keys/{id}`
- `GET /audit?actor=&entity=&range=...`

OpenID Connect

- `/.well-known/openid-configuration`
- `/oauth/authorize`, `/oauth/token`, `/oauth/jwks.json`
- `/oauth/introspect` (service-to-service)

Webhook signatures

- HMAC-SHA256 with per-consumer secrets; include `t` timestamp and replay window (5 min).

Example: `/me` response

```
{
  "user": {"id":"UUID","email":"martin@och.africa","name":"Martin"},
  "roles":[{"role":"program_director","scope":"cohort","scope_ref":"UUID-
COHORT-JAN26"}],
  "consent_scopes":["share_with_mentor","public_portfolio:false"],
  "entitlements":["cohort_seat:JAN26","module_access:profiling"]
}
```

8) UI/UX Requirements

- **Auth screens:** accessible forms; clear error messages; passwordless first, password optional.
- **MFA setup:** QR for TOTP, backup codes (print/download).
- **SSO buttons:** Google/Microsoft; IdP-initiated SSO supported.
- **Sessions & Devices:** "Where you're logged in" with revoke button.
- **Org Admin:** invite users, assign roles, set SSO domain rules.
- **Consent Center:** toggle scopes with plain-language explanations and effective dates.
- **A11y:** WCAG 2.1 AA; keyboard-only flows; screen-reader labels.

9) Security & Compliance

- **Password policy:** Argon2id; min length 12 if used; breach check (HaveIBeenPwned k-anon).
- **MFA required** for Admin, Finance, Director; recommended for others.
- **JWT hardening:** `aud`, `iss`, `iat`, `nbf`, `exp`; clock skew ±60s; key rotation every 90 days.
- **CSP/Headers:** HSTS, CSP, X-Frame-Options, X-Content-Type-Options, Referrer-Policy.
- **Botswana DPA/GDPR:** explicit consent, purpose limitation, data minimization, SAR/erasure endpoints, records of processing.
- **Logging:** no secrets/PII in logs; structured logs with request ID; trace propagation.
- **Backups/DR:** encrypted daily backups; 30-day retention; quarterly restore test.

10) Technical Dependencies

- **Backend:** NestJS (Node) or FastAPI (Python); OIDC provider library; JOSE for JWT.

- **Store:** PostgreSQL 15; Redis for sessions/rate-limits/denylist.
- **Secrets:** KMS/Vault.
- **SMS/Email:** Twilio/MessageBird; SendGrid/SES.
- **Observability:** OpenTelemetry, Prometheus, Grafana, Sentry.
- **Infra:** API Gateway (NGINX/Cloud LB), WAF, CDN for static.

11) Error Handling & Responses

Case	Code	Response
Invalid credentials	401	`{ "error":"invalid_login" }`
MFA required	401	`{ "error":"mfa_required", "factors":["totp","sms"] }`
Magic link expired	410	`{ "error":"link_expired" }`
Scope denied	403	`{ "error":"forbidden", "required_scope":"share_with_mentor" }`
Token expired	401	`{ "error":"token_expired" }`
Rate limit	429	`{ "error":"rate_limited", "retry_after": 30 }`

Idempotency for all auth mutation endpoints via header `Idempotency-Key`.

12) Test Plan (high level)

- **Unit:** password hashing, TOTP verification, JWT claims, policy engine decisions.
- **Integration:** SSO (OIDC/SAML) with mocked IdP; email/SMS adapters; token rotation.
- **E2E:** signup→verify→MFA→SSO mapping; consent-gated resource; role escalation with audit.
- **Security:** brute-force protection, CSRF where applicable, SSRF denial in SAML metadata, JWT kid-swap test.

13) Future Enhancements

- **Passkeys (WebAuthn)** for phishing-resistant login.
- **Continuous session risk scoring** (impossible travel, device posture).
- **Just-In-Time cohort enrollment** based on SSO group membership.
- **Attribute authorities** (HRIS/Student IS) for automated role provisioning.
- **PII tokenization** service for analytics.

Program & Cohort Management Module

Technical Specification (Developer-Ready)

1) Module Overview

The **Program & Cohort Management Module (PCM)** serves as the **core orchestration layer** for OCH SMP.
It defines and manages all academic and training structures — **programs, tracks, specializations, cohorts, intakes, sessions, and calendars** — while ensuring clear relationships between students, mentors, directors, and sponsors.

It also integrates with **Billing, Scheduling, Learning (LRS), Mentorship, and Analytics**, forming the foundation for data consistency across the platform.

Value: Single source of truth for what is running, who is enrolled, when, and under what rules (pricing, schedule, structure).

2) Functional Objectives

1. Define **programs**, **tracks**, and **specializations** with metadata, learning paths, and target roles.
2. Create and manage **cohorts/intakes**, including enrollment caps, schedules, and assigned directors/mentors.
3. Manage **enrollment and seat allocation**, including scholarship/sponsor seats.
4. Provide **calendar and milestone tracking** (orientation, mentorship, submissions, reviews, closure).
5. Expose structured APIs for **Learning, Mentorship, Billing, Gamification**, and **Analytics** modules.
6. Serve as the **policy engine** for program eligibility, duration, and completion rules.
7. Maintain full **audit logs**, **consent hooks**, and **notification triggers**.

3) Core Entities & Hierarchy

```
Program
 ├── Tracks (Builder, Leader, Entrepreneur, Researcher, Educator)
 │    ├── Specializations (optional)
 │    │     ├── Missions (mapped from Registry)
 │    │
 │    └── Cohorts (Jan 2026, May 2026, Sep 2026)
 │          ├── Calendar (start/end, milestones)
 │          ├── Enrollment (students, mentors, sponsors)
 │          └── Metrics (progress, readiness, completion)
 └── Associated Assets (pricing, documents, resources)
```

4) User Roles & Permissions

Role	Create	View	Edit	Approve	Archive/Delete
Admin	All programs, cohorts, seats	All	All	All	All
Program Director	Programs, cohorts (owned tracks)	All within program	Seats, schedules, mentors	Approve enrollments	Archive cohort (after closure)
Mentor	None	Assigned cohorts & mentees	Attendance, milestones	Session reports	—
Student	Self-enroll (if open)	Own program/cohort	Update profile data	—	—
Sponsor/Org Admin	Enroll sponsored seats	Sponsored cohorts	View progress	—	—
Finance	Products, prices, seat caps	Financial dashboards	—	Approve refunds (via Billing)	—

5) Core Functional Requirements

5.1 Program Management

- **Description:** Define programs, durations, tracks, learning goals, pricing models.
- **User Story:** As Admin, I can define "Cybersecurity Leadership Program" with five tracks.
- **Acceptance Criteria:**
  - Metadata: name, description, duration (months), category (mentorship, executive, technical), default price, currency, outcomes.
  - Link to Missions/Competency registry.
  - Define default structure (modules, milestones).
- **I/O:** `program_id`, JSON schema for structure.

5.2 Track & Specialization Definition

- **Description:** Sub-structure defining focused learning paths or professional roles.
- **User Story:** As Director, I define "Cyber Leaders Track" with 3 milestones and 6 modules.
- **Acceptance Criteria:**
  - Define tracks within a program, optional specialization paths.
  - Attach competencies and missions (via registry).
  - Assign directors/mentors per track.

- **Dependencies:** Missions Registry, User & Identity.
- **I/O:** `track_id`, `specialization_id`.

5.3 Cohort Creation & Management

- **Description:** Operational layer — actual running group of learners.
- **User Story:** As Director, I create "Jan 2026 Cohort" for Cyber Builders.
- **Acceptance Criteria:**
  - Attributes: `program_id`, `track_id`, name, start_date, end_date, mode (onsite, hybrid, virtual), seat_cap, mentor_ratio.
  - Status lifecycle: `draft → active → running → closing → closed`.
  - Assign mentors, coordinators, sponsors.
  - Seat pool management (paid, scholarship, sponsored).
  - Calendar template (orientation, sessions, portfolio submission, closure).
  - Integrated with Billing to verify entitlements.
- **I/O:** `cohort_id`, `calendar_id`, `seat_pool`.

5.4 Enrollment & Seat Allocation

- **Description:** Register users into cohorts, control seat utilization.
- **User Story:** As Student, I enroll in a cohort; as Sponsor, I allocate a paid seat.
- **Acceptance Criteria:**
  - Enrollment methods: self-enroll (open), invite, sponsor assign, director assign.
  - Validation: check entitlements (from Billing) or seat availability.
  - Waitlist management (FIFO, auto-promotion).
  - Status: `pending_payment`, `active`, `suspended`, `withdrawn`, `completed`.
  - Multi-cohort prevention (same program).
  - Auto-trigger notifications on enrollment and start.
- **Dependencies:** Billing, Notifications, Consent.
- **I/O:** `enrollment_id`, `seat_status`.

5.5 Calendar & Milestones

- **Description:** Cohort schedule of deliverables and key events.
- **User Story:** As Director, I set calendar milestones (orientation, assessments, reviews).
- **Acceptance Criteria:**
  - Create calendar from template or scratch.
  - Event types: `orientation`, `mentorship`, `project_review`, `holiday`, `submission`, `closure`.
  - Time-zone aware; syncs with Scheduling & Notifications.
  - Completion tracking for each milestone.
- **Dependencies:** Scheduling Hub, Notifications.
- **I/O:** `calendar_id`, `milestone_id`.

5.6 Mentor Assignment

- **Description:** Assign mentors to cohorts or tracks.
- **User Story:** As Director, I assign 3 mentors to Cyber Builders (Jan 2026).
- **Acceptance Criteria:**
  - Auto-match from Mentorship registry by availability and skillset.
  - Assign primary/backup mentors.
  - Role in cohort dashboard: `mentor`, `support_mentor`, `guest`.
- **Dependencies:** Mentorship Management, Profiling.
- **I/O:** `mentor_assignment_id`.

5.7 Program Rules & Completion

- **Description:** Define program success metrics and auto-graduation logic.
- **User Story:** As Director, I define 80% completion and portfolio approval as graduation criteria.
- **Acceptance Criteria:**
  - Rules configurable per program (attendance %, portfolio approval, feedback score, payment complete).
  - Auto-update student status to `completed` or `incomplete`.
  - Notify director, trigger certificate generation.
- **Dependencies:** Portfolio, Learning, Billing, Gamification.
- **I/O:** `completion_rule_id`, `certificate_trigger`.

5.8 Reports & Dashboards

- **Description:** Real-time visibility into cohort status, seats, completion rates.
- **User Story:** As Director, I view seat utilization and progress summaries.
- **Acceptance Criteria:**
  - Cohort dashboard: enrollments, seat utilization, mentor assignments, readiness delta, completion %, payments.
  - Export CSV/JSON to Analytics layer.
- **Dependencies:** Analytics Layer, Gamification.
- **I/O:** `cohort_report_uri`.

6) Data Model (PostgreSQL)

Table	Description
`programs`	id UUID PK, name TEXT, category ENUM[technical, leadership, mentorship], description TEXT, duration_months INT, default_price NUMERIC, currency TEXT, status ENUM[active,inactive], created_at, updated_at
`tracks`	id UUID PK, program_id FK, name TEXT, key TEXT, description TEXT, competencies JSONB, director_id FK, created_at, updated_at

Table	Description
specializations	id UUID PK, track_id FK, name TEXT, description TEXT, missions JSONB, duration_weeks INT
cohorts	id UUID PK, track_id FK, name TEXT, start_date DATE, end_date DATE, mode ENUM[onsite,virtual,hybrid], seat_cap INT, mentor_ratio FLOAT, calendar_id FK, status ENUM[draft,active,running,closing,closed], created_at, updated_at
enrollments	id UUID PK, cohort_id FK, user_id FK, org_id FK nullable, enrollment_type ENUM[self,sponsor,invite], seat_type ENUM[paid,scholarship,sponsored], payment_status ENUM[pending,paid,waived], status ENUM[active,withdrawn,completed], joined_at, completed_at
calendar_events	id UUID PK, cohort_id FK, type ENUM[orientation,session,submission,holiday,closure], title TEXT, description TEXT, start_ts TIMESTAMP, end_ts TIMESTAMP, location TEXT, link TEXT, status ENUM[scheduled,done,cancelled]
mentor_assignments	id UUID PK, cohort_id FK, mentor_id FK, role ENUM[primary,support,guest], assigned_at, active BOOL
program_rules	id UUID PK, program_id FK, rule JSONB (criteria, thresholds, dependencies), version INT, active BOOL
certificates	id UUID PK, enrollment_id FK, file_uri TEXT, issued_at TIMESTAMP
audit_logs	id UUID PK, actor_id UUID, entity TEXT, entity_id UUID, action TEXT, meta JSONB, at TIMESTAMP

7) Process Flow (Happy Path)

Program Setup → Cohort Launch → Enrollment → Execution → Closure

1. **Create Program** → Define tracks/specializations → Assign directors/mentors.
2. **Create Cohort** → Set calendar & seat cap → Open enrollment.
3. **Enrollment Phase**
    - Student/Sponsor enroll → Billing validates payment/entitlement.
    - On success → Enrollment activated → Notifications triggered.
4. **Execution Phase**
    - Cohort milestones tracked via calendar → sessions & projects linked to milestones.
    - Mentorship, Learning, Portfolio updates feed cohort dashboards.
5. **Closure Phase**
    - Evaluate rules → auto-compute completions.
    - Issue certificates → Archive cohort → Trigger Analytics pipeline.

8) Integrations & APIs

Internal REST (prefix `/api/v1/programs`)

Endpoint	Method	Description
`/programs`	POST	Create program
`/programs`	GET	List programs
`/programs/{id}`	GET/PUT	View or update program
`/tracks`	POST/GET	Create/list tracks
`/cohorts`	POST/GET	Create/list cohorts
`/cohorts/{id}/calendar`	POST/GET	Manage cohort calendar
`/cohorts/{id}/enrollments`	POST/GET	Enroll student or list cohort members
`/cohorts/{id}/mentors`	POST/GET	Assign mentors
`/rules`	POST/GET	Define completion rules
`/certificates/{id}`	GET	Download issued certificate

Events (emit via message bus):

- `program.created`
- `cohort.opened`
- `cohort.closed`
- `enrollment.created`
- `enrollment.completed`
- `certificate.issued`

Webhook consumers:

- **Billing:** listen to `cohort.opened`, `enrollment.created`.
- **Analytics:** subscribe to `cohort.closed`.
- **Notifications:** handle `calendar_event.soon`, `certificate.issued`.

9) UI/UX Requirements

Director View

- Dashboard: Active cohorts, seats left, completion rate.
- Forms: create/edit programs, tracks, cohorts.
- Calendar drag-drop with milestone tags.
- Mentor assignment & tracking view.
- Report exports.

Student View

- My Cohort card: mentor list, key dates, completion %, certificate status.
- Enrollment journey: waitlist → payment → confirmation.
- Calendar integration (Google/Outlook).

Admin View

- Global overview of programs, active cohorts, seat utilization by track, completion % across programs.
- Rules configuration (UI for JSON rules).
- Archival dashboard.

10) Security & Compliance

- **RLS:** Student sees only own enrollment/cohort; mentors only assigned cohorts.
- **Access control:** Directors limited to owned programs/tracks.
- **Data retention:** Cohort data retained 5 years post-closure; personal data per DPA retention.
- **Audit:** All changes logged (`actor_id`, `action`, `before/after`).
- **Encryption:** all PII and certificates in AES-256 at rest.
- **Backups:** daily incremental, weekly full; tested quarterly.

11) Technical Dependencies

Component	Tech Stack
Backend	FastAPI / NestJS with PostgreSQL ORM
DB	PostgreSQL 15
Queue	Redis + BullMQ / Celery
Calendar Integration	Google Calendar / Outlook API
Notifications	Twilio (SMS), SendGrid (Email), Firebase (In-App)
File Storage	AWS S3 (certificates, templates)
Auth	Identity Service (JWT middleware)
Analytics	Event stream → Warehouse (dbt models)
Observability	OpenTelemetry, Prometheus, Sentry

12) Error Handling & Exceptions

Scenario	Handling
Seat cap reached	409 Conflict + “Join waitlist”
Duplicate enrollment	409 Conflict + enrollment_id
Calendar conflict	Suggest alternate time; flag in UI
Mentor double-booked	Reject; notify Director

Scenario	Handling
Payment pending > 7 days	Auto-expire enrollment; notify student
Rule eval failure	Log + retry + notify Director

13) Future Enhancements

1. **Auto-scheduling engine** — generate calendars from mentor availability and time zones.
2. **Smart seat allocation** — dynamic seat reallocation based on payment/withdrawals.
3. **Multi-track linking** — allow cross-track modules for specialization.
4. **Internship/placement integration** — link top graduates to employers (via Employer Portal).
5. **Adaptive milestones** — personalize timelines based on learner pace.

API Gateway & Integration Layer (AGIL)

Technical Specification (Developer-Ready)

1) Module Overview

Central ingress/egress plane for all OCH SMP services. Provides:

- **North-south** traffic management (client→platform) and **east-west** service mediation (service→service, partner→platform).
- **Authentication/Authorization enforcement**, **rate limiting/quotas**, **versioning**, **request/response transformation**, **schema contracts**, **webhooks**, **caching**, and **observability**.
- One place to onboard partners (employers/sponsors, learning providers), issue **API keys/OAuth clients**, and manage **event subscriptions**.

Value: consistent, secure, auditable interface; decouples clients/partners from internal service churn; accelerates integrations.

2) Objectives

1. Terminate TLS, enforce **AuthN/Z** (JWT, API keys, OAuth2/OIDC), and apply **RBAC/ABAC** at the edge.
2. Provide **global and per-consumer** rate limits, quotas, and burst control.
3. Support **versioned APIs** (v1/v2) with **deprecation policy** and compatibility shims.
4. Offer **request shaping** (validation, coercion), **field-level filtering**, **pagination** (cursor-based), **idempotency**, and **caching**.
5. Manage **webhooks** (subscriptions, signing, retries, DLQ) and **partner event streams**.
6. Deliver **analytics/usage metering**, **audit**, and **error handling standards** across all APIs.
7. Expose **developer portal** (keys, docs, try-it, sandbox) and **SDKs** (TS/JS, Python).

3) Responsibilities & Boundaries

- **In scope:** Edge security, discovery/routing, policy, contract validation, transformation, caching, webhooks, partner onboarding, usage analytics.
- **Out of scope:** Business logic (lives in domain services), BI analytics (lives in Analytics Layer).

4) Roles & Permissions

Role	Capabilities
Gateway Admin	Configure routes, plugins/policies, certificates, keys rotation, quotas, portals

Role	Capabilities
API Product Owner	Publish OpenAPI, set version lifecycle, define usage plans
Partner Admin (Org)	Manage their apps/keys, rotate secrets, subscribe to webhooks
Developer (internal)	Register service, publish spec, view service metrics
Security/Privacy Officer	Review audit logs, approve scopes, view data residency policies

5) Core Functional Requirements

5.1 Routing & Discovery

- Path and host-based routing: `api.och.africa/v1/*` → appropriate service.
- Weighted & canary routing for gradual rollouts; sticky sessions optional.
- Circuit breaker and retry policies (idempotent methods only).

5.2 Authentication & Authorization

- Accept **JWT (OIDC)** from Identity service; verify via **JWKS** with key rotation.
- **OAuth2** (Auth Code + PKCE for user apps; Client Credentials for machine-to-machine).
- **API Keys** for simple server-to-server integrations (scoped).
- **RBAC/ABAC** enforcement at edge via OPA/rego policies using claims: `roles[]`, `org_id`, `cohort_id`, `consent_scopes[]`, `entitlements[]`.
- Mutual-TLS for privileged internal traffic (service mesh optional).

5.3 Quotas, Rate Limits, and Throttling

- Global default + per-consumer plan (e.g., Free: 60 req/min; Partner: 600 req/min; Internal: 3000 req/min).
- Burst control with token bucket; 429 on exceed; headers: `X-RateLimit-*`.

5.4 Versioning & Deprecation

- **URI versioning** (`/v1`, `/v2`) with **Sunset** response header for deprecated endpoints.
- Compatibility shims (field rename/alias) configurable at gateway transform layer.
- Lifecycle: **alpha → beta → GA → deprecated → EOL** with timelines in dev portal.

5.5 Request/Response Validation & Transformation

- **OpenAPI-backed validation** (schema, enums, formats), reject 400 with detailed errors.
- Field filtering: `?fields=prop1,prop2` and sparse expansions `?expand=items`.
- Normalized **error envelope** across services:
- `{ "error": { "code":"string", "message":"human readable", "details":[], "trace_id":"..." } }`
- Response shaping: rename, drop PII fields by policy (privacy overlays).

5.6 Pagination, Sorting, Filtering

- Standard: **cursor-based** via `?cursor=` + `limit` (default 50, max 200).
- Response meta:
- `{ "data":[...], "page":{"next_cursor":"...", "limit":50, "total_estimate":null} }`

5.7 Idempotency & Retries

- Require `Idempotency-Key` for POST/PUT/PATCH/DELETE that create/modify resources; cache results for 24h.
- Gateway retries **idempotent** upstream calls on 502/503/504 with jitter.

5.8 Caching

- **GET** responses cacheable (edge cache/Redis) honoring `Cache-Control` and `ETag/If-None-Match`.
- Per-route override TTLs; cache key includes auth scope when responses vary by user.

5.9 Webhooks & Event Delivery

- **Subscriptions:** partners register endpoints + topics + secret.
- **Signing:** HMAC-SHA256 with `X-OCH-Signature` and `X-OCH-Timestamp`; 5-minute replay window.
- **Retries:** exponential backoff (5, 30, 120, 600s), max 10 attempts; per-subscription DLQ with re-drive.
- **Message format:**
- `{ "id":"evt_...", "topic":"enrollment.created", "created_at":"...", "data":{...} }`
- **Handshake:** optional challenge response for URL verification.
- Status callbacks and delivery logs exposed to partners.

5.10 Partner Apps & Usage Plans

- Apps have `client_id`, `client_secret` (or public PKCE), **scopes**, **quotas**, **allowed IPs** (optional).
- Key rotation without downtime; sandbox and production environments.
- Partner metrics (calls, error rate, latency) visible in portal.

5.11 Schema Registry & Contract Testing

- **OpenAPI Registry** for all published APIs; diff alerts on breaking changes.
- **Event Schemas** (AsyncAPI) for webhook topics; versioned with deprecation windows.
- CI checks block deploys on contract violations unless explicitly waived.

5.12 Observability & Audit at the Edge

- Access logs (JSON) with `trace_id` and `consumer_id`.
- Metrics: request count, p50/p95/p99 latency, 4xx/5xx rates per route/consumer, cache hit ratio.
- **Audit**: who requested keys, changed quotas, subscribed to topics, downloaded secrets.

5.13 Security Controls

- **CORS**: allowlisted origins; preflight cache; credentials disabled by default.
- **WAF** rules: SQLi/XSS, path traversal, payload size caps, content type allowlist.
- **Header sanitation**: drop `X-Forwarded-*` overrides from clients.
- **PII leak prevention**: response filters; egress policies.
- **Geo/IP allow/deny** (optional for partners).
- **DDoS** integration (provider/CDN).

6) Data Model (PostgreSQL)

```
api_products(
  id UUID PK, name TEXT, version TEXT, status
ENUM[alpha,beta,ga,deprecated,eol],
  openapi_uri TEXT, asyncapi_uri TEXT, owner_user_id UUID, created_at,
updated_at
)

consumers(                -- internal services & partners
  id UUID PK, org_id UUID nullable, name TEXT, type
ENUM[internal,partner,public],
  status ENUM[active,suspended], created_at, updated_at
)

apps(                     -- partner/internal applications
  id UUID PK, consumer_id UUID FK, name TEXT, environment ENUM[sandbox,prod],
  grant_type ENUM[client_credentials,authorization_code,pkce,api_key], scopes
TEXT[],
  quota_per_min INT, quota_per_day INT, ip_allowlist CIDR[] null, created_at
)

app_credentials(
  id UUID PK, app_id UUID FK, type
ENUM[api_key,oauth_secret,webhook_secret,signing_key],
  hash TEXT, last_rotated_at TIMESTAMP, expires_at TIMESTAMP null, active
BOOL
)

webhook_subscriptions(
  id UUID PK, app_id UUID FK, topic TEXT, target_url TEXT, secret_ref UUID FK
app_credentials,
  status ENUM[active,paused], created_at, last_delivery_at TIMESTAMP
)

webhook_deliveries(
  id UUID PK, subscription_id UUID FK, event_id TEXT, attempt INT, status
ENUM[pending,delivered,failed,queued,dead_letter],
```

```
  request_headers JSONB, response_code INT, response_ms INT, error TEXT,
created_at
)

usage_logs_daily(         -- aggregated metering
  day DATE, app_id UUID, product_id UUID, calls INT, bytes_in BIGINT,
bytes_out BIGINT,
  p95_ms INT, errors_4xx INT, errors_5xx INT, PRIMARY KEY(day, app_id,
product_id)
)

policies(                 -- OPA/rego references
  id UUID PK, name TEXT, version INT, scope TEXT, rego TEXT, active BOOL,
published_at
)
```

7) Process Flows

7.1 Partner Onboarding

1. Org admin signs up in **Developer Portal** → creates **App (sandbox)**.
2. Choose product(s)/scopes → approval (auto/manual).
3. Generate credentials; subscribe to webhook topics; verify endpoint (challenge).
4. Test in sandbox; promote to production (new keys); set quotas & IP allowlist.

7.2 Webhook Delivery

1. Core service emits domain event → Event Router → fetch active subscriptions by topic.
2. For each subscription, compose payload, sign with secret, POST to `target_url`.
3. On 2xx: mark delivered; On 4xx/5xx/timeouts: retry with backoff; after max attempts → DLQ.
4. Partner can re-drive DLQ via portal/API.

7.3 API Version Deprecation

1. Product owner marks v1 as **deprecated** with **Sunset** date.
2. Gateway injects **Deprecation** and **Sunset** headers + warning in body for 60 days.
3. Usage reports sent to heavy users; after sunset, traffic 301/410 or routed through shim for grace period.

7.4 Key Rotation

1. Partner initiates rotation → new credential issued (overlap period).
2. Calls accepted with either key for N hours; old key revoked; audit event logged.

8) Public Surface (Examples)

Base: `https://api.och.africa/v1`

- **Identity proxy:** `/auth/*` (authorize, token, jwks.json)
- **Programs:** `/programs`, `/cohorts`, `/tracks`
- **Profiling:** `/profiling/assessments`, `/profiles/{user_id}`
- **Learning:** `/learning/providers`, `/learning/me/progress`
- **Mentorship:** `/mentorship/sessions`, `/mentorship/goals`
- **Portfolio:** `/portfolio/items`, `/portfolio/reviews`, `/public/{slug}`
- **Gamification:** `/gamification/leaderboards`, `/gamification/me/summary`
- **Billing:** `/billing/catalog`, `/checkout/sessions`, `/invoices`, `/entitlements`
- **Analytics:** `/analytics/metrics/{key}`, `/dashboards/{id}/pdf`

Gateway-Managed Endpoints

- **Developer portal:** `/dev/apps`, `/dev/keys`, `/dev/webhooks`, `/dev/usage`
- **Webhook management:** `/dev/subscriptions` (CRUD), `/dev/dlq/{id}/redrive`
- **Metadata:** `/.well-known/openapi`, `/.well-known/asyncapi`, `/status`

Standard Headers

- Requests: `Authorization`, `Idempotency-Key`, `X-Request-Id` (optional), `X-OCH-Org` (optional)
- Responses: `Trace-Id`, `X-RateLimit-Limit`, `X-RateLimit-Remaining`, `X-RateLimit-Reset`, `Deprecation`, `Sunset`, `ETag`

9) Developer Portal Requirements

- OAuth login (IdP or email) with org association.
- Product catalog, scopes, plans, terms.
- **Try-it console** with sandbox credentials; code snippets & SDKs (TS/Python).
- Usage analytics (per day, per route), error drill-down, latency charts.
- Credential lifecycle (create, rotate, revoke), IP allowlist.
- Webhook subscription tester (send sample events, inspect signature).
- Documentation hosting (OpenAPI, AsyncAPI) with change logs and deprecation notices.

10) Security & Compliance

- TLS 1.3 everywhere; HSTS.
- Strict **CSP**, **X-Content-Type-Options**, **Referrer-Policy**, **X-Frame-Options**.
- WAF + bot mitigation; payload size limits (e.g., 5 MB default).
- **PII minimization** on responses; schema overlays from CPPC (Consent).
- **Geo residency tags** to route/store data appropriately if required.

- Full **audit** for key actions (key issue/revoke, scope changes, webhook subscription updates).
- **Secrets** in Vault/KMS; no plaintext at rest.

11) Tech Stack

- **Gateway**: **Kong Gateway** or **Envoy Gateway** (with OPA/rego for policy).
- **Service Mesh (optional)**: Istio/Envoy for east-west mTLS & retries.
- **Schema Registry**: OpenAPI/AsyncAPI in Git registry + registry service.
- **Cache**: Redis (edge cache + idempotency store).
- **DB**: PostgreSQL for registry, consumers, usage metering.
- **Queue**: Kafka/Redis Streams for webhook fan-out & DLQ.
- **Dev Portal**: Kong Dev Portal or custom Next.js app.
- **Observability**: OpenTelemetry, Prometheus, Grafana, Loki, Tempo/Jaeger.
- **CDN/DDoS**: Cloudfront/Cloudflare (optional front of gateway).

12) Error Handling Matrix

Scenario	Behavior
Invalid/expired JWT	401 with `WWW-Authenticate` hint
Scope/ABAC denied	403 with `required_scope`
Rate limit exceeded	429 with retry headers
Unsupported API version	410 (Gone) with alternatives
Schema validation fail	400 with detailed field errors
Upstream timeout	504; gateway retries idempotent routes once
Webhook signature invalid	401 to partner endpoint; delivery marked failed; retry applies

Error envelope (always):

```
{ "error": { "code":"forbidden", "message":"Scope employer.read missing",
"trace_id":"..." } }
```

13) Testing & Release

- **Contract tests** (Dredd/Postman/newman) for OpenAPI & AsyncAPI on CI.
- **Canary rollouts** with traffic mirroring and shadowing.
- **Pen-tests** focusing on auth bypass, header smuggling, SSRF via webhook validation.
- **Chaos tests** for upstream timeouts and retry/circuit behavior.
- **Load tests** to verify quotas and latency SLOs.

14) Future Enhancements

- **GraphQL facade** for selective read models (read-heavy partner use cases).

- **gRPC** gateway for high-throughput internal comms.
- **Fine-grained response encryption** (field-level, partner public keys).
- **Monetization** (billable usage exports to Finance).
- **Async subscriptions over SSE/WebSockets** (in addition to webhooks).

Notifications & Automation Service (NAS)

Technical Specification (Developer-Ready)

1) Module Overview

The **Notifications & Automation Service (NAS)** is the unified communication and workflow orchestration layer across the OCH SMP platform.
It manages **all outbound and in-app communications**, **event-driven triggers**, **scheduled automations**, and **reminders** for user activities, program operations, and business processes.

It enables both **system-driven** (automated) and **manual** (director-initiated) notifications through multiple channels:

- Email (e.g. SendGrid, SES)
- SMS/WhatsApp (e.g. Twilio)
- In-App Notifications
- Push Notifications (mobile/web)
- Webhooks (for integrations)

Value: Provides consistency, reliability, and observability for all cross-platform alerts, messages, and scheduled workflows.

2) Functional Objectives

1. Deliver **multi-channel notifications** with guaranteed delivery, retries, and tracking.
2. Support **rule-based automations** and **scheduled triggers** (daily, weekly, milestone-based).
3. Manage **notification templates** for all modules (Profiling, Mentorship, Billing, etc.).
4. Enable **personalized dynamic content** with variable injection ({{name}}, {{cohort}}, {{due_date}}).
5. Provide **preference management** (opt-in/out per channel or notification type).
6. Centralize **notification logs** and **delivery metrics** for analytics and audit.
7. Expose **API hooks and SDKs** for other microservices to publish events and schedule automations.
8. Guarantee **idempotent, fault-tolerant message delivery** with exponential backoff and DLQs (Dead Letter Queues).

3) User Roles & Permissions

Role	Capability	Example Use Case
Admin	Full control over templates, automations, schedules, and delivery logs	Update system-wide templates or resend campaigns

Role	Capability	Example Use Case
Program Director	Trigger cohort-wide notifications; configure automation rules	Send reminders for deadlines or sessions
Mentor	Send personalized messages to mentees (if allowed by policy)	Notify mentees of next mentorship meeting
Student	View in-app notifications and manage preferences	Opt out of SMS but keep email alerts
Finance	View payment-related message logs	Track failed payment notification logs
System Services	Publish system events to NAS	Trigger notifications automatically

4) Core Functional Requirements

4.1 Template Management

- **Description:** Centralized store for email, SMS, in-app, and push templates.
- **Acceptance Criteria:**
  - CRUD for templates by type: `email`, `sms`, `in_app`, `push`, `webhook`.
  - Dynamic variable injection (Liquid/Mustache syntax).
  - Template versioning with approval flow.
  - Preview and test-send capability.
  - Support for localization (language variants).
- **I/O:** `template_id`, `type`, `variables[]`, `version`, `language`, `subject`, `body_html`, `body_text`.

4.2 Event Subscription & Trigger Engine

- **Description:** Event bus consumer that listens for key domain events and triggers notification workflows.
- **Acceptance Criteria:**
  - Subscribes to platform events (e.g. `enrollment.created`, `payment.failed`, `certificate.issued`, `calendar_event.soon`).
  - Match events to configured templates or automation rules.
  - Deduplicate repeated events (idempotency key = `event_id`).
  - Retry failed deliveries with exponential backoff (max 5 tries).
- **Dependencies:** Event bus (Redis Streams, Kafka).
- **I/O:** `event_name`, `payload`, `trigger_id`, `delivery_status`.

4.3 Automation Rules Engine

- **Description:** Define conditional triggers for workflows and reminders.
- **Acceptance Criteria:**
  - Rule definition: **if (event or schedule) + condition + action(s)**.

    - Actions: send message, invoke webhook, schedule reminder, create task.
    - Support for composite rules (AND/OR logic).
    - Built-in variables: user, cohort, program, milestone, mentor, date.
    - Example:
    - ```
      {
        "trigger": "calendar_event.soon",
        "conditions": { "event_type": "submission", "hours_before": 24 },
        "actions": [
          { "type": "email", "template": "submission_reminder" },
          { "type": "in_app", "template": "submission_alert" }
        ]
      }
      ```
    - Rule status: draft, active, paused.
- **I/O:** `rule_id`, `trigger`, `conditions`, `actions`, `status`.

4.4 Scheduler & Reminder System

- **Description:** Time-based jobs for cohort or system-wide events.
- **Acceptance Criteria:**
    - Cron-based scheduling: hourly, daily, weekly, monthly.
    - Context-aware scheduling (based on user timezone).
    - Retry missed executions due to downtime.
    - Linked to calendar milestones (e.g., reminders before submissions or sessions).
- **Dependencies:** Calendar Hub, Program Module.
- **I/O:** `schedule_id`, `cron_expr`, `context`, `last_run`, `next_run`.

4.5 Delivery Engine

- **Description:** Multi-channel delivery and routing layer.
- **Acceptance Criteria:**
    - Channels: Email, SMS, Push, In-App, Webhook.
    - Fallback hierarchy: e.g., push → email → SMS.
    - Delivery queues per channel (RabbitMQ/Redis).
    - Track statuses: `queued`, `sent`, `delivered`, `bounced`, `failed`.
    - Rate limiting (per user/org/channel).
    - Support for async and bulk sends.
- **I/O:** `notification_id`, `channel`, `recipient`, `payload`, `status`, `retries`, `sent_at`.

4.6 Notification Preferences Center

- **Description:** Allow users to control notification frequency and channels.
- **Acceptance Criteria:**
    - Preferences per notification type: `system`, `billing`, `mentorship`, `reminder`, `marketing`.
    - Channel-specific toggles (email, sms, in-app).
    - Default preferences by role (e.g., mentors get SMS reminders by default).
      ```

    - Consent integration (disable marketing if not consented).
- **I/O:** `preference_id`, `user_id`, `type`, `channels[]`, `muted`, `updated_at`.

4.7 Notification Center (In-App)

- **Description:** Real-time notification hub embedded in SMP UI.
- **Acceptance Criteria:**
    - WebSocket or long-polling for instant updates.
    - Grouped notifications (by date/type).
    - Mark as read/unread; archive/delete.
    - Links to originating module (Portfolio, Billing, etc.).
- **I/O:** `notification_id`, `user_id`, `title`, `message`, `type`, `url`, `read_status`, `timestamp`.

4.8 Logging & Metrics

- **Description:** Central tracking for delivery, performance, and errors.
- **Acceptance Criteria:**
    - Logs: sent_count, failed_count, retry_count, channel_latency_ms, last_delivery_at.
    - BI integration for open/click/conversion rates.
    - Exportable CSV and API endpoints for audit.
- **Dependencies:** Analytics Layer, Observability Stack.
- **I/O:** `notification_log_id`, `metric_name`, `value`, `timestamp`.

5) Data Model (PostgreSQL)

Table	Description
`notification_templates`	id UUID PK, key TEXT unique, channel ENUM[email,sms,in_app,push,webhook], subject TEXT, body_html TEXT, body_text TEXT, variables JSONB, language TEXT, version INT, active BOOL, created_by, created_at
`notification_rules`	id UUID PK, name TEXT, trigger TEXT, conditions JSONB, actions JSONB, status ENUM[draft,active,paused], created_at, updated_at
`notification_schedules`	id UUID PK, name TEXT, cron_expr TEXT, context JSONB, last_run TIMESTAMP, next_run TIMESTAMP
`notification_deliveries`	id UUID PK, user_id UUID, channel ENUM[email,sms,in_app,push,webhook], template_id FK, payload JSONB, status ENUM[queued,sent,delivered,bounced,failed], retries INT, sent_at TIMESTAMP, delivered_at TIMESTAMP, error TEXT

Table	Description
`user_notification_preferences`	id UUID PK, user_id UUID, type TEXT, channels TEXT[], muted BOOL, updated_at TIMESTAMP
`notification_logs`	id UUID PK, event_id TEXT, trigger TEXT, rule_id FK, status TEXT, meta JSONB, created_at TIMESTAMP

6) Process Flow Examples

1. Event-Driven (Immediate)

```
event: enrollment.created
↓
rule match: send "welcome_email" to user
↓
queue: email_channel
↓
delivery engine → provider → status update
↓
log + analytics + user in-app record
```

2. Scheduled (Reminder)

```
rule: calendar_event.soon (24h before)
↓
scheduler triggers daily scan
↓
find matching events
↓
send SMS + in-app reminder
↓
record delivery & mark event as reminded
```

3. Manual Broadcast

```
director → dashboard → select cohort
↓
choose template "weekly_update"
↓
NAS bulk queue → personalized merges
↓
track per-user delivery and open rate
```

7) API Endpoints

Service prefix: `/api/v1/notifications`

Endpoint	Method	Description
`/templates`	POST/GET/PUT/DELETE	Manage templates

Endpoint	Method	Description
`/send`	POST	Send ad-hoc or test notification
`/rules`	POST/GET	Define or list automation rules
`/preferences`	GET/PUT	View or update user preferences
`/deliveries`	GET	Query notification logs
`/inapp`	GET/PUT	List or mark in-app notifications as read
`/schedules`	POST/GET	Create or manage cron jobs
`/stats`	GET	Summary metrics (per channel/type)

Events emitted:

- `notification.sent`
- `notification.failed`
- `notification.read`
- `notification.rule.triggered`

8) Integrations

Service	Integration
Program & Cohort Mgmt	Triggers for start/end dates, milestones, seat changes
Billing	Payment confirmations, failed payment reminders
Profiling & Portfolio	Assessment invites, submission reminders
Mentorship Mgmt	Session reminders, feedback summaries
Gamification	Achievement unlocked, leaderboard rank changes
Analytics	Export logs for reporting
Consent/Privacy	Enforce communication opt-in/out

9) UI/UX Requirements

For Students

- In-App “bell” icon showing latest 10 notifications.
- Read/unread visual cues.
- Settings page for preferences (per category/channel).
- SMS/email unsubscribe link.

For Directors/Mentors

- “Broadcast” panel to message cohorts.
- Template preview & personalization fields.
- Automation rule builder (drag-drop conditions & actions).

For Admin

- Dashboard: channel performance, delivery success %, top rules, error logs.
- Rule testing sandbox.

10) Security & Compliance

- **Access Control:**
  - Only admins can modify templates or global rules.
  - Directors limited to their cohorts.
  - Students can only view their notifications.
- **Data Protection:**
  - PII redacted in logs (e.g., emails masked).
  - Consent checks before sending marketing content.
  - Email/SMS providers integrated via secure API keys in Vault/KMS.
- **Compliance:**
  - Botswana DPA / GDPR-compliant opt-in/out model.
  - Email headers include unsubscribe link.
  - Audit trail on every notification sent.
- **Resilience:**
  - Retry queue with backoff; DLQ for failed deliveries.
  - SLA: 99.9% notification delivery uptime; <3s median latency.

11) Technical Dependencies

Component	Tech
Event Bus	Kafka / Redis Streams
Scheduler	BullMQ / Celery Beat
Email	SendGrid / AWS SES
SMS/WhatsApp	Twilio / MessageBird
Push	Firebase Cloud Messaging
In-App	WebSocket (Socket.io / Pusher)
Backend	FastAPI / NestJS
Database	PostgreSQL
Caching	Redis
Monitoring	Prometheus + Grafana; Sentry for errors

12) Error Handling

Case	Handling
Provider API timeout	Retry 3x exponential backoff
Invalid contact (email/SMS)	Flag + suppress future sends

Case	Handling
Template parse error	Fallback to plain text default
Rate limit reached	Queue hold + delayed retry
Event loop failure	Write to DLQ + alert admin

13) Future Enhancements

1. **AI-Powered Engagement Scoring** — predict disengaged students and auto-nudge them.
2. **A/B Testing for Templates** — optimize open and click rates per cohort.
3. **Omnichannel Orchestration Dashboard** — visualize message timelines per user.
4. **Voice notifications** (optional integration with Twilio Voice for directors).
5. **Mentor Chatbot Integration** — automate responses for common questions.

Consent, Privacy & Policy Center (CPPC)

Technical Specification (Developer-Ready)

1) Module Overview

Central service for **consents, privacy preferences, policy versions, subject rights (SAR/Export/Erasure)**, and **data retention**.
It enforces purpose limitation and lawful basis for processing across modules (Profiling, LRS, Portfolio, Mentorship, Billing, Analytics, Notifications, Employer Portal).

Value: single source of truth for who consented to what, when, under which policy; automated fulfillment of privacy requests; auditable compliance for Botswana DPA/GDPR-aligned regimes.

2) Objectives

1. Capture & version **consent records** tied to **policies** and **purposes** (scopes).
2. Provide **real-time consent checks** (allow/deny) for any resource access or event emission.
3. Manage **user privacy preferences** (channels, marketing, sharing, public profile).
4. Execute **data subject rights**: **Access/Export**, **Rectification**, **Restriction**, **Erasure**, **Objection**.
5. Automate **data retention & deletion** (scheduled, event-based, hold exceptions).
6. Maintain **immutable audit trail** of privacy-relevant events.
7. Supply **compliance reports** and evidence packs.

3) Scope & Definitions

- **Policy**: human-readable + machine rules (versioned).
- **Purpose/Scope** examples:
  - `profiling.share_with_mentor`
  - `portfolio.public_page`
  - `learning.connect_external_provider`
  - `notifications.marketing`
  - `employer.view_candidate_profile`
  - `analytics.personalized` (vs `analytics.aggregate_only`)
- **Lawful Basis**: consent, contract, legal obligation, legitimate interests (configurable per scope).

4) Roles & Permissions

Role	Capabilities
Admin (Privacy Officer)	Publish policies, configure purposes/basis, approve SAR/Erasure, retention rules

Role	Capabilities
Data Steward	Triage SARs, run exports, coordinate erasures
Program Director	View consent status for students in owned cohorts; request consent updates
Mentor	See limited consent flags on assigned mentees
Student	View/update preferences; grant/revoke consents; create privacy requests
Sponsor/Employer Admin	View only sponsor-related consents (if granted)
System Services	Call `/check` gate; receive erase/export tasks

5) Core Functional Requirements

5.1 Policy & Purpose Registry

- CRUD for **policies** (title, version, effective dates, locales), **purposes/scopes**, **lawful basis**, **default state**.
- Render human-readable policy pages; expose policy JSON for SDK enforcement.
- Versioning rules: **no edits to published versions**; create new version & migrate users on acceptance.

5.2 Consent Capture & Management

- Consent prompts: inline (e.g., first connect to provider), onboarding bundle, and **transactional prompts** (e.g., publishing portfolio).
- Evidence: who/when/how (UI/IP/UA or API), policy version, locale.
- States: `granted`, `denied`, `withdrawn`, `expired`.
- Bulk request flows (e.g., new scope introduced → ask all affected users).

5.3 Preference Center

- Fine-grained toggles for **marketing**, **in-product tips**, **SMS**, **email**, **public page**, **employer sharing**.
- Quiet hours; channel priorities; category opt-outs (enforced by NAS).

5.4 Real-Time Enforcement API

- `/consents/check` returns **allow/deny + reason + policy_version**.
- Supports **subject + resource + purpose** decision with **sticky caching** (short TTL) for performance.
- Deny returns `required_scope` to drive UX prompts.

5.5 Subject Access Requests (SAR) & Erasure

- SAR types: **Access/Export (machine-readable)**, **Rectify**, **Restrict**, **Object**, **Erase**.
- Workflow: **submitted → triage → data collection → review → deliver/execute → close**.
- Export bundles: JSON/CSV + file artifacts (portfolio) in ZIP; watermarked; link expires.
- Erasure: cascade **delete/anonymize** across services via task fan-out; **legal holds** and **billing exceptions** honored.
- SLA timers and reminders (e.g., 30 days).

5.6 Retention & Deletion Engine

- Rules per entity type: `audit_logs: 5y`, `session_logs: 90d`, `in_app_messages: 18m`, `portfolio_drafts: 12m`, etc.
- "**Last activity**" and "**closure**" based timers; **holds** for ongoing disputes.
- Dry-run mode + reports before irreversible deletions.

5.7 Audit & Reporting

- Immutable audit of: consent changes, policy publications, SAR lifecycle, deletions.
- Prebuilt reports: consent coverage by scope, marketing opt-in rate, open SARs, retention actions.
- Evidence pack generator for regulators (PDF + CSV).

6) Data Model (PostgreSQL)

```
policies(
  id UUID PK, key TEXT, version INT, title TEXT, locale TEXT, body_md TEXT,
  effective_from TIMESTAMP, effective_to TIMESTAMP null, status
ENUM[draft,published,retired],
  created_by UUID, created_at TIMESTAMP
)

purposes(
  id UUID PK, key TEXT unique, name TEXT, description TEXT,
  lawful_basis ENUM[consent,contract,legal_obligation,legitimate_interest],
  default_state ENUM[granted,denied], policy_id UUID FK, active BOOL
)

consents(
  id UUID PK, user_id UUID, purpose_key TEXT, policy_id UUID, state
ENUM[granted,denied,withdrawn,expired],
  evidence JSONB, granted_at TIMESTAMP, updated_at TIMESTAMP
  -- UNIQUE (user_id, purpose_key) current state; keep history in
consent_history
)

consent_history(
  id UUID PK, consent_id UUID FK, prev_state TEXT, new_state TEXT, changed_by
UUID,
  changed_at TIMESTAMP, reason TEXT, evidence JSONB
)
```

```
preferences(
  id UUID PK, user_id UUID, category
ENUM[system,reminders,billing,mentorship,marketing,public,employer_share],
  channels JSONB, quiet_hours JSONB, updated_at TIMESTAMP
)

privacy_requests(
  id UUID PK, user_id UUID, type
ENUM[access,export,rectify,restrict,object,erase],
  status
ENUM[submitted,triage,collecting,review,fulfilled,rejected,canceled],
  submitted_at TIMESTAMP, due_at TIMESTAMP, notes TEXT, assigned_to UUID
)

privacy_request_items(
  id UUID PK, request_id UUID FK, service TEXT, entity TEXT, entity_id TEXT,
  action ENUM[export,erase,anonymize,notify], status
ENUM[pending,done,error,skipped], detail JSONB
)

retention_rules(
  id UUID PK, entity TEXT, scope TEXT, condition JSONB, ttl_days INT,
  action ENUM[delete,anonymize], active BOOL, created_at
)

retention_jobs(
  id UUID PK, rule_id UUID FK, started_at TIMESTAMP, finished_at TIMESTAMP,
  stats JSONB, report_uri TEXT, status ENUM[ok,partial,failed]
)

privacy_audit_logs(
  id UUID PK, actor_id UUID, event TEXT, subject_id UUID, entity TEXT,
entity_id TEXT,
  meta JSONB, at TIMESTAMP
)
```

Notes

- PII minimized in CPPC; store **references** to entities; heavy data stays in source systems.
- All exports stored temporarily (time-boxed URLs), encrypted at rest.

7) APIs (prefix `/api/v1/privacy`)

Public/User

- `GET /policies/current?locale=` — active policy bundle.
- `GET /consents/my` — list current consents.
- `POST /consents/my` — grant/withdraw (`{purpose_key, state}`), returns policy_version.
- `GET /preferences/my` / `PUT /preferences/my` — manage preferences.
- `POST /requests` — create SAR (`type`, optional notes).

- `GET /requests/my` — track SAR status; download exports.

Service-to-Service

- `POST /check` — decision point:
- `{ "user_id":"...", "purpose_key":"portfolio.public_page", "context": {"cohort_id":"..."} }`

  Response: `{ "allow": true, "policy_version": 3, "reason": "granted" }`

- `POST /requests/{id}/items` — services push progress on erase/export.
- `POST /retention/run` — trigger rule by key (admin only).

Admin

- `POST /policies` / `POST /policies/{id}/publish` / `POST /policies/{id}/retire`
- `POST /purposes` / `PUT /purposes/{id}`
- `GET /reports/consent-coverage?purpose_key=`
- `GET /audits?event=&subject=&range=`

Idempotency on all mutations via `Idempotency-Key` header.

8) Process Flows

A) Consent Gate (Portfolio Publish)

1. Student clicks “Publish Portfolio”.
2. App calls `/privacy/check` with `purpose_key=portfolio.public_page`.
3. If `allow=false` & basis=consent → CPPC returns `required_scope`.
4. UI shows consent modal (policy vN summary).
5. On confirm, `POST /consents/my` → **granted** + evidence stored.
6. Retry action → allowed; emit audit event.

B) Employer Access (Sharing)

1. Employer requests candidate view → app calls `/check` for `employer.view_candidate_profile`.
2. If granted → allow; else present **just-in-time** consent to student via NAS; status pending until user responds.

C) SAR — Erasure

1. User submits **Erase** → CPPC creates `privacy_requests` + items per service (Identity, Mentorship, Portfolio, LRS, Gamification, NAS).

2. Fan-out tasks via message bus; services execute **delete/anonymize**; report back with `privacy_request_items`.
3. CPPC collates results; if billing/legal hold exists → partial erasure + note.
4. Close request; send evidence report.

D) Retention Job

1. Nightly scheduler reads `retention_rules`.
2. For each rule, CPPC requests candidate IDs from services (via lightweight query endpoints).
3. Run delete/anonymize; write `retention_jobs` report; send summary to Privacy Officer.

9) Integrations

Module	Enforcement / Interaction
Identity	Embed consent scopes into JWT claims; handle account erasure & session revocation
Program/Cohorts	Policy links in onboarding; cohort-level notices
LRS (Learning)	Connection requires `learning.connect_external_provider` consent
Mentorship	Mentor visibility relies on `profiling.share_with_mentor`
Portfolio	Public page and employer-share consents gate publishing & sharing
Billing	Retention exceptions; invoices kept per law; mask PII in exports
Notifications (NAS)	Marketing sends require opt-in; preferences center writes
Analytics	Respect `analytics.aggregate_only`; RLS masks PII; SAR exports pull curated facts

10) UI/UX Requirements

- **Consent Banner & Drawer** on first use; link to full policy.
- **Just-In-Time Modals** for sensitive actions (publish/share/connect).
- **Preferences Page**: categories, channels, quiet hours, “delete my account”, “download my data”.
- **Request Tracker**: timeline for SARs with ETA; secure download links.
- Accessibility (WCAG 2.1 AA), clear plain-language explanations.

11) Security & Compliance

- **Lawful Basis Registry**: every purpose must declare basis; logs show decisions.
- **Audit Everything**: policy publishes, consent changes, SAR steps, retention actions.
- **Encryption**: exports at rest (KMS) + signed, short-lived URLs; TLS 1.3 in transit.
- **PII Minimization**: CPPC stores references & evidence, not full datasets.

- **Data Residency**: tag records with residency requirements if applicable.
- **SLAs**: SAR ack ≤72h, fulfill ≤30 days (configurable).
- **Rate Limits**: on SAR creation & export downloads.
- **Abuse Controls**: verify identity before fulfilling sensitive SAR (step-up MFA).

12) Tech Stack

- **Service**: FastAPI / NestJS; JSONSchema for policy rules.
- **DB**: PostgreSQL 15; partition `privacy_audit_logs`.
- **Queue**: Redis Streams / Kafka for fan-out tasks.
- **Storage**: S3 (KMS) for exports & reports.
- **Infra**: Runs behind API Gateway; internal service auth via mTLS + service tokens.
- **Observability**: OpenTelemetry, Prometheus, Grafana, Sentry.

13) Error Handling

Case	Handling
Missing purpose_key	400 with hint to register purpose
Check timeout	Fail closed (deny) + retry; raise alert
Conflicting consent states	Resolve by `updated_at` (latest wins); keep history
Erasure task fails	Mark item `error`; retry policy; escalate to steward
Export link expired	410; re-issue if request still open
Retention dry-run violations	Report only; require admin confirm to execute

14) Future Enhancements

- **Dynamic Risk-based Prompts** (less intrusive if low risk).
- **Consent Provenance Signatures** (cryptographic receipts).
- **DSAR Self-Serve Diff Exports** (only changes since last export).
- **PII Discovery Scanner** to map stray PII across services.
- **Regional Policy Overlays** per country (Botswana, Kenya, Namibia, Zimbabwe).

Audit & Observability (A&O)

Technical Specification (Developer-Ready)

1) Module Overview

Unified platform layer for:

- **Audit** (who did what, to which entity, when, from where, under which policy).
- **Observability** (metrics, logs, traces, health/SLA, alerts, SLO error budgets).
- **Runtime diagnostics** (feature flags exposure, config snapshots, incident timeline).

Value: provable compliance, faster incident response, trustworthy SLAs, and high developer velocity with safe ops.

2) Objectives

1. Capture **immutable audit events** for security-/privacy-relevant actions across all modules.
2. Provide **centralized metrics, logs, and distributed tracing** with correlation IDs across services.
3. Expose **health, readiness, and dependency checks** + SLOs and error budgets.
4. Offer **dashboards, alerting policies, and runbooks**; store incidents and postmortems.
5. Enforce **data retention & access control** for operational data (PII-minimal, compliant).

3) Scope & Responsibilities

- **In-scope:** platform-wide audit pipeline, log/trace collection, metrics collection, health/SLOs, alerting, incident records.
- **Out-of-scope:** business analytics (handled by Analytics & Reporting Layer), app feature telemetry beyond standard events.

4) Roles & Permissions

Role	Capabilities
Admin (SRE/Platform)	Configure collectors, alert policies, SLOs, retention; view all telemetry; export incident packs
Security/Privacy Officer	Search/view **audit** stream, export evidence, sign postmortems
Program Director / Finance	Read only **service health dashboards**; no raw logs

Role	Capabilities
Developers	Read own service logs/traces & metrics; masked PII; request temporary elevated access via break-glass
Auditors (External)	Time-boxed, read-only access to scoped audit exports

5) Core Functional Requirements

5.1 Audit Trail (Immutable)

- **Event shape (canonical):**
- `{`
- `  audit_id, at, actor: {type[user|service], id, roles[]},`
- `  action, entity: {type, id}, scope: {org_id?, cohort_id?},`
- `  result: {status[success|failure], reason?},`
- `  source: {ip, user_agent, device?, geo?},`
- `  policy: {rbac:[], abac:[], consent_scopes[]},`
- `  correlation_id, metadata (redacted where needed)`
- `}`
- **Mandatory audit events** (non-exhaustive):
  - Auth: login/logout, MFA enroll/verify/disable, SSO link/unlink, token revocation
  - Identity: role/policy/permission change, API key create/revoke
  - Privacy: consent grant/withdraw, SAR lifecycle, retention delete/anonymize
  - Billing: price change, refund/void, payout reconciliation override
  - Program/Cohort: enrollment create/update/status, certificate issue
  - Portfolio/Mentorship/LRS: publish/unpublish/visibility change, manual score overrides
- **Characteristics:** append-only, write-once store; clock synchronized (NTP), monotonic IDs; tamper-evident hash chain (optional v2).
- **Access:** query API with RLS; exports with watermarking.

5.2 Structured Application Logs

- JSON logs with **correlation_id** injected at gateway; fields: level, ts, svc, env, req_id, span_id, route, latency_ms, status, error_code, safe_message, tags[].
- **No secrets/PII**; redaction middleware for common keys (email, phone, tokens).
- Retention defaults: **7–14 days hot**, **90 days warm**; summaries to warehouse (optional).

5.3 Metrics (Golden Signals + Business-adjacent)

- **Service metrics:** RPS/QPS, latency (p50/p90/p99), error_rate, saturation, queue_depth, job_failures.
- **Channel metrics:** notifications sent/delivered/bounced; payment auth/capture success; webhook success rate.
- **Resource metrics:** DB connections, cache hit ratio, CPU/mem/disk, GC pauses.

- **Custom module probes:** dunning recovery %, cohort seat utilization sync lag (as health indicators).

5.4 Distributed Tracing

- Propagate **W3C tracecontext** across gateway → services → DB/cache → external APIs.
- Sample rate: 10–20% baseline, 100% for error traces; tail-based sampling where supported.
- Redact attributes; link **audit_id** & **correlation_id**.

5.5 Health, Readiness & SLOs

- Endpoints per service: `/healthz` (liveness), `/readyz` (dependencies), `/metrics` (Prometheus).
- **SLO templates:**
  - API availability: **99.9% monthly** (latency < 500 ms for p95; error_rate < 0.1%).
  - Checkout success: **≥ 98%** auth success (rolling 7-day).
  - Notification delivery: **≥ 99%** in ≤ 30s (email/in-app), SMS ≤ 2 min.
- **Error budgets** with burn alerts (fast/slow burn multi-window policies).

5.6 Alerting & On-Call

- Severity levels (SEV1–SEV4), ownership per service, **runbook URL** on every alert.
- Multi-channel: PagerDuty/Opsgenie, email, Teams/Slack; quiet hours for low-sev.
- **Correlation:** deduplicate by correlation_id; group related failures.

5.7 Incidents & Postmortems

- Create incident records, timeline (alerts, changes, deployments, user impact), linked traces/dashboards.
- Postmortem template (5 whys, corrective actions, owners, due dates); track **action items** until closed.

5.8 Change Tracking & Config Snapshots

- Record deploys (commit SHA, artifacts, migration IDs), feature flag flips, config changes.
- Show diff per environment; attach to incidents automatically.

6) Data Model (PostgreSQL + Columnar/OLAP for hot queries)

Operational (PostgreSQL)

- `audit_events`

    - `(id UUID PK, at TIMESTAMP, actor_type TEXT, actor_id UUID, roles JSONB, action TEXT, entity_type TEXT, entity_id TEXT, scope JSONB, result JSONB, source JSONB, policy JSONB, correlation_id TEXT, metadata JSONB, hash TEXT, prev_hash TEXT)`
- `incidents`
    - `(id UUID PK, severity INT, status TEXT, title TEXT, started_at, resolved_at, commander_id UUID, description TEXT, runbook_uri TEXT)`
- `incident_events`
    - `(id UUID PK, incident_id FK, at, type TEXT, payload JSONB, correlation_id TEXT)`
- `postmortems`
    - `(id UUID PK, incident_id FK, summary TEXT, causes JSONB, actions JSONB, published_at)`
- `deploy_events`
    - `(id UUID PK, service TEXT, env TEXT, version TEXT, commit_sha TEXT, started_at, finished_at, result TEXT, metadata JSONB)`

Telemetry (external backends)

- Logs → OpenSearch/Loki;
- Metrics → Prometheus (TSDB);
- Traces → Tempo/Jaeger.
  (Keep only **IDs**/links in Postgres for cross-reference.)

Indexes: `audit_events(correlation_id)`, `(entity_type, entity_id, at)`, `(actor_id, at)`. Partition `audit_events` by month.

7) Ingestion & Pipelines

- **SDKs/Middlewares** for all services (Node/FastAPI/NestJS):
    - Inject correlation_id; emit audit events on tagged controller/service methods.
    - Structured logging; trace spans; error capture.
- **Sidecar/Agent** for logs & traces (OTel Collector).
- **Gateway filter** sets request IDs; drops PII headers; rate-limit log floods.
- **Backpressure/DLQ** for audit writes (Kafka/Redis Streams → writer service with idempotency).

8) APIs (prefix `/api/v1/ao`)

- `POST /audit` (service-to-service) — batch write (max 500 events; idempotency key).
- `GET /audit/search?actor=&action=&entity_type=&entity_id=&from=&to=&scope=` — RLS enforced.
- `GET /incidents/:id` / `POST /incidents` / `POST /incidents/:id/close`
- `POST /incidents/:id/events`
- `POST /postmortems` / `GET /postmortems/:id`
- `GET /deploys?service=&env=&from=&to=`

- `GET /dashboards/links` — returns URLs to metrics/logs/traces per service (signed).

Health/SLO

- Each service exposes `/healthz`, `/readyz`, `/metrics`.
- A&O provides `/status/overview` (fleet health) and `/slo/*` endpoints.

Example audit search response

```
{
  "items": [
    {
      "audit_id":"UUID",
      "at":"2025-11-08T08:15:22Z",
      "actor":{"type":"user","id":"UUID","roles":["finance"]},
      "action":"billing.refund.processed",
      "entity":{"type":"invoice","id":"INV-1234"},
      "result":{"status":"success"},
      "correlation_id":"req_4e2c...",
      "policy":{"consent_scopes":["billing.required"]},
      "source":{"ip":"192.0.2.1"}
    }
  ],
  "next_cursor": null
}
```

9) Dashboards (prebuilt)

- **Platform Overview:** uptime, error budget burn, active incidents, top offenders by error_rate.
- **Service Drill-downs:** latency histograms, dependency graphs, DB/Cache panels, queue depth.
- **Security/Audit:** auth failures by geo, role changes, API key usage, SAR actions timeline.
- **Billing Health:** payment failures, webhook retries, reconciliation lag.
- **Notifications Health:** deliverability, provider latency, bounce rate, template errors.

10) Security & Compliance

- **Access control:**
  - Audit queries restricted: Security/Privacy officers, Admins.
  - Developers see own service telemetry with PII masked.
  - Break-glass tokens (time-boxed) with audit of use.
- **Data protection:**
  - PII redaction in logs; explicit allowlist of fields.
  - Audit store encrypted at rest; optional tamper-evident hash chain.
  - Retention aligned to CPPC rules (e.g., audit 5y, logs 90d hot/365d archive).
- **Compliance hooks:**
  - SAR: return audit records related to user ID (filtered).

- Legal hold: suspend deletion of relevant audit rows.

11) Tech Stack

- **Collection:** OpenTelemetry SDK + Collector (OTLP).
- **Metrics:** Prometheus + Alertmanager; Grafana dashboards.
- **Logs:** Loki or OpenSearch (JSON).
- **Traces:** Tempo/Jaeger.
- **Store:** PostgreSQL 15 for audit/incidents; Kafka/Redis Streams for audit queue.
- **API:** FastAPI/NestJS service with RLS backed by JWT (scoped).
- **Infra:** NGINX/API Gateway, TLS 1.3, WAF, CDN for static dashboards (if needed).

12) Error Handling & Resilience

Case	Handling
Telemetry backend down	Buffer via OTel Collector; local file tail; backoff & alert
Audit DB slow/locked	Queue to Kafka/Redis; writer retries; DLQ after N attempts
Oversized payload	413; advise chunking; enforce size limits per endpoint
PII detected in logs	Redaction filter; raise security alert; open incident automatically
Corrupted trace spans	Drop span, keep request logs; mark partial trace
Alert storm	Auto-dedupe & rate-limit; single incident with updates

13) Test & Validation

- **Unit:** audit envelope validation, redaction functions, hash chain continuity.
- **Integration:** trace propagation across 3 services; metrics scraping; alert fire drill.
- **E2E:** simulate payment outage → alerts → incident → postmortem with linked artifacts.
- **Chaos/Resilience:** inject latency, drop dependencies; verify SLO burn and paging.

14) Future Enhancements

- **Tamper-evident ledger** (Merkle tree, periodic notarization).
- **Adaptive sampling** guided by anomaly detection.
- **Cost attribution** (per cohort/service) for FinOps.
- **User journey heatmaps** (privacy-safe) for UX diagnostics.
- **Automated change impact analysis** (correlate deploys to error spikes).

B) Learner Journey

Future You Profiling System (Technical Specification)

1) Module Overview

A multi-dimensional assessment and profiling engine that evaluates aptitude, personality, scenario judgment, and cybersecurity KSA (Knowledge–Skills–Abilities). It produces a standardized **Future You Profile** with role-readiness scores (e.g., SOC Analyst, Cloud Security Engineer), learning recommendations, and mentor-matching inputs.
Scope: authoring/question bank, assessment orchestration, secure test delivery, scoring/normalization, profile synthesis, recommendation engine, and APIs to other SMP modules (Mentorship, Portfolio, Analytics).

Adds value by: (a) standardizing selection and placement, (b) personalizing learning paths, (c) guiding mentors with evidence-based readiness metrics.

2) Functional Objectives

1. Orchestrate configurable assessments (aptitude, personality, scenario-based, KSA) per cohort/track.
2. Deliver secure, reliable test sessions (resume, timebox, randomization, proctoring signals).
3. Score and normalize results; compute trait vectors and role-readiness indices.
4. Generate a machine-readable **Future You Profile** for downstream modules.
5. Produce recommendations (courses, missions, mentors) with explicit rationale.
6. Support admin analytics: cohort distributions, reliability indicators, completion funnels.
7. Enforce RBAC, audit trails, data protection, and export controls.
8. Provide REST APIs + webhooks for SMP and external consumers.

3) User Roles & Permissions

Role	Create	View	Edit	Approve	Delete
Student	Assessment attempt	Own profile, results, recommendations	Own demographic/profile fields	—	Own attempt drafts (pre-submit)
Mentor	Feedback notes on mentee	Mentee profiles assigned	Feedback notes	—	Own notes

Role	Create	View	Edit	Approve	Delete
Program Director	Assessment configs (with Admin), cohorts	All cohort analytics	Cohort settings	Publish assessment windows	Deactivate configs
Admin	Question bank, rubrics, roles matrix	All	All	Finalize/publish versions	Remove/deprecate items
Finance (FYI)	—	Subscription/seat usage metrics	—	—	—
Analyst (Data)	Derived datasets	Aggregates & exports	Data pipelines (non-PII)	—	Temp exports

4) Core Functional Requirements

Grouped features. Each includes story, acceptance, deps, I/O.

4.1 Assessment Authoring & Versioning

- **Description:** Create/edit question banks (aptitude, personality, situational judgment tests—SJT, KSA), rubrics, and role matrices. Versioned publishes.
- **User Story:** As an Admin, I can create item banks with metadata and publish immutable versions used in live assessments.
- **Acceptance Criteria:**
  - CRUD on questions with types: MCQ single/multi, Likert, ranking, matrix, free-text (scored via rubric), coding (external grader), file-upload (disabled by default).
  - Item metadata: domain, difficulty, discrimination, time_hint, randomization_group, tags (role/track), locale.
  - Versioning: published versions locked; draft copies supported.
  - Rubrics: scale definitions, weights, anchors.
- **Dependencies:** DB, storage, auth, audit log.
- **I/O:** Inputs: JSON payloads for items/rubrics. Outputs: item_ids, version_ids.

4.2 Assessment Blueprint & Orchestration

- **Description:** Define assessment bundles per cohort/track with timing, section weights, and eligibility.
- **User Story:** As Program Director, I configure “Jan-2026 Core Profile” with four sections, 75 min total, randomized items.
- **Acceptance Criteria:**

    - Blueprint = ordered sections with item pools/constraints (e.g., 10±2 aptitude_MCQs from pool A).
    - Timeboxes per section and global; mandatory breaks configuration.
    - Eligibility rules (new joiners, re-takes after X days, fee paid).
    - Publish window (start/end), timezone aware.
- **Dependencies:** Authoring, payments, cohorts.
- **I/O:** Blueprint JSON; schedule metadata.

4.3 Secure Test Delivery (Session Engine)

- **Description:** Reliable, resumable test runner with anti-cheat signals.
- **User Story:** As a Student, I can complete a timed assessment with autosave and resume within window.
- **Acceptance Criteria:**
    - Autosave every 10s or on change; resume tokens; offline tolerance (local cache + sync).
    - Randomization: item order, option order, parallel forms.
    - Soft proctoring signals: tab blur count, focus loss, fast switching, IP changes (log only).
    - Accessibility: keyboard navigation, ARIA roles, font scaling.
- **Dependencies:** Frontend, session store (Redis), CDN.
- **I/O:** Attempts, responses, proctoring events.

4.4 Scoring & Normalization Engine

- **Description:** Score raw responses, apply rubrics, normalize across forms, compute trait vectors.
- **User Story:** As Admin, I get normalized T-scores and reliability per scale; as Student, I see clear results.
- **Acceptance Criteria:**
    - Scoring per item type; partial credit; negative marking optional.
    - Psychometrics: z/T-score normalization; percentile by cohort; internal consistency (Cronbach's α).
    - Composite scales: Aptitude, Personality (Big Five/HEXACO-like), Judgment, KSA per domain.
- **Dependencies:** Authoring metadata, analytics pipeline.
- **I/O:** score_breakdown, normalized_scores, reliability_indicators.

4.5 Role Readiness Calculator

- **Description:** Map trait vector → role readiness per target roles via **Role–Competency Matrix**.
- **User Story:** As a Program Director, I see readiness per role and gaps at competency level.
- **Acceptance Criteria:**
    - Matrix of role → competencies → required thresholds/weights.

    - Output readiness 0–100 with gap deltas and confidence scores.
- **Dependencies:** Role matrix, scoring outputs.
- **I/O:** readiness_by_role[], gap_analysis[].

4.6 Recommendation Engine

- **Description:** Produce prioritized learning and mentoring recommendations with rationale.
- **User Story:** As a Student, I get a ranked list of actions tied to my gaps.
- **Acceptance Criteria:**
    - Rules + (optional) ML ranking: map gaps → missions/courses/activities; deduplicate and justify.
    - Include effort_estimate, pre-requisites, links to external LMS or OCH missions.
- **Dependencies:** Course catalog (external + curated), missions registry, mentor pool.
- **I/O:** recommendations[], rationale_text.

4.7 Mentor Matching Signal Generator

- **Description:** Generate features for mentor–mentee matching (not final match UI).
- **User Story:** As Mentor Coordinator, I get candidate mentees ranked by fit.
- **Acceptance Criteria:**
    - Fit score features: role_interest, gaps_to_mentor_strengths, time_zone, availability.
    - Expose via API for Mentorship module consumption.
- **Dependencies:** Mentor profiles, availability service.
- **I/O:** matching_features per student.

4.8 Profile Synthesis & Delivery

- **Description:** Build the canonical **Future You Profile** JSON + PDF summary.
- **User Story:** As Student, I can download a well-structured profile; as Admin, I can embed it in dashboards.
- **Acceptance Criteria:**
    - Canonical JSON schema; versioned.
    - Optional PDF with charts (radar for traits; bar for readiness).
    - Consent gates for sharing with mentors/recruiters.
- **Dependencies:** Storage (S3), renderer.
- **I/O:** profile_json_uri, profile_pdf_uri.

4.9 Admin Analytics & Exports

- **Description:** Cohort distributions, funnels, test health.
- **User Story:** As Program Director, I view completion rates, trait histograms, role readiness by cohort.
- **Acceptance Criteria:**

    - Dashboards: completion %, average readiness by role, distribution charts, reliability.
    - CSV/Parquet exports with PII redaction options.
- **Dependencies:** Analytics warehouse, BI embed.
- **I/O:** dashboard widgets, export files.

4.10 Compliance, Audit, & Consent

- **Description:** Explicit consents, audit trails, data retention rules.
- **User Story:** As Admin, I can evidence consent and access logs for any profile.
- **Acceptance Criteria:**
    - Consent capture with timestamps and purpose.
    - Row-level audit (who viewed/exported what).
    - Retention schedule (e.g., 36 months unless renewed).
- **Dependencies:** Auth, logging, key management.
- **I/O:** consent_records, audit_events.

5) Data Model & Key Entities (PostgreSQL)

Types indicative; adjust to final ORM.

5.1 Core Tables

- **users**(id UUID PK, email TEXT unique, role ENUM, status ENUM, tz TEXT, created_at TIMESTAMP)
- **cohorts**(id UUID PK, name TEXT, track ENUM, start_date DATE, end_date DATE)
- **assessments**(id UUID PK, name TEXT, version INT, status ENUM[draft,published,retired], created_by UUID FK users)
- **assessment_sections**(id UUID, assessment_id FK, name TEXT, time_limit_sec INT, weight NUMERIC)
- **questions**(id UUID PK, assessment_id FK, type ENUM[mcq,likert,rank,matrix,free,coding], body JSONB, metadata JSONB, version INT)
- **options**(id UUID PK, question_id FK, label TEXT, value JSONB, is_correct BOOL nullable)
- **rubrics**(id UUID PK, question_id FK, scale JSONB, weights JSONB)
- **blueprints**(id UUID PK, assessment_id FK, constraints JSONB, eligibility JSONB, window_start TIMESTAMP, window_end TIMESTAMP)
- **attempts**(id UUID PK, user_id FK, assessment_id FK, status ENUM[in_progress,submitted,scored], started_at, submitted_at, proctor_signals JSONB)
- **responses**(id UUID PK, attempt_id FK, question_id FK, answer JSONB, time_spent_sec INT)
- **scores**(id UUID PK, attempt_id FK, raw JSONB, normalized JSONB, reliability JSONB)
- **traits**(id UUID PK, attempt_id FK, vector JSONB) // e.g., {aptitude:72, conscientiousness:64, ...}
- **roles**(id UUID PK, name TEXT unique, description TEXT)

- **competencies**(id UUID PK, name TEXT, category TEXT)
- **role_competency_matrix**(role_id FK, competency_id FK, weight NUMERIC, min_threshold NUMERIC, PRIMARY KEY(role_id, competency_id))
- **readiness**(id UUID PK, user_id FK, role_id FK, readiness NUMERIC, gaps JSONB, computed_at TIMESTAMP)
- **recommendations**(id UUID PK, user_id FK, items JSONB, rationale TEXT, generated_at TIMESTAMP)
- **mentor_features**(id UUID PK, user_id FK, features JSONB, generated_at TIMESTAMP)
- **future_you_profiles**(id UUID PK, user_id FK, json_uri TEXT, pdf_uri TEXT, version INT, consent JSONB)
- **audit_logs**(id UUID PK, actor_id FK, entity TEXT, entity_id UUID, action TEXT, meta JSONB, at TIMESTAMP)

Validation & Security:

- PII columns encrypted at rest (pgcrypto or KMS-managed).
- Consent JSON includes `{purpose, granted_at, expires_at}`.
- Soft-delete via `deleted_at` on authoring tables.

6) Process Flow (Happy Path)

1. **Configure**: Admin authors items → builds blueprint → publishes version.
2. **Schedule**: Program Director sets window/eligibility for cohort.
3. **Start**: Student starts assessment → session token issued → autosave.
4. **Submit**: Student submits → lock attempt.
5. **Score**: Scoring engine computes raw → normalized → trait vector; reliability metrics.
6. **Readiness**: Map to roles via matrix → compute readiness and gaps.
7. **Recommend**: Generate ordered recommendations with rationale.
8. **Synthesize**: Build Future You Profile (JSON + PDF) → store URIs.
9. **Expose**: Mentorship module fetches mentor_features; Analytics ingests aggregates.
10. **Notify**: Student + Program Director receive results/insights as configured.

Exceptions: window expired, retry on scoring failure (exponential backoff), partial uploads, integrity checks on version drift.

7) Integrations & APIs

Internal REST (prefix `/api/v1/profiling`)

- `POST /assessments` (Admin): create draft.
- `POST /assessments/{id}/publish` (Admin): lock version.
- `POST /blueprints` (Director/Admin): define orchestration.
- `POST /attempts` (Student): start attempt (returns session token).
- `PUT /attempts/{id}`: autosave/submit. Body: `{status, responses[]}`.

- `POST /attempts/{id}/score` (svc): trigger scoring (queued).
- `GET /profiles/{user_id}`: get Future You Profile (authz enforced).
- `GET /readiness/{user_id}`: role readiness + gaps.
- `GET /recommendations/{user_id}`.
- `GET /mentor-features/{user_id}`.
- `GET /analytics/cohort?cohort_id=...` (Director/Admin).

Webhooks

- `profiling.profile_published`
- `profiling.scored`
- `profiling.readiness_computed`
- `profiling.recommendations_ready`

Payload Example (Profile JSON excerpt)

```
{
  "version": 3,
  "user_id": "UUID",
  "attempt_id": "UUID",
  "traits": {"aptitude": 71, "conscientiousness": 63, "judgment": 68,
"ksa_network": 54},
  "readiness": [
    {"role": "SOC Analyst", "score": 72, "gaps": [{"competency":"L1
Triage","delta":-8}]}
  ],
  "recommendations": [
    {"type":"mission","id":"M-REDTEAM-01","title":"Intro to ATT&CK
TTPs","effort_hours":6,"rationale":"Gap: Detection Engineering"}
  ],
  "consent": {"share_with_mentor": true, "granted_at": "2025-11-
07T09:00:00Z"}
}
```

8) UI/UX Requirements

Student

- Pages: “Start Assessment”, “Resume”, “Results & Insights”, “My Profile”.
- Components: timebox header, progress bar, autosave indicator, accessibility controls.
- Visualizations: radar (traits), bars (readiness/gaps), cards (recommendations).

Admin/Director

- Authoring studio (tables + inline editors), blueprint wizard, publish flow.
- Dashboards: completion funnel, trait distributions, role readiness heatmap.
- Exports: CSV download with PII toggle.

Responsive & A11y

- Mobile-friendly runner, min 44px touch targets, WCAG 2.1 AA color contrast, ARIA roles.

9) Notifications & Automation

Event	Audience	Channel	Timing
Assessment window opens	Students	Email + In-app	T-24h & T-1h
Incomplete attempt reminder	Students	In-app	48h before close
Scoring completed	Student, Director	Email + In-app	Immediately
Profile published/updated	Student, Mentor (if consent)	In-app	Immediately
Cohort analytics weekly digest	Director	Email (PDF snapshot)	Weekly Mon 08:00

10) Analytics & Reporting

KPIs

- Start→Submit conversion %, median time on task, reliability (α) per scale.
- Readiness by role (avg, p25–p75), top 3 gap competencies per cohort.
- Recommendation uptake (click-through, completion via LMS callbacks).
- Mentor match success proxy (downstream).

Data refresh: near-real-time for attempts; hourly for cohort aggregates.
Retention: raw attempts 36 months; aggregates indefinite.

11) Security & Compliance

- RBAC + least privilege; attribute-based checks on `user_id`.
- TLS 1.3, HSTS; JWT with short TTL + refresh; rotate signing keys.
- PII/consent encryption (KMS); field-level for sensitive traits if needed.
- Audit every read/export on profiles; tamper-evident logs.
- Botswana DPA alignment: explicit consent, purpose limitation, data minimization, subject access & delete requests.

12) Technical Dependencies

- **Frontend:** React, TanStack Router, React Query, Zustand.
- **Backend:** FastAPI or Node.js (NestJS), Celery/BullMQ for async scoring.
- **DB:** PostgreSQL 15, Redis (sessions/queues), S3 for artifacts.
- **CI/CD:** GitHub Actions; IaC via Terraform; containerized deploys.

- **Observability:** OpenTelemetry, Prometheus, Grafana; Sentry for FE/BE.

13) Error Handling & Exceptions

Case	Response
Window expired	409 with actionable message; link to next window
Token invalid/expired	401; refresh flow
Autosave failure	Local cache + retry (exponential backoff up to 5x), user banner
Scoring job crash	Requeue up to 3; on failure mark `needs_review` and alert Admin
Version drift (blueprint changed mid-attempt)	Continue with pinned snapshot; flag in audit
Data export permission fail	403 + audit entry

14) Future Enhancements

- IRT (Item Response Theory) calibration, adaptive testing.
- Behavioral telemetry (keystroke/interaction) opt-in for deeper insights.
- ML-based mentor matching and success prediction.
- Multilingual item banks with DIF (bias) checks.

Portfolio Management System (Technical Specification)

1) Module Overview

A structured repository for each learner's **evidence of work** (projects, labs, missions, certifications, research, presentations, write-ups, awards) with review workflows, scoring rubrics, and a **public, privacy-controlled portfolio page** (optional) at `och.africa/student/<handle>`. It ingests artifacts from external sources (GitHub, GitLab, TryHackMe, HackTheBox, Coursera, Google Drive) and links them to OCH missions/assessments. It feeds signals into Mentorship, Analytics, and Employer/Partner showcases.

Value:

- Converts learning into verifiable outcomes.
- Standardizes review and scoring for comparability across cohorts.
- Powers mentor feedback and hiring-ready showcases.

2) Functional Objectives

1. Capture artifacts from multiple sources (uploads, links, API imports) with strong metadata.
2. Provide versioned **Portfolio Items** tied to missions, roles, and competencies.
3. Enable rubric-based reviews and approvals by mentors/reviewers.
4. Offer a configurable **public portfolio** with granular visibility.
5. Generate portfolio health and progress analytics at student/cohort levels.
6. Expose canonical JSON for downstream consumers (Mentorship, Employer Access).
7. Enforce RBAC, consent, audit, and retention policies.

3) User Roles & Permissions

Role	Create	View	Edit	Approve/Score	Delete/Archive
Student	Draft items, uploads, link artifacts	Own items (all states)	Own drafts & privacy settings	—	Own drafts; request archive of approved items
Mentor	Feedback notes	Assigned mentee portfolios	Feedback notes; request changes	Approve with rubric (if assigned as reviewer)	—

Role	Create	View	Edit	Approve/Score	Delete/Archive
Program Director	Portfolio templates, rubric sets	All cohort portfolios	Edit templates; reassign reviewers	Final approve; override states	Archive items (policy)
Admin	System configs, tags, taxonomies	All	All	All	Hard delete (exception process)
Employer (future)	—	Public pages only (if permitted)	—	—	—

4) Core Functional Requirements

4.1 Portfolio Item Creation & Metadata

- **Description:** Create structured items: project, lab report, certification, talk/paper, reflection, case study.
- **User Story:** As a Student, I can submit a new item with files/links, tag it to missions/competencies, and save as draft.
- **Acceptance Criteria:**
  - Item types with required fields by type (e.g., Certification → issuer, credentialId, verificationUrl, issueDate).
  - Multiple artifacts per item (files/links), drag-and-drop, max size configurable.
  - Tags: mission(s), role(s), competency(ies), tools/tech stack, keywords.
  - Status lifecycle: `draft → submitted → in_review → changes_requested → approved → published`.
- **Dependencies:** Storage (S3), taxonomies, mission registry.
- **I/O:** Inputs: form JSON + files. Outputs: `portfolio_item_id`.

4.2 Artifact Ingestion (External)

- **Description:** Link or import artifacts from GitHub, GitLab, THM/HTB, Coursera badges, Drive, Vimeo/YouTube (talks).
- **User Story:** As a Student, I can connect providers and import selected artifacts with metadata prefilled.
- **Acceptance Criteria:**
  - OAuth links to providers; scoped read permissions.
  - Normalizers per provider (e.g., GitHub repo → name, stars, last_commit_at).
  - Deduping by canonical URL or provider object ID.
- **Dependencies:** Integration layer, provider SDKs, secrets vault.
- **I/O:** Provider tokens; normalized artifact JSON.

4.3 Templates, Checklists & Evidence Requirements

- **Description:** Program Director configures templates per item type to guide submissions.
- **User Story:** As a Director, I set required sections and checklists for "Mission Case Study".
- **Acceptance Criteria:**
  - Markdown/JSON templates with placeholders.
  - Validation rules (min fields, word counts, required attachments).
  - Per-cohort overrides.
- **Dependencies:** Template service.
- **I/O:** Template JSON versioned.

4.4 Review Workflow & Rubric Scoring

- **Description:** Assign reviewer(s); review inline; score via rubric; capture feedback and change requests.
- **User Story:** As a Mentor, I review, score, and either approve or request changes.
- **Acceptance Criteria:**
  - Multi-reviewer support (avg or weighted).
  - Rubrics: criteria with levels (e.g., 1–5), weights; auto compute total & grade.
  - Comments with anchored references to sections/files.
  - Lock on approval; versioning retained.
- **Dependencies:** Reviewer assignment, notifications.
- **I/O:** `review_id`, rubric scores JSON, decision.

4.5 Public Portfolio Page & Privacy Controls

- **Description:** Publish selected items to a public page; per-item visibility (private/unlisted/public).
- **User Story:** As a Student, I can curate my public page with a cover, bio, featured items, and custom slug.
- **Acceptance Criteria:**
  - Page generator with theme presets; SEO meta; OpenGraph.
  - Visibility matrix: owner, mentors, OCH staff, public.
  - Consent gate for employer access (future).
- **Dependencies:** Static generator/CDN, consent service.
- **I/O:** `public_url`, sitemap entry.

4.6 Mission & Competency Mapping

- **Description:** Attach items to OCH missions and competencies; compute cumulative competency evidence scores.
- **User Story:** As a Director, I see cohort competency coverage and gaps based on approved items.
- **Acceptance Criteria:**
  - N:M mapping item↔mission, item↔competency.
  - Evidence score = rubric_total × weight_by_item_type × reviewer_confidence.
  - Competency heatmaps per student/cohort.

- **Dependencies:** Missions registry, competency taxonomy.
- **I/O:** evidence_aggregates.

4.7 Portfolio Health & Readiness Signals

- **Description:** Compute **portfolio health score** and feed signals to Mentorship & Analytics.
- **User Story:** As a Mentor, I see mentee's health score trends and missing evidence areas.
- **Acceptance Criteria:**
  - Health score combining: item volume, diversity, approvals %, evidence coverage, recency, external validation (stars/badges).
  - Expose `/signals/portfolio-health` API per user.
- **Dependencies:** Analytics engine, scoring config.
- **I/O:** `health_score`, `coverage`, trend metrics.

4.8 Certificates & Verifiable Claims

- **Description:** Store and display certifications; verify via issuer link or uploaded PDF with hash.
- **User Story:** As a Student, I add my CompTIA Security+ with verification link.
- **Acceptance Criteria:**
  - Credential fields, issuer catalog, verification checker (HTTP 200 & domain match).
  - Optional PDF hashing; integrity check on access.
- **Dependencies:** Issuer registry.
- **I/O:** `credential_id`, `verification_status`.

4.9 Exports & Showcase Packs

- **Description:** Generate downloadable packs (ZIP/PDF) or sharable showcase links for applications.
- **User Story:** As a Student, I export a curated pack for an internship application.
- **Acceptance Criteria:**
  - Selection UI; export manifest JSON; ZIP with files + PDF summary.
  - Time-boxed share links with access logs.
- **Dependencies:** Renderer, storage, signed URLs.
- **I/O:** `export_url`, `expires_at`.

4.10 Compliance, Audit & Retention

- **Description:** Consent capture, audit for views/exports, retention rules by item type.
- **User Story:** As Admin, I can evidence who accessed which portfolio and when.
- **Acceptance Criteria:** Full audit trail; retention policy engine; SAR (subject access request) support.

5) Data Model & Key Entities (PostgreSQL)

5.1 Core Tables

- **portfolio_items**(id UUID PK, user_id UUID FK, type ENUM[project,lab,cert,talk,paper,case_study,reflection,other], title TEXT, summary TEXT, content_md TEXT, status ENUM[draft,submitted,in_review,changes_requested,approved,published], visibility ENUM[private,unlisted,public], slug TEXT unique nullable, template_version INT, created_at, updated_at, published_at, tags TEXT[], mission_ids UUID[] FK, competency_ids UUID[] FK)
- **artifacts**(id UUID PK, portfolio_item_id FK, kind ENUM[file,link,embed,provider], url TEXT, file_uri TEXT, provider ENUM[github,gitlab,thm,htb,coursera,drive,youtube,vimeo,other], provider_ref TEXT, metadata JSONB, checksum TEXT)
- **reviews**(id UUID PK, portfolio_item_id FK, reviewer_id UUID FK, status ENUM[pending,approved,changes_requested], comments_md TEXT, rubric_scores JSONB, total_score NUMERIC, confidence NUMERIC, decided_at TIMESTAMP)
- **rubrics**(id UUID PK, name TEXT, item_type ENUM, criteria JSONB, weights JSONB, version INT, status ENUM[draft,published])
- **public_profiles**(user_id UUID PK, slug TEXT unique, bio_md TEXT, headline TEXT, avatar_uri TEXT, banner_uri TEXT, theme JSONB, is_enabled BOOL, visibility ENUM[private,public])
- **evidence_aggregates**(id UUID PK, user_id UUID, competency_id UUID, evidence_score NUMERIC, items_count INT, last_updated TIMESTAMP)
- **credentials**(id UUID PK, user_id UUID, issuer_id UUID FK, name TEXT, credential_id TEXT, issue_date DATE, expiry_date DATE, verification_url TEXT, verification_status ENUM[pending,verified,failed], document_uri TEXT, doc_sha256 TEXT)
- **issuer_registry**(id UUID PK, name TEXT, domain TEXT, api_base TEXT, verification_mode ENUM[link,api,manual])
- **exports**(id UUID PK, user_id UUID, selection JSONB, url TEXT, expires_at TIMESTAMP, access_policy JSONB)
- **audit_logs**(id UUID PK, actor_id UUID, entity TEXT, entity_id UUID, action TEXT, meta JSONB, at TIMESTAMP)

Validation & Security

- PII minimal on public pages (no email/phone unless explicitly opted-in).
- File scanning (AV) on upload; max size & type whitelist.
- Hash artifacts for integrity; signed URLs for private downloads.

6) Process Flow

Happy path (Project item):

1. Student **creates draft** → selects type “project”, fills metadata, attaches repo link & screenshots.
2. Student **submits** → reviewer auto-assigned (mentor) or by Director.
3. Reviewer **scores via rubric**, leaves comments → approves or requests changes.
4. On **approval**, item status becomes `approved`; student can **publish** (if public profile enabled).
5. Evidence scores update **competency aggregates**; **health score** recalculated.
6. Public page updates with featured item (if chosen).

Exceptions: invalid file type (block), provider OAuth revoked (prompt re-auth), duplicate artifact (warn & prevent), reviewer timeout (auto-reassign after N days).

7) Integrations & APIs

REST (prefix `/api/v1/portfolio`)

- `POST /items` (Student): create draft.
- `PUT /items/{id}` (Student): update draft.
- `POST /items/{id}/submit` (Student).
- `POST /items/{id}/reviewers` (Director/Admin): assign.
- `POST /items/{id}/reviews` (Reviewer): submit rubric & decision.
- `POST /items/{id}/publish` (Student/Director per policy).
- `POST /artifacts` (Student): attach file/link/provider.
- `GET /users/{id}/portfolio` (authz): list items by visibility.
- `GET /public/{slug}`: public profile and published items.
- `GET /signals/portfolio-health?user_id=....`
- `POST /exports` (Student): create showcase pack.
- `GET /exports/{id}`: signed link.

Provider Webhooks

- `github.repo_updated`, `coursera.badge_earned`, `drive.file_changed` (ingest & refresh metadata).

Provider OAuth Scopes (examples)

- GitHub: `read:user`, `repo:read`.
- Google Drive: `drive.readonly`.
- Coursera/THM: read badges & progress if available.

Example: POST /items (request)

```
{
  "type": "project",
  "title": "DNS Sinkhole on pfSense",
  "summary": "Built a network-wide DNS sinkhole...",
```

```
  "content_md": "## Objective\nBlock adware ...",
  "missions": ["UUID-MISSION-NET-01"],
  "competencies": ["UUID-COMP-NET", "UUID-COMP-IR"],
  "tags": ["pfSense","DNS","network"],
  "visibility": "private"
}
```

8) UI/UX Requirements

Student

- Portfolio dashboard with **Health score**, **Coverage heatmap**, “Add Item” CTA.
- Editor: rich Markdown with template injection; artifact panel (files, links, providers).
- Review status timeline; publish switch with visibility selection.
- Public page builder: bio, avatar/banner upload, select featured items, preview.

Reviewer

- Review queue; item reader with side-by-side **artifact viewer**; rubric panel; approve/request changes.

Director/Admin

- Cohort overview: approval rates, evidence coverage by competency, reviewer SLA tracking.
- Template/rubric managers; taxonomy editors.

Accessibility & Responsiveness

- Mobile-first authoring supported; alt text for images; keyboard navigation; WCAG 2.1 AA.

9) Notifications & Automation

Trigger	Audience	Channel	Rule
Item submitted	Assigned reviewer	In-app + email	Immediate
Review overdue	Reviewer, Director	In-app + email	72h SLA
Changes requested	Student	In-app	Immediate
Item approved	Student	In-app + email	Immediate
Public page enabled	Student	In-app	Confirmation with tips
Monthly digest	Student	Email	Portfolio health & suggestions

10) Analytics & Reporting

KPIs

- Items per student; approval rate; time-to-approval; reviewer SLA %.
- Competency coverage (breadth/depth); portfolio health distribution.
- External validation metrics (GitHub stars, THM/HTB ranks).
- Public engagement (views, link clicks) for published pages.

Dashboards

- Student: health trend, coverage heatmap, reviewer feedback summary.
- Cohort: approval funnel, competency heatmap, outlier detection (high/low evidence).

Refresh

- Near real-time for item status; hourly for aggregates.

11) Security & Compliance

- RBAC & ABAC (item owner; mentor of record; cohort director).
- File AV scanning; MIME/type enforcement; content sanitization on embeds.
- Signed URLs, short TTL for private downloads; CDN for public assets.
- Consent for public publishing; prominent privacy controls.
- Botswana DPA/GDPR alignment; right to be forgotten on private items and public pages.
- Audit logs for all reads/exports of non-public items.

12) Technical Dependencies

- **Frontend:** React, React Hook Form, TipTap/Markdown editor, UploadCare/Uppy for uploads.
- **Backend:** FastAPI/NestJS, Celery/BullMQ for async exports, FFmpeg (video thumbnails optional).
- **Storage:** S3 (versioned buckets), CloudFront CDN, ClamAV for scanning.
- **DB/Cache:** PostgreSQL 15, Redis.
- **Search (optional):** OpenSearch/Meilisearch for items & tags.
- **Observability:** OpenTelemetry, Prometheus/Grafana, Sentry.

13) Error Handling & Exceptions

Case	Handling
File too large/unsupported	413 with allowed types/sizes; UI helper text
Provider OAuth revoked	401 on provider calls; prompt re-connect

Case	Handling
Duplicate artifact URL	409 with option to reference existing
Reviewer timeout	Auto-reassign after SLA breach; notify Director
Publishing without consent	Block with modal capturing explicit consent
Export job failure	Retry ×3; notify Student with error trace ID

14) Future Enhancements

- **Verifiable credentials** via OpenBadges/VCs with on-chain hash (optional).
- **Auto-summaries** of projects via LLM (privacy-aware).
- **Employer showcase portal** with structured role filters and shortlisting.
- **Smart prompts** suggesting evidence items based on LMS progress & profiling gaps.

Mentorship Management Module (Technical Specification)

1) Module Overview

The **Mentorship Management Module (MMM)** orchestrates structured mentor–mentee relationships, sessions, milestones, feedback loops, and progress tracking across all OCH cohorts.
It automates mentor assignments (based on profiling data), schedules sessions, records notes, tracks outcomes, and integrates results into the student's portfolio and overall progress dashboard.

This module serves as the **interaction layer** between students and mentors, powered by data from:

- The **Future You Profiling System** (for matching),
- The **Portfolio Management System** (for evidence tracking),
- And the **Program Calendar** (for session scheduling and reporting).

Value Added:

- Enables personalized, measurable mentorship experiences.
- Standardizes progress tracking and mentor accountability.
- Feeds continuous improvement data to analytics dashboards and cohort management.

2) Functional Objectives

1. Automate mentor–mentee pairing using profiling and track data.
2. Provide structured session scheduling, attendance, and note-taking.
3. Enable milestone-based progress tracking and goal setting.
4. Allow two-way feedback and satisfaction scoring.
5. Integrate outputs into the Portfolio and Analytics modules.
6. Generate mentor performance and engagement reports.
7. Manage mentor onboarding, capacity, and assignments.
8. Maintain secure, compliant communication records.

3) User Roles & Permissions

Role	Create	View	Edit	Approve/Close	Delete/Archive
Student	Goals, session requests, feedback	Own sessions, feedback	Own goals and updates	Close session with feedback	Own goals (draft only)

Role	Create	View	Edit	Approve/Close	Delete/Archive
Mentor	Session notes, feedback, milestone updates	Assigned mentees	Session notes & feedback	Close or approve milestone	—
Program Director	Assign mentors, define milestones, configure cycles	All mentorship data	Reassign mentors, adjust goals	Approve or finalize mentorship cycles	Archive records
Admin	System settings, mentor capacity, data exports	All	All	—	Hard delete (policy exceptions)

4) Core Functional Requirements

4.1 Mentor–Mentee Matching Engine

- **Description:** Automated pairing based on profile compatibility, availability, and preferences.
- **User Story:** As a Director, I can auto-generate optimal mentor–mentee matches or override manually.
- **Acceptance Criteria:**
  - Input data: `profiling.matching_features`, track, timezone, mentor capacity, preferences.
  - Matching algorithm: weighted scoring (skills overlap 50%, career alignment 30%, personality 20%).
  - Allow manual overrides and pair-locking.
- **Dependencies:** Profiling module, mentor registry.
- **I/O:** `match_id`, `score`, `rationale`.

4.2 Mentorship Cycle & Cohort Setup

- **Description:** Define mentorship duration, milestones, and goals per cohort or specialization.
- **User Story:** As a Director, I configure a 3-month mentorship with 6 biweekly sessions and defined milestones.
- **Acceptance Criteria:**
  - Configurable duration, frequency, milestones, and goals per track.
  - Default templates by program type (e.g., Builders, Leaders).
  - Auto-generate mentor–mentee calendars.
- **Dependencies:** Scheduling service, cohort registry.
- **I/O:** `cycle_id`, milestone template JSON.

4.3 Session Scheduling & Management

- **Description:** Schedule and manage mentorship sessions (physical or virtual).
- **User Story:** As a Student, I request a session with my mentor; as a Mentor, I confirm or reschedule.
- **Acceptance Criteria:**
  - Real-time calendar integration (Google/Outlook).
  - Session states: `scheduled → confirmed → completed → feedback_pending → closed`.
  - Time-zone normalization; prevent conflicts.
  - Optional location field (onsite or link).
- **Dependencies:** gCal integration, notification system.
- **I/O:** `session_id`, session metadata.

4.4 Session Notes & Record-Keeping

- **Description:** Record structured discussion notes, next steps, and insights.
- **User Story:** As a Mentor, I record session notes and flag key takeaways.
- **Acceptance Criteria:**
  - Note templates: recap, challenges, wins, next actions.
  - File attachments (e.g., screenshots, reports).
  - Editable by Mentor (lock after session closed).
  - Tagged to milestones/goals.
- **Dependencies:** Storage, session engine.
- **I/O:** `session_notes_id`, `notes_json`.

4.5 Goal & Milestone Tracking

- **Description:** Define and track measurable goals for each mentee.
- **User Story:** As a Student, I create goals aligned with milestones; as a Mentor, I review progress.
- **Acceptance Criteria:**
  - SMART goals (title, metric, target, timeline).
  - Milestone linkage (Goal → Milestone).
  - Status lifecycle: `draft → active → achieved → verified`.
  - Progress input types: numeric, checklist, narrative.
- **Dependencies:** Milestone templates, analytics.
- **I/O:** `goal_id`, `progress_log`.

4.6 Two-Way Feedback System

- **Description:** Post-session and post-cycle feedback for both sides.
- **User Story:** As a Student, I rate my mentor; as a Mentor, I evaluate mentee engagement.
- **Acceptance Criteria:**
  - Standardized forms (5-point scale + open comments).
  - Aggregate ratings visible to Director.

    - Anonymized feedback option for sensitive topics.
- **Dependencies:** Notification engine, analytics.
- **I/O:** `feedback_id`, `rating`, `comments`.

4.7 Mentor Performance Analytics

- **Description:** Measure mentor effectiveness and engagement.
- **User Story:** As a Director, I track session completion rate, feedback average, and impact score.
- **Acceptance Criteria:**
    - Metrics: attendance %, session quality, mentee satisfaction, goal achievement ratio.
    - Rank mentors and flag underperformers (below SLA).
    - Export to analytics engine.
- **Dependencies:** Feedback data, sessions table.
- **I/O:** `mentor_performance_summary`.

4.8 Communication & Notifications

- **Description:** Facilitate structured, auditable communication (chat + notifications).
- **User Story:** As a Mentor, I message my mentee securely within SMP.
- **Acceptance Criteria:**
    - In-app chat (optional) with file sharing.
    - Email/SMS reminders for upcoming sessions, overdue feedback, or milestones.
    - Auto-archive after mentorship closure.
- **Dependencies:** Messaging microservice, notification service.
- **I/O:** `message_id`, `notification_log`.

4.9 Cohort & Cycle Closure

- **Description:** Finalize mentorship cycles and generate closure reports.
- **User Story:** As a Director, I close a cohort's mentorship cycle and generate analytics for the report.
- **Acceptance Criteria:**
    - Ensure all feedback submitted; flag missing sessions.
    - Generate cycle report (PDF + JSON) summarizing engagement and impact.
    - Lock cycle records post-closure.
- **Dependencies:** Analytics engine, reporting service.
- **I/O:** `cycle_report_uri`, `closure_log`.

4.10 Compliance & Audit

- **Description:** Maintain privacy, consent, and audit for mentorship data.
- **User Story:** As Admin, I can evidence who accessed mentorship data and when.
- **Acceptance Criteria:**
    - Explicit consent for data sharing with mentors.

- Audit logs for all record access.
- Configurable retention (default: 24 months).

5) Data Model & Key Entities (PostgreSQL)

Table	Description
mentors(id UUID PK, user_id UUID FK, expertise TEXT[], capacity INT, active BOOL, timezone TEXT, rating NUMERIC, joined_at TIMESTAMP)	
mentees(id UUID PK, user_id UUID FK, track ENUM, cohort_id UUID, profile_id UUID FK, current_mentor_id UUID FK)	
matches(id UUID PK, mentor_id FK, mentee_id FK, score NUMERIC, rationale TEXT, matched_by ENUM[auto,manual], matched_at TIMESTAMP, status ENUM[active,inactive,completed])	
mentorship_cycles(id UUID PK, name TEXT, cohort_id UUID FK, start_date DATE, end_date DATE, milestones JSONB, config JSONB, status ENUM[active,closed])	
sessions(id UUID PK, match_id FK, date_start TIMESTAMP, date_end TIMESTAMP, status ENUM[scheduled,confirmed,completed,feedback_pending,closed], location TEXT, link TEXT, notes_id UUID, created_at TIMESTAMP)	
session_notes(id UUID PK, session_id FK, mentor_id FK, notes_md TEXT, attachments JSONB, created_at TIMESTAMP)	
goals(id UUID PK, mentee_id FK, title TEXT, metric TEXT, target NUMERIC, progress NUMERIC, milestone TEXT, status ENUM[draft,active,achieved,verified], last_updated TIMESTAMP)	
feedbacks(id UUID PK, session_id FK, from_role ENUM[mentor,mentee], rating NUMERIC, comments TEXT, created_at TIMESTAMP)	
mentor_performance(id UUID PK, mentor_id FK, sessions_completed INT, avg_rating NUMERIC, mentee_goal_achievement NUMERIC, impact_score NUMERIC, updated_at TIMESTAMP)	
notifications(id UUID PK, user_id FK, type TEXT, message TEXT, channel ENUM[email,sms,inapp], sent_at TIMESTAMP, read_at TIMESTAMP)	
audit_logs(id UUID PK, actor_id UUID, entity TEXT, entity_id UUID, action TEXT, meta JSONB, at TIMESTAMP)	

Key Notes:

- `matches` table is the relational core.
- All communications and notes are auditable.
- Mentorship cycles are versioned per cohort.
- Cascade deletion disabled (soft delete only).

6) Process Flow

1. **Match:** Profiling data + mentor registry → matching engine → assignments confirmed.
2. **Initialize:** Director defines cycle & milestones → invites mentors and mentees.
3. **Schedule:** Student requests → mentor confirms → session added to calendar.
4. **Conduct:** Session completed → mentor records notes → student provides feedback.
5. **Track:** Goals updated → progress reflected in dashboards.
6. **Review:** Director monitors metrics, adjusts assignments if needed.
7. **Close:** End of cycle → generate reports → archive and reset for next cycle.

Exceptions:

- Mentor unavailable → auto-reassign or suspend match.
- Feedback missing → reminder triggers.
- Goal conflict → director override.

7) Integrations & APIs

Internal REST (prefix `/api/v1/mentorship`)

- `POST /matches/auto` (Director): run matching engine.
- `POST /cycles` (Director): create mentorship cycle.
- `POST /sessions` (Student): request session.
- `PUT /sessions/{id}/confirm` (Mentor).
- `POST /sessions/{id}/notes` (Mentor).
- `POST /sessions/{id}/feedback` (Mentor/Mentee).
- `POST /goals` (Student).
- `PUT /goals/{id}` (Student/Mentor).
- `GET /matches/{id}/summary` (Mentor/Director).
- `GET /mentor-performance` (Director/Admin).
- `POST /cycle/{id}/close` (Director).

External Integrations

- Google Calendar / Microsoft Outlook API for scheduling.
- Twilio / SendGrid for notifications.
- Optional Zoom/Meet link creation via OAuth.

Example Request:

```
POST /sessions
{
  "match_id": "UUID",
  "proposed_date": "2025-11-20T16:00:00Z",
  "duration_min": 60,
  "location": "virtual",
```

```
  "link": "https://meet.google.com/example"
}
```

8) UI/UX Requirements

Student View

- Dashboard: mentor name, next session, goals progress, feedback form.
- Goal Tracker: editable cards per milestone.
- Feedback modal after each session.

Mentor View

- Session queue with statuses; quick confirm/reschedule.
- Mentee profiles (summary + portfolio link).
- Notes editor (Markdown) with rubric checklist.
- Goal dashboard for each mentee.

Director/Admin View

- Overview of active mentorships, cycles, and completion rates.
- Match quality scores & heatmaps.
- Performance dashboards and PDF report generation.

UI/UX Standards

- Real-time sync on schedule updates.
- Accessibility (keyboard navigation, WCAG 2.1 AA).
- Light/Dark mode ready.

9) Notifications & Automation

Event	Audience	Channel	Trigger
Match confirmed	Mentor & Mentee	Email/In-app	Auto-match approval
Session upcoming	Both	SMS/Email	24h before
Feedback overdue	Both	In-app	48h after session
Milestone achieved	Mentor & Director	In-app	Real-time
Cycle closure	All participants	Email	On finalization
Monthly summary	Director	Email (PDF)	1st of month

10) Analytics & Reporting

KPIs

- Total sessions completed per mentor.
- Session attendance rate (%).
- Average mentor rating.
- Average mentee engagement (goals achieved %).
- Cycle completion rate.
- Top 10 mentors by impact score.
- Correlation: profiling readiness → mentorship improvement delta.

Dashboards

- Mentorship Overview (Cohort view).
- Mentor Impact Dashboard.
- Mentee Growth Dashboard (trend from profiling → current readiness).

Data Refresh: Hourly.
Retention: 24 months.

11) Security & Compliance

- RBAC enforcement (only assigned pairs can access private notes).
- TLS 1.3, JWT + refresh token rotation.
- Consent check before mentor sees profiling data.
- Audit trail for every read/write on mentorship data.
- DPA/GDPR compliance (consent, deletion, minimization).
- Sanitized communication logs.
- Encrypted data at rest (AES-256) for notes and feedback.

12) Technical Dependencies

Component	Technology
Front-End	React + Zustand/Redux + React Query
Back-End	Node.js (NestJS) or Python (FastAPI)
Database	PostgreSQL 15
Messaging/Notifications	Twilio, SendGrid, Firebase Cloud Messaging
Calendar	Google Calendar, Microsoft Graph
Storage	AWS S3 for notes attachments
Queue	Redis + BullMQ / Celery
Analytics	Power BI / Metabase integration
CI/CD	GitHub Actions + Docker

Component	Technology
Logging	ELK or OpenSearch Stack

13) Error Handling & Exceptions

Case	Handling
Session conflict	Return 409 + suggest alternative slots
Mentor unreachable	Notify Director; auto-suspend match after 7 days
Feedback timeout	Reminder ×2, escalation to Director
Calendar API failure	Retry (3x) then fallback to email
Unauthorized access	403 + audit entry
Cycle closure before feedback complete	Block closure + list pending items

14) Future Enhancements

1. **AI-Powered Mentor Matching:** Deep profile embedding model using psychometric + KSA + interest vectors.
2. **Voice-to-text Notes:** Automatic transcription for recorded sessions (via Whisper API).
3. **Mentorship Insights:** Predictive success score and recommended mentor interventions.
4. **Integration with Employer Access:** Tag mentee progress readiness for hiring.
5. **Sentiment Analysis:** NLP on notes and feedback for relationship health insights.

Learning & Progress Integration Module (Technical Specification)

1) Module Overview

Central data ingestion and normalization layer that pulls **course/enrollment/progress/badge** signals from external learning systems (e.g., Coursera, TryHackMe, Chris Sanders/Applied Network Defense, Hack The Box, Google Cloud Skills, custom LMS) and reconciles them into a unified **Learning Record Store (LRS)**. It exposes progress dashboards, feeds recommendations, portfolio evidence, cohort analytics, gamification points, and entitlements checks.

Value: one canonical truth for learning activity across providers; low-friction sync; actionable views for students, mentors, and directors.

2) Functional Objectives

1. Connect external providers via OAuth/API keys; schedule secure syncs (pull + webhook).
2. Normalize provider data into a **common schema** (courses, modules, attempts, completions, badges).
3. Map provider artifacts to **OCH missions, competencies, roles**.
4. Provide **student/cohort progress dashboards** and **API endpoints** for other modules.
5. Support manual uploads (CSV/JSON) for providers without APIs.
6. Emit **events** for achievement milestones (to Gamification, Portfolio, Mentorship).
7. Enforce privacy, rate-limits, retries, and audit.

3) User Roles & Permissions

Role	Create	View	Edit	Approve	Delete
Student	Link providers, manual proof upload	Own progress	Disconnect provider	—	Remove own manual uploads
Mentor	Notes on mentee progress	Assigned mentees	—	—	—
Program Director	Provider mappings, mission links	Cohort dashboards	Mapping rules	Approve bulk imports	Revoke bad imports
Admin	Provider adapters, secrets, schedules	All	All	Enable/disable providers	Hard delete (policy)

4) Core Functional Requirements

4.1 Provider Connections & Auth

- **Description:** Securely connect student/provider accounts.
- **User Story:** As a Student, I connect my TryHackMe and Coursera accounts so my progress syncs automatically.
- **Acceptance Criteria:**
  - OAuth 2.0 where supported; API keys/username tokens for legacy; per-provider scopes documented.
  - Token vault (rotation, encryption, revocation).
  - Connection status page (healthy, needs-reauth, revoked).
- **Dependencies:** Secrets vault, auth service.
- **I/O:** `provider_connection_id`, status.

4.2 Sync Orchestration (Pull & Webhooks)

- **Description:** Scheduled pulls + webhook handlers for near-real-time updates.
- **User Story:** As Admin, I set hourly sync for THM and daily for Coursera; webhooks update badges instantly.
- **Acceptance Criteria:**
  - Cron per provider; backoff on rate-limit; idempotent upserts.
  - Webhook signature validation; replay protection.
  - Per-user and bulk sync paths; progress windows (since last cursor).
- **Dependencies:** Job queue, scheduler, webhook endpoints.
- **I/O:** sync_job_id, cursors, metrics.

4.3 Normalization Pipeline

- **Description:** Map heterogeneous provider objects to common schema.
- **User Story:** As Director, I see unified progress regardless of provider naming.
- **Acceptance Criteria:**
  - Canonical objects: **course**, **module**, **attempt**, **completion**, **badge**, **score**, **hours**, **artifact**.
  - Provider adapters fill canonical fields, keep raw JSON for traceability.
  - Deterministic dedupe (provider_user_id + object_id).
- **Dependencies:** Data model, adapter registry.
- **I/O:** normalized rows + raw archive.

4.4 Mission & Competency Mapping

- **Description:** Align provider courses/modules to OCH missions/competencies.
- **User Story:** As Director, I map “THM: SOC Level 1 Path” to Mission NET-01 and competencies X/Y.
- **Acceptance Criteria:**

    - Many-to-many mappings; rule engine supports string/regex/ID lists.
    - Weighting matrix (module weight → competency coverage %).
    - Versioned mapping rules; audit changes.
- **Dependencies:** Missions registry, taxonomy.
- **I/O:** mapping_rules, coverage_aggregates.

4.5 Progress Computation & Reconciliation

- **Description:** Compute **completion %**, **XP**, **scores**, **time-on-task**, and reconcile conflicts.
- **User Story:** As Student, my dashboard shows accurate % and last activity date.
- **Acceptance Criteria:**
    - Progress formula per provider (configurable); normalize to 0–100.
    - Conflict resolution precedence: latest timestamp > highest score (provider-specific overrides).
    - Time-on-task capped per session (anti-inflation).
- **Dependencies:** Rules config.
- **I/O:** progress_snapshots.

4.6 Manual Uploads (No-API Providers)

- **Description:** CSV/JSON template for staff to ingest class outcomes.
- **User Story:** As Director, I import bootcamp results for a weekend lab.
- **Acceptance Criteria:**
    - Template validator; preview diffs; dry-run checks.
    - Owner assignment; provenance tag = `manual`.
- **Dependencies:** File parser, storage.
- **I/O:** import_batch_id, results.

4.7 Events & Downstream Hooks

- **Description:** Emit events for milestones.
- **User Story:** When a student completes a mission-mapped course, award points and suggest portfolio item.
- **Acceptance Criteria:**
    - Events: `learning.progress_updated`, `learning.course_completed`, `learning.badge_earned`, `learning.hours_threshold_reached`.
    - Payload includes user_id, provider, canonical object, mappings, deltas.
- **Dependencies:** Event bus, Gamification/Portfolio listeners.

4.8 Dashboards & APIs

- **Description:** UIs and endpoints for per-student and cohort progress.
- **User Story:** As Mentor, I view my mentee's weekly progress delta and stuck modules.
- **Acceptance Criteria:**

    - Student dashboard: progress over time, last activity, hours, completed items, badges.
    - Cohort view: completion distribution, active users, provider mix, mission coverage.
    - Filters: date range, provider, mission, competency, track.
- **Dependencies:** BI embed or native charts.
- **I/O:** REST/Graph APIs for widgets.

4.9 Data Quality & Health

- **Description:** Monitor adapter health, missing fields, stale tokens.
- **Acceptance Criteria:**
    - Adapter scorecard: success rate, latency, error types.
    - Alerts on token expiry, provider outage, schema drift.

4.10 Privacy, Consent & Opt-Out

- **Description:** Respect student consent per provider.
- **Acceptance Criteria:**
    - Consent record per connection; granular (course titles vs. only aggregates).
    - Opt-out erases provider tokens and de-links objects (retain aggregates if policy allows).

5) Data Model & Key Entities (PostgreSQL)

- **provider_accounts**(id UUID PK, user_id UUID FK, provider ENUM[coursera,thm,and,htb,gcss,custom,...], provider_user_id TEXT, status ENUM[connected,reauth,revoked], scopes TEXT[], connected_at, updated_at, token_ref TEXT)
- **provider_syncs**(id UUID PK, provider ENUM, started_at, finished_at, status ENUM[ok,partial,failed], cursor TEXT, stats JSONB, error TEXT)
- **courses**(id UUID PK, provider ENUM, provider_course_id TEXT, title TEXT, url TEXT, metadata JSONB, raw_ref UUID FK raw_store, UNIQUE(provider, provider_course_id))
- **modules**(id UUID PK, course_id UUID FK, provider_module_id TEXT, title TEXT, order_idx INT, metadata JSONB)
- **enrollments**(id UUID PK, user_id UUID, course_id UUID, status ENUM[enrolled,completed,withdrawn], started_at, completed_at, last_activity_at, provenance ENUM[api,manual], UNIQUE(user_id, course_id))
- **attempts**(id UUID PK, enrollment_id UUID, module_id UUID, attempt_no INT, score NUMERIC, passed BOOL, duration_sec INT, started_at, ended_at, raw_ref UUID)
- **badges**(id UUID PK, user_id UUID, provider ENUM, provider_badge_id TEXT, name TEXT, issued_at DATE, url TEXT, metadata JSONB)
- **progress_snapshots**(id UUID PK, user_id UUID, course_id UUID, completion_pct NUMERIC, hours NUMERIC, last_activity_at, provider ENUM, computed_at)

- **mission_mappings**(id UUID PK, target_type ENUM[course,module,badge], target_id UUID, mission_id UUID, rules JSONB, weight NUMERIC, version INT)
- **competency_coverage**(id UUID PK, user_id UUID, competency_id UUID, coverage NUMERIC, last_updated)
- **raw_store**(id UUID PK, provider ENUM, object_type TEXT, object_id TEXT, json JSONB, checksum TEXT, fetched_at TIMESTAMP)
- **consents**(id UUID PK, user_id UUID, provider ENUM, scope TEXT, granted_at, revoked_at)
- **audit_logs**(... as standard ...)

Notes:

- PII minimization in course titles (if consent = aggregates only).
- Foreign keys to missions/competencies registries.

6) Process Flow

Happy Path (OAuth provider e.g., TryHackMe):

1. Student **connects** provider → OAuth success → `provider_accounts.connected`.
2. Scheduler enqueues **sync** → fetch enrollments, modules, badges since `cursor`.
3. Adapter **normalizes** and upserts **courses/modules/enrollments/attempts/badges**; persists raw JSON.
4. **Compute progress** & **coverage** → update snapshots; emit `learning.progress_updated`.
5. Mapping engine updates **mission/competency aggregates** → emit events for completions/badges.
6. Dashboards refresh; Portfolio receives "suggest item" signal; Gamification awards points.

Exceptions: rate-limit → backoff; invalid token → mark `reauth`; schema drift → quarantine raw & alert.

7) Integrations & APIs

Internal REST (prefix `/api/v1/learning`)

- `POST /providers/{provider}/connect` (Student): start OAuth/keys.
- `GET /me/progress` (Student): unified progress view.
- `GET /cohorts/{id}/progress` (Director): cohort aggregates.
- `POST /imports/manual` (Director/Admin): CSV/JSON upload.
- `GET /signals/coverage?user_id=...` (Mentorship/Gamification): competency coverage.

Webhooks (ingest)

- `/webhooks/learning/{provider}`: verify signature; process `badge_earned`, `course_completed`, etc.

Events (emit)

- `learning.progress_updated`
- `learning.course_completed`
- `learning.badge_earned`
- `learning.mapping_updated`

Example: GET /me/progress (response)


```
{
  "summary": {"completion_pct": 62, "hours": 48.5, "last_activity_at": "2025-
11-07T20:10:00Z"},
  "providers": [

{"name":"tryhackme","completion_pct":78,"hours":22.5,"badges":[{"name":"Blue
Team 1","issued_at":"2025-10-02"}]},
    {"name":"coursera","completion_pct":41,"hours":26.0}
  ],
  "missions": [{"id":"UUID-NET-01","coverage":0.7},{"id":"UUID-DFIR-
02","coverage":0.35}]
}
```


8) UI/UX Requirements

Student

- "Linked Accounts" page (status, last sync, reauth action).
- Progress dashboard: course list, completion %, hours, last activity; provider filter; mission coverage heatmap; suggested next steps.
- Manual proof upload (when allowed) with validation.

Mentor

- Mentee progress card (last 7 days delta, stalled modules).
- "Nudge" action to send encouragement (in-app message).

Director/Admin

- Provider health board; cohort coverage; mapping editor with test preview; import console.

Accessibility & Perf

- Lazy-load long lists; skeleton loaders; WCAG 2.1 AA.

9) Notifications & Automation

Trigger	Audience	Channel	Rule
Token expiring/invalid	Student	In-app + email	Immediate
No activity 7 days	Student	In-app	Weekly
Course completed	Student, Mentor	In-app	Real-time
Badge earned	Student (copy Mentor)	In-app	Real-time
Mapping changed (affects coverage)	Director	In-app	On change

10) Analytics & Reporting

KPIs

- Active learners/week; average completion %; time-on-task; streaks.
- Provider success rates; adapter error rates; sync latency.
- Mission/competency coverage progress by cohort.
- Impact: correlation of learning progress → portfolio approvals → role readiness delta.

Data Refresh: Near real-time (webhooks), hourly (scheduled pulls).
Exports: `/reports/learning/*` CSV/Parquet.

11) Security & Compliance

- OAuth tokens encrypted (KMS) with rotation; never store provider passwords.
- Provider webhooks HMAC-verified; nonce replay protection.
- Consent scoping; data minimization for titles/content if "aggregates-only".
- RBAC & ABAC on records (owner, cohort director, assigned mentor).
- Full audit on imports/exports; retention rules per provider.

12) Technical Dependencies

Component	Tech
Backend	FastAPI/NestJS, adapter plugin system
Queue/Schedule	Redis + BullMQ/Celery; cron
DB	PostgreSQL 15; partitioned tables for snapshots
Storage	S3 for raw_store archives
Observability	OpenTelemetry, Sentry, Prometheus/Grafana
Secrets	KMS/HashiCorp Vault
BI	Metabase/Power BI embeds

13) Error Handling & Exceptions

Case	Handling
Provider 429 (rate-limit)	Exponential backoff with jitter; persist cursor; partial sync OK
Token revoked	Mark `reauth`; notify student; pause sync jobs
Schema drift	Quarantine raw; raise adapter alert; fallback to last good schema
Duplicate records	Idempotent upsert by (provider,object_id); log de-dupe
Large manual import failure	Dry-run validation; error file with line-level causes
Webhook signature invalid	401 + drop; alert admin

14) Future Enhancements

- **xAPI/Caliper** support for richer event semantics.
- **Adaptive learning nudges** (recommend the “next best action” using gaps).
- **LLM summarization** of a learner’s weekly progress.
- **On-device progress capture** for offline workshops with QR check-ins.

Program & Cohort Management Module

Technical Specification (Developer-Ready)

1) Module Overview

The **Program & Cohort Management Module (PCM)** serves as the **core orchestration layer** for OCH SMP.
It defines and manages all academic and training structures — **programs, tracks, specializations, cohorts, intakes, sessions, and calendars** — while ensuring clear relationships between students, mentors, directors, and sponsors.

It also integrates with **Billing, Scheduling, Learning (LRS), Mentorship, and Analytics**, forming the foundation for data consistency across the platform.

Value: Single source of truth for what is running, who is enrolled, when, and under what rules (pricing, schedule, structure).

2) Functional Objectives

1. Define **programs**, **tracks**, and **specializations** with metadata, learning paths, and target roles.
2. Create and manage **cohorts/intakes**, including enrollment caps, schedules, and assigned directors/mentors.
3. Manage **enrollment and seat allocation**, including scholarship/sponsor seats.
4. Provide **calendar and milestone tracking** (orientation, mentorship, submissions, reviews, closure).
5. Expose structured APIs for **Learning, Mentorship, Billing, Gamification**, and **Analytics** modules.
6. Serve as the **policy engine** for program eligibility, duration, and completion rules.
7. Maintain full **audit logs**, **consent hooks**, and **notification triggers**.

3) Core Entities & Hierarchy

```
Program
 ├── Tracks (Builder, Leader, Entrepreneur, Researcher, Educator)
 │    ├── Specializations (optional)
 │    │     ├── Missions (mapped from Registry)
 │    │
 │    └── Cohorts (Jan 2026, May 2026, Sep 2026)
 │          ├── Calendar (start/end, milestones)
 │          ├── Enrollment (students, mentors, sponsors)
 │          └── Metrics (progress, readiness, completion)
 └── Associated Assets (pricing, documents, resources)
```

4) User Roles & Permissions

Role	Create	View	Edit	Approve	Archive/Delete
Admin	All programs, cohorts, seats	All	All	All	All
Program Director	Programs, cohorts (owned tracks)	All within program	Seats, schedules, mentors	Approve enrollments	Archive cohort (after closure)
Mentor	None	Assigned cohorts & mentees	Attendance, milestones	Session reports	—
Student	Self-enroll (if open)	Own program/cohort	Update profile data	—	—
Sponsor/Org Admin	Enroll sponsored seats	Sponsored cohorts	View progress	—	—
Finance	Products, prices, seat caps	Financial dashboards	—	Approve refunds (via Billing)	—

5) Core Functional Requirements

5.1 Program Management

- **Description:** Define programs, durations, tracks, learning goals, pricing models.
- **User Story:** As Admin, I can define "Cybersecurity Leadership Program" with five tracks.
- **Acceptance Criteria:**
  - Metadata: name, description, duration (months), category (mentorship, executive, technical), default price, currency, outcomes.
  - Link to Missions/Competency registry.
  - Define default structure (modules, milestones).
- **I/O:** `program_id`, JSON schema for structure.

5.2 Track & Specialization Definition

- **Description:** Sub-structure defining focused learning paths or professional roles.
- **User Story:** As Director, I define "Cyber Leaders Track" with 3 milestones and 6 modules.
- **Acceptance Criteria:**
  - Define tracks within a program, optional specialization paths.
  - Attach competencies and missions (via registry).
  - Assign directors/mentors per track.

- **Dependencies:** Missions Registry, User & Identity.
- **I/O:** `track_id`, `specialization_id`.

5.3 Cohort Creation & Management

- **Description:** Operational layer — actual running group of learners.
- **User Story:** As Director, I create "Jan 2026 Cohort" for Cyber Builders.
- **Acceptance Criteria:**
  - Attributes: `program_id`, `track_id`, name, start_date, end_date, mode (onsite, hybrid, virtual), seat_cap, mentor_ratio.
  - Status lifecycle: `draft → active → running → closing → closed`.
  - Assign mentors, coordinators, sponsors.
  - Seat pool management (paid, scholarship, sponsored).
  - Calendar template (orientation, sessions, portfolio submission, closure).
  - Integrated with Billing to verify entitlements.
- **I/O:** `cohort_id`, `calendar_id`, `seat_pool`.

5.4 Enrollment & Seat Allocation

- **Description:** Register users into cohorts, control seat utilization.
- **User Story:** As Student, I enroll in a cohort; as Sponsor, I allocate a paid seat.
- **Acceptance Criteria:**
  - Enrollment methods: self-enroll (open), invite, sponsor assign, director assign.
  - Validation: check entitlements (from Billing) or seat availability.
  - Waitlist management (FIFO, auto-promotion).
  - Status: `pending_payment`, `active`, `suspended`, `withdrawn`, `completed`.
  - Multi-cohort prevention (same program).
  - Auto-trigger notifications on enrollment and start.
- **Dependencies:** Billing, Notifications, Consent.
- **I/O:** `enrollment_id`, `seat_status`.

5.5 Calendar & Milestones

- **Description:** Cohort schedule of deliverables and key events.
- **User Story:** As Director, I set calendar milestones (orientation, assessments, reviews).
- **Acceptance Criteria:**
  - Create calendar from template or scratch.
  - Event types: `orientation`, `mentorship`, `project_review`, `holiday`, `submission`, `closure`.
  - Time-zone aware; syncs with Scheduling & Notifications.
  - Completion tracking for each milestone.
- **Dependencies:** Scheduling Hub, Notifications.
- **I/O:** `calendar_id`, `milestone_id`.

5.6 Mentor Assignment

- **Description:** Assign mentors to cohorts or tracks.
- **User Story:** As Director, I assign 3 mentors to Cyber Builders (Jan 2026).
- **Acceptance Criteria:**
  - Auto-match from Mentorship registry by availability and skillset.
  - Assign primary/backup mentors.
  - Role in cohort dashboard: `mentor`, `support_mentor`, `guest`.
- **Dependencies:** Mentorship Management, Profiling.
- **I/O:** `mentor_assignment_id`.

5.7 Program Rules & Completion

- **Description:** Define program success metrics and auto-graduation logic.
- **User Story:** As Director, I define 80% completion and portfolio approval as graduation criteria.
- **Acceptance Criteria:**
  - Rules configurable per program (attendance %, portfolio approval, feedback score, payment complete).
  - Auto-update student status to `completed` or `incomplete`.
  - Notify director, trigger certificate generation.
- **Dependencies:** Portfolio, Learning, Billing, Gamification.
- **I/O:** `completion_rule_id`, `certificate_trigger`.

5.8 Reports & Dashboards

- **Description:** Real-time visibility into cohort status, seats, completion rates.
- **User Story:** As Director, I view seat utilization and progress summaries.
- **Acceptance Criteria:**
  - Cohort dashboard: enrollments, seat utilization, mentor assignments, readiness delta, completion %, payments.
  - Export CSV/JSON to Analytics layer.
- **Dependencies:** Analytics Layer, Gamification.
- **I/O:** `cohort_report_uri`.

6) Data Model (PostgreSQL)

Table	Description
`programs`	id UUID PK, name TEXT, category ENUM[technical, leadership, mentorship], description TEXT, duration_months INT, default_price NUMERIC, currency TEXT, status ENUM[active,inactive], created_at, updated_at
`tracks`	id UUID PK, program_id FK, name TEXT, key TEXT, description TEXT, competencies JSONB, director_id FK, created_at, updated_at

Table	Description
specializations	id UUID PK, track_id FK, name TEXT, description TEXT, missions JSONB, duration_weeks INT
cohorts	id UUID PK, track_id FK, name TEXT, start_date DATE, end_date DATE, mode ENUM[onsite,virtual,hybrid], seat_cap INT, mentor_ratio FLOAT, calendar_id FK, status ENUM[draft,active,running,closing,closed], created_at, updated_at
enrollments	id UUID PK, cohort_id FK, user_id FK, org_id FK nullable, enrollment_type ENUM[self,sponsor,invite], seat_type ENUM[paid,scholarship,sponsored], payment_status ENUM[pending,paid,waived], status ENUM[active,withdrawn,completed], joined_at, completed_at
calendar_events	id UUID PK, cohort_id FK, type ENUM[orientation,session,submission,holiday,closure], title TEXT, description TEXT, start_ts TIMESTAMP, end_ts TIMESTAMP, location TEXT, link TEXT, status ENUM[scheduled,done,cancelled]
mentor_assignments	id UUID PK, cohort_id FK, mentor_id FK, role ENUM[primary,support,guest], assigned_at, active BOOL
program_rules	id UUID PK, program_id FK, rule JSONB (criteria, thresholds, dependencies), version INT, active BOOL
certificates	id UUID PK, enrollment_id FK, file_uri TEXT, issued_at TIMESTAMP
audit_logs	id UUID PK, actor_id UUID, entity TEXT, entity_id UUID, action TEXT, meta JSONB, at TIMESTAMP

7) Process Flow (Happy Path)

Program Setup → Cohort Launch → Enrollment → Execution → Closure

1. **Create Program** → Define tracks/specializations → Assign directors/mentors.
2. **Create Cohort** → Set calendar & seat cap → Open enrollment.
3. **Enrollment Phase**
    - Student/Sponsor enroll → Billing validates payment/entitlement.
    - On success → Enrollment activated → Notifications triggered.
4. **Execution Phase**
    - Cohort milestones tracked via calendar → sessions & projects linked to milestones.
    - Mentorship, Learning, Portfolio updates feed cohort dashboards.
5. **Closure Phase**
    - Evaluate rules → auto-compute completions.
    - Issue certificates → Archive cohort → Trigger Analytics pipeline.

8) Integrations & APIs

Internal REST (prefix `/api/v1/programs`)

Endpoint	Method	Description
`/programs`	POST	Create program
`/programs`	GET	List programs
`/programs/{id}`	GET/PUT	View or update program
`/tracks`	POST/GET	Create/list tracks
`/cohorts`	POST/GET	Create/list cohorts
`/cohorts/{id}/calendar`	POST/GET	Manage cohort calendar
`/cohorts/{id}/enrollments`	POST/GET	Enroll student or list cohort members
`/cohorts/{id}/mentors`	POST/GET	Assign mentors
`/rules`	POST/GET	Define completion rules
`/certificates/{id}`	GET	Download issued certificate

Events (emit via message bus):

- `program.created`
- `cohort.opened`
- `cohort.closed`
- `enrollment.created`
- `enrollment.completed`
- `certificate.issued`

Webhook consumers:

- **Billing:** listen to `cohort.opened`, `enrollment.created`.
- **Analytics:** subscribe to `cohort.closed`.
- **Notifications:** handle `calendar_event.soon`, `certificate.issued`.

9) UI/UX Requirements

Director View

- Dashboard: Active cohorts, seats left, completion rate.
- Forms: create/edit programs, tracks, cohorts.
- Calendar drag-drop with milestone tags.
- Mentor assignment & tracking view.
- Report exports.

Student View

- My Cohort card: mentor list, key dates, completion %, certificate status.
- Enrollment journey: waitlist → payment → confirmation.
- Calendar integration (Google/Outlook).

Admin View

- Global overview of programs, active cohorts, seat utilization by track, completion % across programs.
- Rules configuration (UI for JSON rules).
- Archival dashboard.

10) Security & Compliance

- **RLS:** Student sees only own enrollment/cohort; mentors only assigned cohorts.
- **Access control:** Directors limited to owned programs/tracks.
- **Data retention:** Cohort data retained 5 years post-closure; personal data per DPA retention.
- **Audit:** All changes logged (`actor_id`, `action`, `before/after`).
- **Encryption:** all PII and certificates in AES-256 at rest.
- **Backups:** daily incremental, weekly full; tested quarterly.

11) Technical Dependencies

Component	Tech Stack
Backend	FastAPI / NestJS with PostgreSQL ORM
DB	PostgreSQL 15
Queue	Redis + BullMQ / Celery
Calendar Integration	Google Calendar / Outlook API
Notifications	Twilio (SMS), SendGrid (Email), Firebase (In-App)
File Storage	AWS S3 (certificates, templates)
Auth	Identity Service (JWT middleware)
Analytics	Event stream → Warehouse (dbt models)
Observability	OpenTelemetry, Prometheus, Sentry

12) Error Handling & Exceptions

Scenario	Handling
Seat cap reached	409 Conflict + “Join waitlist”
Duplicate enrollment	409 Conflict + enrollment_id
Calendar conflict	Suggest alternate time; flag in UI
Mentor double-booked	Reject; notify Director

Scenario	Handling
Payment pending > 7 days	Auto-expire enrollment; notify student
Rule eval failure	Log + retry + notify Director

13) Future Enhancements

1. **Auto-scheduling engine** — generate calendars from mentor availability and time zones.
2. **Smart seat allocation** — dynamic seat reallocation based on payment/withdrawals.
3. **Multi-track linking** — allow cross-track modules for specialization.
4. **Internship/placement integration** — link top graduates to employers (via Employer Portal).
5. **Adaptive milestones** — personalize timelines based on learner pace.

C) Engagement & Value

Gamification & Engagement Module (Technical Specification)

1) Module Overview

System that **motivates, measures, and nudges** learner behavior across OCH by awarding **points, badges, levels, streaks**, and **challenge/quest** completions, with **leaderboards** at student, cohort, track, and team levels. It consumes events from Profiling, Learning Integration, Mentorship, Portfolio, and Billing to compute **engagement signals** and drive **automations** (reminders, rewards, unlocks). Includes **anti-cheat**, **season management**, and optional **rewards redemption**.

Value: consistent incentive layer aligned to OCH outcomes (skills, evidence, consistency, collaboration); improves retention, throughput, and quality.

2) Functional Objectives

1. Maintain a **points economy** with configurable rules and rate limits.
2. Issue **badges** (manual or rule-based), **levels/tiers**, and **streaks** with decay logic.
3. Run **challenges/quests** (solo) and **events/competitions** (team/cohort) with clear scoring.
4. Provide **leaderboards** with privacy controls and anti-gaming safeguards.
5. Trigger **nudges** and **rewards** (e.g., credits, access unlocks) based on achievements.
6. Expose **signals & APIs** for dashboards and downstream modules.
7. Enforce RBAC, audit, consent, and data minimization.

3) User Roles & Permissions

Role	Create	View	Edit	Approve	Delete/Archive
Student	Join challenges, claim rewards (per policy)	Own stats, public boards	Update team opt-in/alias	—	—
Mentor	Propose challenges, give verifications	Mentees' progress	Submit manual verifications	Approve small verifications (policy)	—
Program Director	Challenge templates, seasons, rules	All cohort boards	Adjust scoring rules (versioned)	Approve large verifications & awards	Archive seasons
Admin	System configs, integrations	All	All	Enable reward catalogs	Hard delete (exception)

Role	Create	View	Edit	Approve	Delete/Archive
Finance	Rewards budgets (credits/vouchers)	Reward ledgers	Issue vouchers	Approve > threshold payouts	Revoke vouchers (policy)

4) Core Functional Requirements

4.1 Points Engine & Rule Sets

- **Description:** Deterministic rules converting platform events into points with rate limits, caps, and decay.
- **User Story:** As Director, I define rules: "+50 pts when a mission-mapped course is completed" (daily cap 200).
- **Acceptance Criteria:**
  - Rules modeled as: `trigger → filter → formula → cap → cooldown`.
  - Supported triggers (from event bus):
    - `profiling.profile_completed`, `learning.course_completed`, `learning.badge_earned`, `portfolio.item_approved`, `mentorship.session_completed`, `mentorship.goal_achieved`, `billing.payment_succeeded` (optional), `attendance.check_in` (future).
  - Formulas allow constants and weighted fields, e.g., `points = base + 10*module_weight`.
  - Per-user **cooldown** (e.g., award at most once per day for "daily study").
  - **Caps** (daily/weekly/seasonal) and **decay** (e.g., −5% per inactive week).
- **Dependencies:** Event bus, rate-limit store (Redis).
- **I/O:** `award_id`, `points_awarded`, `reason`.

4.2 Badges & Issuance

- **Description:** Rule-based and manual digital badges with criteria, evidence, and verification links.
- **User Story:** As Director, I configure "Blue Team Apprentice" for completing 3 SOC missions + 2 portfolio approvals.
- **Acceptance Criteria:**
  - Badge metadata: title, tier, criteria, icon, issue authority, version, expiry (optional).
  - Issuance via rule conditions or manual verification flows.
  - Evidence links (portfolio items, learning records).
  - Optional OpenBadges/Verifiable Credential export (future).
- **Dependencies:** Portfolio, Learning LRS.
- **I/O:** `badge_id`, `issued_to`, `evidence[]`.

4.3 Levels / Tiers

- **Description:** Progressive tiers based on cumulative points and milestone badges.
- **User Story:** As Student, I level up from **Explorer** to **Builder** at 1,000 pts + "Portfolio Starter" badge.
- **Acceptance Criteria:**
  - Tier config: thresholds, prerequisites (badges), benefits (visual + entitlements).
  - Auto-promote/demote (demotion optional; default no demote).
  - Emit `gamification.level_changed` events.
- **Dependencies:** Entitlements (optional), Billing (if perks).
- **I/O:** `level_change_event`.

4.4 Streaks & Consistency

- **Description:** Track consecutive active days/weeks with flexible definitions.
- **User Story:** As Student, my 14-day streak persists if I log learning ≥20 minutes/day.
- **Acceptance Criteria:**
  - Streak rules: activity kinds, minimum activity thresholds, grace windows.
  - Streak freeze token (rare: illness/exam) granted by Director.
  - Decay or reset rules configurable.
- **Dependencies:** Learning snapshots, Mentorship sessions.
- **I/O:** `streak_counter`, `streak_events`.

4.5 Challenges, Quests & Events

- **Description:** Time-boxed tasks with objectives and multi-step quests; team events (e.g., Capture-the-Flag).
- **User Story:** As Director, I launch a **21-Day DFIR Quest** with daily tasks; as Student, I claim steps and get verified.
- **Acceptance Criteria:**
  - Challenge types: **solo**, **team**, **cohort**; with start/end, scoring schema, tie-breakers.
  - Steps: checklist/scores; auto-verifiable via LRS or mentor verification.
  - Leaderboards per challenge; fraud flags for anomalous claims.
  - Prize pools linked to Rewards (credits/vouchers/items).
- **Dependencies:** LRS, Mentorship, Rewards module.
- **I/O:** `challenge_id`, `enrollment_id`, `claims[]`.

4.6 Leaderboards

- **Description:** Rankings by points, streaks, badges, or challenge score at different scopes.
- **User Story:** As Student, I see top 10 in my cohort and my percentile.
- **Acceptance Criteria:**
  - Scopes: **cohort**, **track**, **team**, **regional** (if needed).
  - Privacy: opt-out from public boards (still visible to staff).
  - Anti-gaming: exclude flagged awards; smoothing windows; minimum activity rules.

- **Dependencies:** Aggregation jobs.
- **I/O:** paginated leaderboard APIs.

4.7 Rewards & Entitlements (Optional)

- **Description:** Convert achievements into **rewards**: credits, vouchers, swag, access unlocks.
- **User Story:** As Student, I redeem 1,000 pts for a mentorship clinic pass.
- **Acceptance Criteria:**
  - Reward catalog with inventory/budgets; redemption costs; approval thresholds.
  - On redemption, create **entitlement** or **voucher**; adjust points ledger; issue receipt.
  - Finance approvals for high-value redemptions.
- **Dependencies:** Billing/Entitlements, Finance.
- **I/O:** `redemption_id`, `entitlement_id` or `voucher_code`.

4.8 Anti-Cheat & Moderation

- **Description:** Detect and manage abuse: mass submissions, duplicate evidence, anomalous patterns.
- **Acceptance Criteria:**
  - Heuristics: velocity checks, duplicate artifact hashes, out-of-pattern timestamps, IP anomalies.
  - **Flags** create review cases; moderators can claw back points/badges.
  - Appeal workflow with audit trail.
- **Dependencies:** Portfolio hashes, LRS, IP logs.
- **I/O:** `fraud_case_id`, `action_log`.

4.9 Seasons & Theming

- **Description:** Segment progress into **seasons** (e.g., "Jan–Mar 2026") with unique content and resets.
- **Acceptance Criteria:**
  - Season configs: start/end, themed badges, seasonal leaderboards, carry-over rules.
  - Archive season stats; keep lifetime totals separately.
- **I/O:** `season_id`, archives.

4.10 Engagement Nudges & Comms

- **Description:** Behavior-based notifications (e.g., "streak at risk", "close to level-up").
- **Acceptance Criteria:**
  - Trigger templates with thresholds & quiet hours; channels (in-app, email, optional SMS).
  - Personalization via recent activity and gaps (from Profiling/LRS).
  - Opt-out preferences per channel.
- **Dependencies:** Notifications service.

5) Data Model & Key Entities (PostgreSQL)

- **seasons**(id UUID PK, name TEXT, starts_at, ends_at, theme JSONB, active BOOL)
- **points_ledger**(id UUID PK, user_id UUID, season_id UUID, delta INT, reason TEXT, source_event TEXT, event_id UUID, created_at, status ENUM[applied,reversed,held], meta JSONB)
- **balances**(user_id UUID PK, season_id UUID, total_points INT, lifetime_points INT, last_decay_at TIMESTAMP)
- **badges**(id UUID PK, key TEXT unique, title TEXT, tier ENUM[bronze,silver,gold,platinum], icon_uri TEXT, criteria JSONB, version INT, active BOOL)
- **badge_awards**(id UUID PK, badge_id UUID FK, user_id UUID FK, issued_at TIMESTAMP, evidence JSONB, issuer_id UUID, status ENUM[issued,revoked], reason TEXT)
- **levels**(id UUID PK, name TEXT, threshold_points INT, prerequisites JSONB, benefits JSONB, order_idx INT)
- **level_states**(user_id UUID, level_id UUID, attained_at TIMESTAMP, active BOOL, PRIMARY KEY(user_id, level_id))
- **streaks**(id UUID PK, user_id UUID, type ENUM[daily,weekly], counter INT, last_activity DATE, longest INT, frozen_until DATE nullable)
- **challenges**(id UUID PK, name TEXT, type ENUM[solo,team,cohort], season_id UUID, starts_at, ends_at, rules JSONB, scoring JSONB, prizes JSONB, status ENUM[draft,live,closed])
- **challenge_enrollments**(id UUID PK, challenge_id UUID, user_id UUID nullable, team_id UUID nullable, joined_at TIMESTAMP, status ENUM[active,withdrawn,completed])
- **challenge_claims**(id UUID PK, enrollment_id UUID, step_key TEXT, proof JSONB, verified_by UUID nullable, verified_at TIMESTAMP, status ENUM[pending,auto_verified,approved,rejected], score INT)
- **leaderboard_snapshots**(id UUID PK, scope ENUM[cohort,track,team,global], scope_id UUID, metric ENUM[points,streaks,challenge], computed_at TIMESTAMP, rankings JSONB)
- **rewards**(id UUID PK, name TEXT, type ENUM[credit,voucher,swag,access], cost_points INT, inventory INT, policy JSONB, active BOOL)
- **redemptions**(id UUID PK, reward_id UUID, user_id UUID, status ENUM[pending,approved,fulfilled,rejected], requested_at, decided_at, entitlement_id UUID nullable, voucher_code TEXT nullable)
- **fraud_cases**(id UUID PK, user_id UUID, detected_at TIMESTAMP, signals JSONB, status ENUM[open,under_review,closed], actions JSONB)
- **audit_logs**(... standard ...)

Indexes:

- `points_ledger(user_id, created_at)`; `badge_awards(user_id)`; `leaderboard_snapshots(scope, scope_id, computed_at)`.

6) Process Flow (Happy Path)

1. **Event Ingest:** `learning.course_completed` arrives → rule engine evaluates → awards +50 pts → write to `points_ledger`, update `balances`.
2. **Badge Check:** Rules met → issue "Path Finisher" badge → write `badge_awards`.
3. **Level Update:** Balance crosses threshold → set `level_states` → emit `level_changed`.
4. **Streak Update:** Daily activity detected → increment `streaks.counter`.
5. **Challenge Claim:** Student submits quest step → auto-verify with LRS → approve and score.
6. **Leaderboards:** Hourly jobs compute `leaderboard_snapshots`.
7. **Nudges:** Near level-up/streak risk triggers notifications.
8. **Rewards:** Student redeems points → create `redemptions` → grant `entitlement` or `voucher`.

Exceptions: rule cap reached (award held), fraud signals raised (ledger entry set `held`), manual moderation.

7) Integrations & APIs

Internal REST (prefix `/api/v1/gamification`)

- `GET /me/summary` (Student): points, level, streak, badges, active challenges.
- `GET /leaderboards?scope=&id=&metric=&limit=`: ranked list + your_rank.
- `POST /challenges/{id}/enroll` (Student/Team).
- `POST /claims` (Student): submit challenge step proof.
- `POST /redemptions` (Student): redeem reward.
- `POST /rules` (Director/Admin): create/update rule set (versioned).
- `POST /badges` / `POST /badges/{id}/issue` (Director/Admin).
- `POST /moderation/fraud/{user_id}/actions` (Admin): clawback, revoke, warn.

Events (emit)

- `gamification.points_awarded`
- `gamification.badge_issued`
- `gamification.level_changed`
- `gamification.streak_updated`
- `gamification.challenge_updated`
- `gamification.redemption_created|fulfilled`

Example: GET /me/summary (response)

```
{
```

```
  "season": {"id":"UUID-SEASON","name":"Q1 2026"},
  "points": {"season": 1240, "lifetime": 4510},
  "level": {"name":"Builder","next_threshold":1500},
  "streak": {"type":"daily","current":12,"longest":19},
  "badges": [{"key":"soc_path_apprentice","tier":"bronze","issued_at":"2026-
01-14"}],
  "active_challenges": [{"id":"UUID-CH-21DFIR","progress": 0.67}],
  "rewards": {"eligible": [{"reward_id":"UUID-RW-
CLINIC","type":"access","cost_points":1000}]}
}
```

8) UI/UX Requirements

Student

- **My Progress**: points card, level banner, streak widget, recent awards.
- **Challenges**: browse/join; quest detail with steps, proofs, timers.
- **Badges**: grid with criteria; shareable badge view (optional).
- **Leaderboards**: toggle scope; percentile; opt-out control.
- **Rewards**: catalog, redemption flow, history.

Mentor

- **Mentee Gamification Panel**: points/streak trend, badges earned, claims awaiting verification.
- Quick verify/reject UI with evidence preview.

Director/Admin

- **Rule Builder** (visual): triggers → filters → formula → caps → cooldown.
- **Season Manager**; **Fraud Cases** dashboard; **Reward Catalog** manager.
- Accessibility: WCAG 2.1 AA; sensible empty states; skeleton loaders.

9) Notifications & Automation

Trigger	Audience	Channel	Timing
Near level-up (≤100 pts)	Student	In-app	Immediate
Streak at risk (no activity by 19:00 local)	Student	In-app + email	Daily
Challenge step verified/rejected	Student	In-app	Immediate
Fraud case opened/closed	Student + Director	In-app + email	Immediate
Redemption approved/fulfilled	Student	In-app + email	Immediate
Weekly digest	Student, Mentor	Email	Mondays 08:00

Quiet hours and user preferences respected.

10) Analytics & Reporting

KPIs

- Weekly Active Learners (WAL), average points per active user, median streak length.
- Badge issuance rates by cohort; challenge participation & completion.
- Correlations: gamification metrics → portfolio approvals → role readiness delta.
- Redemption liability (points outstanding) and burn rate.

Dashboards

- **Engagement Overview**, **Challenge Analytics**, **Fraud Signals**, **Rewards Finance**.
  Refresh: near real-time for awards; hourly for aggregates.
  Exports: `/reports/gamification/*` CSV/Parquet.

11) Security & Compliance

- RBAC/ABAC on all endpoints; students only see allowed scopes.
- Rate-limit all mutation endpoints; idempotency keys for claims/redemptions.
- Signed URLs for evidence previews; content sanitization.
- Botswana DPA/GDPR alignment; explicit consent for public leaderboards & shareable badges.
- Full audit logs for point changes, badge revocations, and redemptions.
- Anti-cheat signals **do not** expose sensitive profiling data to peers.

12) Technical Dependencies

Component	Tech
Backend	FastAPI/NestJS; rules engine (jsonlogic or custom); BullMQ/Celery for jobs
Cache/Queue	Redis for rate limits, cooldowns, leaderboard pipelines
DB	PostgreSQL 15 (partition points_ledger if high volume)
Storage	S3 for badge icons, evidence attachments
Observability	OpenTelemetry, Prometheus/Grafana, Sentry
BI	Metabase/Power BI embeds
Feature flags	LaunchDarkly/Open-source equivalent (optional)

13) Error Handling & Exceptions

Case	Handling
Duplicate event re-processed	Idempotent awards via `(event_id, user_id)` unique key
Rule cap exceeded	Return `held` status; surface message to UI

Case	Handling
Fraud suspicion	Mark ledger entry `held`; open `fraud_case`; no leaderboard update
Evidence URL invalid	422 with reason; allow resubmit
Redemption inventory 0	409; suggest alternatives
Leaderboard compute fail	Retry with backoff; show last snapshot

14) Future Enhancements

- **Coaching Bots**: micro-nudges with next best action from gaps.
- **Team "Houses"** with cross-track collaboration bonuses.
- **Season Pass** content and premium reward tiers (ties to Billing).
- **On-chain badge proofs** (hash only) for tamper-evidence.
- **Real-time sockets** for live events/CTFs.

Payment & Subscription Module (Technical Specification)

1) Module Overview

Central billing layer for OCH SMP that sells **program seats, cohorts, add-ons** (e.g., exam fees), and **subscriptions** (monthly mentorship plans, alumni access), supporting:

- **Cards + Mobile Money + Bank Transfer** via regional gateways (e.g., Paystack, Flutterwave) and direct invoice flows.
- **One-time** and **recurring** charges, **installments**, **scholarship/sponsorship codes**, **corporate invoicing**, **refunds**, **credits**, **dunning**, and **reconciliation**.
- Gates platform access (profiling, sessions, content) based on **entitlements** derived from payment state.

Value: single source of truth for financial state and learner entitlements, minimal PCI scope (tokenization), automated revenue ops, and clear reporting.

2) Functional Objectives

1. Define **products/plans** and prices per currency with tax rules and cohort capacity ties.
2. Accept payments (card, mobile money) and support **offline** bank transfer & **corporate PO/invoice**.
3. Manage **subscriptions**, **installments**, **proration**, **pauses**, **up/downgrades**.
4. Issue **invoices/receipts** (PDF), **credit notes**, and perform **refunds/voids**.
5. Implement **dunning** (retries & reminders) and **fraud/risk checks** (gateway + internal).
6. Generate **entitlements** to unlock features/cohorts on success; revoke on failure/cancel.
7. Provide **ledger-grade** reconciliation and export for finance.
8. Enforce RBAC, audit, consent, and data-minimization (PCI SAQ-A posture).

3) User Roles & Permissions

Role	Create	View	Edit	Approve	Delete/Archive
Student	Checkout sessions, billing profile	Own invoices, subs, credits	Update billing info (tokenized), cancel/pause sub (policy)	—	—
Finance	Products, price books, tax rules, invoices, refunds	All financial data	Amend invoices (where allowed), post credits	Approve refunds, write-offs	Archive price versions

Role	Create	View	Edit	Approve	Delete/Archive
Program Director	Cohort price overrides, seat caps	Sales dashboards, seat fill	Issue scholarship/sponsorship codes	Approve manual seat assignment	—
Admin	System config, gateways, webhooks, roles	All	All	Enable gateways	Hard delete (policy exceptions only)

4) Core Functional Requirements

4.1 Product Catalog & Price Books

- **Description:** Define Products (e.g., "OCH Mentorship – Jan 2026 Cohort"), Plans (billing cadence), Add-ons.
- **User Story:** As Finance, I create a cohort product with BWP/USD prices and early-bird discount.
- **Acceptance Criteria:**
  - Product types: `one_time`, `subscription`, `installment_bundle`, `addon`.
  - Price book by currency (BWP, USD, ZAR, KES, configurable), tax rules, validity (start/end), channel (public/private/corporate).
  - Versioned prices; no edits after sales—new version required.
  - Seat cap per cohort product; reserve on checkout start (expiring hold).
- **I/O:** product_id, price_id, seat_hold_id.

4.2 Checkout & Payments

- **Description:** Hosted checkout (gateway) and embedded form; tokenized—no PAN stored.
- **User Story:** As a Student, I pay via mobile money or card and receive instant confirmation.
- **Acceptance Criteria:**
  - Gateways: Paystack/Flutterwave adapters; pluggable interface.
  - Methods: cards, mobile money (as supported per country), bank transfer reference.
  - Coupons/vouchers/scholarship codes; sponsorship (third-party pays for student).
  - Taxes/fees itemized; currency and FX display where relevant.
  - Strong customer auth as required by issuer/gateway.
- **Dependencies:** Gateway SDKs, price book, tax engine, seat holds.
- **I/O:** payment_intent_id, charge_id, status.

4.3 Subscriptions & Installments

- **Description:** Create/manage recurring plans and installment schedules.

- **User Story:** As a Student, I select a 3-installment plan; system auto-charges monthly.
- **Acceptance Criteria:**
  - Frequencies: monthly/quarterly/annual; installments (n of m).
  - Proration on mid-cycle changes; pause/resume; grace periods.
  - Auto-retry rules (e.g., day 1/3/7) before marking as failed.
  - Payment calendar and upcoming charges visible to student.
- **Dependencies:** Scheduler/queue, webhook handlers.
- **I/O:** subscription_id, schedule, next_charge_at.

4.4 Invoicing, Receipts, Credit Notes

- **Description:** Issue tax-compliant invoice PDFs and receipts; credit notes for refunds/adjustments.
- **User Story:** As Finance, I send an invoice to a corporate sponsor and track payment.
- **Acceptance Criteria:**
  - Invoice states: `draft → issued → paid / void / written_off`.
  - PDF render (logo, address, tax reg, line items, tax breakdown).
  - Email delivery with secure link; view history.
  - Credit notes linked to original invoice lines.
- **Dependencies:** PDF renderer, mailer.
- **I/O:** invoice_id, pdf_uri, credit_note_id.

4.5 Dunning & Collections

- **Description:** Automated retries and reminders for failed payments.
- **User Story:** As Finance, I configure 3 retries and escalate after 7 days.
- **Acceptance Criteria:**
  - Retry schedule configurable; reason codes logged.
  - Notifications: T+0 failure, T+1 retry, T+3 final notice.
  - Optional entitlement downgrade on failure (after grace).
- **Dependencies:** Notifications, scheduler.
- **I/O:** dunning_case_id, events[].

4.6 Refunds, Voids, Chargebacks

- **Description:** Process full/partial refunds; react to chargebacks.
- **User Story:** As Finance, I refund 50% due to withdrawal policy.
- **Acceptance Criteria:**
  - Policy checks; approvals for amounts > threshold.
  - Gateway API calls; update ledger; regenerate docs.
  - Chargeback webhooks create cases, freeze entitlements pending outcome.
- **I/O:** refund_id, chargeback_case_id.

4.7 Sponsorships, Scholarships, Corporate Invoicing

- **Description:** Third-party payer flows and codes covering full/partial amounts.

- **User Story:** As Director, I issue 10 sponsor codes for a foundation.
- **Acceptance Criteria:**
  - Code types: % discount, fixed amount, seat grant, named sponsor.
  - Budget tracking per sponsor; utilization reports.
  - Corporate: PO number, invoice to company, assign seats to named students later (seat wallet).
- **I/O:** sponsor_code_id, sponsor_wallet_id.

4.8 Entitlements & Access Control

- **Description:** Convert successful financial events into **entitlements** (e.g., "Jan-2026 Cohort Seat", "Mentorship Access (3 months)").
- **User Story:** As SMP, on paid event I unlock Profiling & Mentorship features.
- **Acceptance Criteria:**
  - Entitlement types: `cohort_seat`, `module_access`, `time_bounded_access`, `addon`.
  - Grant on `payment_succeeded`; revoke/downgrade on `payment_failed_final` or refund.
  - Read API for other modules to check access gates.
- **I/O:** entitlement_id, entitlement_status.

4.9 Reconciliation & Finance Reporting

- **Description:** Daily reconciliation of gateway payouts vs. internal ledger.
- **User Story:** As Finance, I export settlements by currency and method.
- **Acceptance Criteria:**
  - Import gateway payout files/webhooks; match charges/fees.
  - Reports: MRR, ARR (subs), cohort revenue, refunds, taxes, receivables.
  - Export CSV/Parquet to BI.
- **I/O:** payout_batch_id, recon_report_uri.

4.10 Compliance, Audit & Risk

- **Description:** Minimal PCI scope; audit all financial actions; configurable regional tax.
- **Acceptance Criteria:**
  - PCI SAQ-A: hosted fields/redirects; store only tokens & last4/brand.
  - Tax engine with per-jurisdiction rates (configurable); invoice compliance fields.
  - Fraud signals (AVS/CVV result from gateway), velocity checks, blocklists/allowlists.
  - Full audit trail for price changes, refunds, voids, write-offs.

5) Data Model & Key Entities (PostgreSQL)

- **products**(id UUID PK, type ENUM[one_time,subscription,installment_bundle,addon], name TEXT, description TEXT, cohort_id UUID nullable, seat_cap INT, active BOOL, created_at)
- **prices**(id UUID PK, product_id FK, currency TEXT, amount_cents INT, tax_rule_id UUID, channel ENUM[public,private,corporate], valid_from TIMESTAMP, valid_to TIMESTAMP, version INT, active BOOL)
- **tax_rules**(id UUID PK, name TEXT, country TEXT, region TEXT, rate_bp INT, inclusive BOOL, metadata JSONB)
- **coupons**(id UUID PK, code TEXT unique, type ENUM[pct,amount,seat,scholarship], value INT, currency TEXT nullable, max_redemptions INT, expires_at TIMESTAMP, sponsor_id UUID nullable, metadata JSONB)
- **seat_holds**(id UUID PK, product_id FK, user_id FK, status ENUM[held,expired,converted], expires_at TIMESTAMP)
- **subscriptions**(id UUID PK, user_id FK, product_id FK, status ENUM[active,past_due,paused,canceled,completed], start_at TIMESTAMP, current_period_end TIMESTAMP, schedule JSONB, gateway_sub_id TEXT)
- **invoices**(id UUID PK, user_id FK, account_id UUID nullable, status ENUM[draft,issued,paid,void,written_off], currency TEXT, totals JSONB, pdf_uri TEXT, issued_at TIMESTAMP, due_at TIMESTAMP, po_number TEXT nullable)
- **invoice_lines**(id UUID PK, invoice_id FK, product_id FK, description TEXT, qty INT, unit_amount_cents INT, tax_cents INT, total_cents INT, metadata JSONB)
- **payments**(id UUID PK, invoice_id FK, gateway TEXT, method ENUM[card,mobile_money,bank_transfer], currency TEXT, amount_cents INT, status ENUM[pending,succeeded,failed,refunded,voided], gateway_ref TEXT, received_at TIMESTAMP, failure_reason TEXT)
- **refunds**(id UUID PK, payment_id FK, amount_cents INT, reason TEXT, status ENUM[pending,processed,failed], processed_at TIMESTAMP)
- **credits**(id UUID PK, user_id FK, currency TEXT, balance_cents INT, source TEXT, metadata JSONB)
- **entitlements**(id UUID PK, user_id FK, type ENUM[cohort_seat,module_access,time_bounded_access,addon], ref_id UUID nullable, status ENUM[active,inactive,revoked,expired], starts_at TIMESTAMP, ends_at TIMESTAMP)
- **sponsor_wallets**(id UUID PK, sponsor_id UUID, currency TEXT, budget_cents INT, consumed_cents INT, metadata JSONB)
- **payouts**(id UUID PK, gateway TEXT, currency TEXT, amount_cents INT, fees_cents INT, paid_at TIMESTAMP, batch_ref TEXT)
- **recon_logs**(id UUID PK, payout_id FK, payment_id FK, status ENUM[matched,unmatched,disputed], note TEXT)
- **audit_logs**(id UUID PK, actor_id UUID, entity TEXT, entity_id UUID, action TEXT, meta JSONB, at TIMESTAMP)

Security: store only gateway tokens & last4/brand/expiry; **never** PAN/CVV.

6) Process Flow

Happy path (one-time cohort seat):

1. Student selects **Cohort Product** → system creates **seat_hold** (15 min).
2. Checkout initializes **payment_intent** via gateway; apply coupon/sponsor if any.
3. Gateway confirms **succeeded** → create **invoice (paid)** + **payment**; convert seat_hold to seat.
4. Issue **entitlement: cohort_seat**; notify Student + Director; release seat count.
5. Email **receipt (PDF)**; update analytics.

Subscriptions/Installments:

- Create **subscription** with schedule → charge now/prorate → entitlement **module_access** with rolling validity → renew via scheduler; on failure run **dunning** and potentially **downgrade**.

Refund:

- Finance initiates → gateway processes → create **credit note/refund** → revoke or adjust entitlements.

Corporate invoice:

- Finance issues **invoice** to sponsor → on **paid** (bank transfer/gateway) → generate **sponsor wallet** or assign seats to named students → entitlements granted on assignment.

7) Integrations & APIs

Internal REST (prefix `/api/v1/billing`)

- `GET /catalog` (public): products & active prices (by currency).
- `POST /checkout/sessions` (student): start checkout, returns gateway session/url.
- `POST /coupons/apply` (student): validate/apply code.
- `GET /invoices/my` (student): list with PDFs.
- `GET /entitlements/my` (student): current access rights.
- `POST /subscriptions` (student): create.
- `POST /subscriptions/{id}/pause|resume|cancel` (student/finance per policy).
- `POST /invoices` (finance): create corporate invoice.
- `POST /refunds` (finance): initiate refund.
- `GET /reports/revenue|cohorts|subs` (finance/admin).

Webhook Endpoints (per gateway)

- `/webhooks/payments` events: `payment_succeeded`, `payment_failed`, `chargeback_opened`, `chargeback_closed`, `payout_settled`.
- On `payment_succeeded` → finalize invoice → grant entitlements → notify.
- On `payment_failed` → create dunning case → notify.

Example: Create Checkout Session (request)

```
{
  "product_id": "UUID-PROD-COHORT-JAN26",
  "price_id": "UUID-PRICE-BWP",
  "currency": "BWP",
  "quantity": 1,
  "coupon_code": "EARLYBIRD14",
  "sponsor_code": null,
  "metadata": {"cohort_id":"UUID-COHORT-JAN26"}
}
```

Response

```
{
  "seat_hold_id":"UUID-HOLD",
  "gateway_session_url":"https://gateway.example/...",
  "expires_at":"2025-11-08T12:10:00Z"
}
```

8) UI/UX Requirements

Student

- Pricing page (currency switch), plan selector, coupon entry.
- Clear tax breakdown, total, refund policy link.
- Wallet/credits display, upcoming invoices & renewals.
- Invoices/receipts list with PDF download.

Finance/Admin

- Catalog manager (products/prices), tax rules, cohorts seat caps.
- Real-time payments monitor (success/fail), dunning queue.
- Reconciliation dashboard (payouts vs internal).
- Sponsors & wallets, code issuance, utilization view.

Director

- Seat utilization (per cohort), revenue summary, sponsorship usage.

Accessibility

- WCAG 2.1 AA; strong error messages; mobile-first checkout.

9) Notifications & Automation

Trigger	Audience	Channel	Timing
Payment succeeded	Student, Director	Email + In-app	Immediate
Upcoming renewal	Student	Email + In-app	7 days & 24h
Payment failed	Student	Email + In-app	Immediate
Dunning final notice	Student	Email + SMS (optional)	T+3 days
Invoice issued (corporate)	Sponsor Contact	Email	Immediate
Refund processed	Student/Sponsor	Email	Immediate
Seat cap 90% reached	Director/Finance	In-app	Real-time

10) Analytics & Reporting

KPIs

- Gross/Net revenue by product/currency; MRR/ARR; Cohort fill %.
- Payment method mix; authorization vs. capture success; refund & chargeback rates.
- DSO (days sales outstanding) for invoices; dunning recovery %.
- Sponsorship utilization; average discount; CAC proxy (if tracked).

BI

- Hourly ingestion to warehouse; prebuilt Metabase/Power BI dashboards.
- Export endpoints (`/reports/*`) for CSV/Parquet.

11) Security & Compliance

- **PCI SAQ-A:** Hosted fields/redirect; store only tokens, last4, brand, exp; **never** PAN/CVV.
- TLS 1.3, HSTS; JWT with short TTL; webhook signatures verified (HMAC).
- PII minimization in invoices; encryption at rest (AES-256/KMS).
- RBAC/ABAC for finance data; row-level access by account.
- Audit every price change, refund, and entitlement grant/revoke.
- Regional privacy compliance (Botswana DPA, GDPR alignment); data retention policies.

12) Technical Dependencies

Component	Tech
Frontend	React, React Hook Form, React Query
Backend	FastAPI or NestJS; Pydantic/TypeORM; Celery/BullMQ for schedules/dunning
DB/Cache	PostgreSQL 15; Redis (jobs, seat holds)

Component	Tech
Payments	Paystack/Flutterwave SDKs; pluggable gateway interface
Storage	S3 for PDFs; wkhtmltopdf/WeasyPrint for rendering
Observability	OpenTelemetry, Prometheus/Grafana, Sentry
CI/CD	GitHub Actions, Docker, Terraform (IaC)

13) Error Handling & Exceptions

Case	Handling
Seat hold expired	410; prompt to retry; release seat
Gateway timeout	Retry with idempotency key; show pending state
Webhook signature invalid	401; drop event; alert admin
Currency mismatch	422; enforce price currency or reselect
Partial gateway success, invoice fail	Compensating txn: roll back entitlement, mark payment for manual review
Refund greater than captured	400; block and inform finance
Chargeback	Freeze entitlements; open case; notify finance/director

14) Future Enhancements

- **Payment Orchestration** (multi-provider failover/routing for better auth rates).
- **Buy-Now-Pay-Later** partners (where compliant).
- **Usage-based** add-ons (e.g., extra mentorship hours).
- **Self-serve quotes** and **CPQ** for corporate deals.
- **Tax service integration** (e.g., Avalara-like) once cross-border volumes grow.

D) Data & Insights

Analytics & Reporting Layer (Technical Specification)

1) Module Overview

Cross-module analytics fabric that ingests operational data from **Profiling, Portfolio, Mentorship, Learning/LRS, Billing, Gamification** into a governed warehouse and serves **dashboards, self-serve reports, and data APIs**. It provides cohort and program KPIs, drill-downs, scheduled reports, and export feeds for partners.

Value: one trusted source for performance, outcomes, and revenue; reduces ad-hoc data work; powers decision assurance.

2) Functional Objectives

1. Provide a **semantic model** of OCH entities with conformed dimensions (student, cohort, track, mission, competency, role).
2. Implement reliable **ETL/ELT pipelines** with late-arriving data handling and SCDs.
3. Deliver **prebuilt dashboards** (student, mentor, cohort, finance, exec) and **scheduled reports** (PDF/CSV).
4. Expose **metrics APIs** and governed **ad-hoc SQL** for analysts.
5. Enforce **row/column-level security**, PII minimization, and auditability.
6. Support **SLAs** (data freshness, uptime) and **quality checks** (schema, nulls, referential).

3) User Roles & Permissions

Role	View	Build	Export	Admin
Student	Own progress snapshot	—	Export own data	—
Mentor	Mentee/cohort assigned	—	CSV for mentee lists	—
Program Director	Program/cycle/cohort	Create saved looks	Schedule reports	Approve shared datasets
Finance	Revenue, payouts, AR/AP	Build finance looks	Export finance packs	Reconcile locks
Admin	All	Publish models	Approve external feeds	RLS/CLS policies
Data Analyst	All non-PII by policy	Build models/ETL	Dataset exports	DQ rules, lineage

4) Core Functional Requirements

4.1 Data Ingestion & Staging

- **Description:** Pull from operational DBs and event streams into staging with CDC (change data capture).
- **Acceptance Criteria:**
  - Connectors for Postgres (logical decoding) and event bus (Kafka/Redis streams).
  - Partitioned staging tables with load metadata (batch_id, source_ts).
  - Schema evolution handling (nullable add, versioning).
- **I/O:** `_stg_*` tables; load logs.

4.2 Transformation & Modeling (ELT)

- **Description:** Build **core warehouse** with dimensional models (Kimball-style) and **SCD Type 2** for slowly changing dims.
- **Acceptance Criteria:**
  - Dimensions: `dim_student`, `dim_cohort`, `dim_track`, `dim_role`, `dim_competency`, `dim_mission`, `dim_time`, `dim_mentor`, `dim_product`.
  - Facts: `fact_profiling`, `fact_learning_progress`, `fact_portfolio`, `fact_mentorship`, `fact_gamification`, `fact_revenue`, `fact_entitlements`.
  - SCD2 on student and mentor attributes (e.g., track, cohort).
  - Surrogate keys; conformed dimension usage across facts.
- **Dependencies:** Orchestrator, SQL engine (warehouse).

4.3 Metric Layer / Semantic Model

- **Description:** Central catalog of **certified metrics** with definitions and filters.
- **Acceptance Criteria:**
  - KPIs with SQL semantics + description + owner:
    - **Readiness Δ** (profiling pre/post), **Portfolio Health**, **Mentorship Engagement**, **Learning Completion %**, **MRR/ARR**, **Seat Fill %**, **Challenge Participation**, **Dunning Recovery %**.
  - Versioned; breaking changes require deprecation period.
  - Exposed to BI (Metabase/Power BI) and Metrics API.

4.4 Dashboards & Reports

- **Description:** Prebuilt dashboards by persona with drill-throughs and PDF/CSV schedule.
- **Acceptance Criteria:**
  - **Executive:** North-star KPIs, cohort comparisons, revenue.
  - **Program Director:** cohort funnel (onboard→profile→learn→portfolio→ready), competency heatmap, mentor performance.
  - **Mentor:** mentee progress, risk flags, suggested nudges.
  - **Finance:** revenue, refunds, payouts, receivables, sponsorship utilization.

    - Scheduling: cron, timezone aware, recipients lists, secure links.
- **Dependencies:** BI server, renderer.

4.5 Data Quality (DQ) & Lineage

- **Description:** Automated checks and lineage graphs.
- **Acceptance Criteria:**
    - DQ tests: row counts, null ratios, key uniqueness, referential integrity, KPI reconciliation (e.g., revenue).
    - On failure: alert, quarantine downstream loads, show banner on dashboards.
    - Lineage: source→model→dashboard map with last refresh timestamps.

4.6 Row/Column Security & Privacy

- **Description:** RLS by role and cohort; CLS for PII masking.
- **Acceptance Criteria:**
    - Students see only self; mentors see assigned mentees; directors see owned cohorts; finance sees finance only.
    - Column masking for PII (email, phone) outside permitted roles.
    - Aggregation thresholds to prevent small-n deanonymization.

4.7 Exports & Data Shares

- **Description:** Controlled dataset exports and partner feeds.
- **Acceptance Criteria:**
    - Exports: CSV/Parquet with watermarks and access logs.
    - External shares via signed URLs or secure SFTP; schema contracts (versioned).
    - Rate limits and expiry.

4.8 Metrics API

- **Description:** Read-only REST for certified metrics and timeseries.
- **Acceptance Criteria:**
    - `GET /metrics/{key}` with `filters` (cohort_id, date_range, track).
    - Consistent responses with metadata (definition, last_refreshed).
    - Caching; ETags.

4.9 Alerting & Anomaly Detection (Optional)

- **Description:** KPI threshold and anomaly alerts.
- **Acceptance Criteria:**
    - Rules: static threshold and STL/Prophet-based seasonality.
    - Channels: email, Slack/MS Teams, in-app.
    - Alert suppression windows; ownership.

5) Data Model & Warehouse Schema (Example)

Dimensions (SCD2 where noted)

- **dim_student**(sk INT PK, user_id UUID, name, email_masked, country, timezone, track_sk, cohort_sk, status, valid_from, valid_to, is_current BOOL) — *SCD2*
- **dim_cohort**(sk INT PK, cohort_id UUID, name, track_sk, start_date, end_date)
- **dim_track**(sk INT PK, track_key TEXT, name TEXT)
- **dim_role**(sk INT PK, role_key TEXT, name TEXT)
- **dim_competency**(sk INT PK, key TEXT, category TEXT, name TEXT)
- **dim_mission**(sk INT PK, mission_id UUID, name TEXT, quadrant TEXT)
- **dim_time**(date_key INT PK, date, week, month, quarter, year)
- **dim_mentor**(sk INT PK, mentor_id UUID, name_masked, expertise, status, valid_from, valid_to, is_current) — *SCD2*
- **dim_product**(sk INT PK, product_id UUID, name, type, currency)

Facts (grain & sample fields)

- **fact_profiling**(student_sk, time_key, readiness_score, trait_vector_json, attempt_id UUID)
- **fact_learning_progress**(student_sk, time_key, provider, completion_pct, hours, last_activity_at)
- **fact_portfolio**(student_sk, time_key, items_submitted, items_approved, health_score, competency_coverage_json)
- **fact_mentorship**(student_sk, mentor_sk, time_key, sessions_completed, feedback_avg, goals_achieved)
- **fact_gamification**(student_sk, time_key, points, badges_earned, streak_days)
- **fact_revenue**(student_sk nullable, time_key, currency, amount_cents, refunds_cents, net_cents, product_sk, payment_method, status)
- **fact_entitlements**(student_sk, time_key, entitlement_type, active_flag, starts_at, ends_at)

Indexes: date partitions on fact tables; composite keys `(student_sk, time_key)`.

6) Process Flow

1. **Capture:** CDC + events stream to `_stg_*`.
2. **Validate:** DQ checks; quarantine on failure.
3. **Transform:** ELT builds `dim_*` then `fact_*` (incremental by `time_key`).
4. **Publish:** Refresh semantic model; invalidate BI caches.
5. **Deliver:** Dashboards update; schedules dispatch PDFs/CSVs.
6. **Monitor:** SLAs (e.g., hourly refresh 06:00–22:00 Gaborone time); alerts on breach.

Exceptions: source schema drift → halt downstream models, alert with lineage; late data → backfill windows.

7) Integrations & APIs

Metrics API (prefix `/api/v1/analytics`)

- `GET /metrics/{key}?cohort_id=&track=&from=&to=`
- `GET /dashboards/{id}/pdf?params=...` (secure).
- `GET /exports/{dataset}` → signed link or stream.

Events (emit)

- `analytics.refresh_completed` (includes datasets updated, duration, status).

Example Response:

```
{
  "metric": "cohort_readiness_delta",
  "definition": "Average (post - pre) profiling readiness for cohort",
  "filters": {"cohort_id":"UUID-COHORT-JAN26"},
  "value": 12.4,
  "unit": "points",
  "last_refreshed": "2025-11-08T08:05:00+02:00"
}
```

8) UI/UX Requirements

BI Embeds (Director/Finance/Exec)

- Tiled KPIs, time-series, cohort comparison tables, competency heatmaps.
- Drill-through to student (authorized roles) and mentor views.
- Export buttons (PDF/CSV) with watermarking.
- Data health banner (green/amber/red) with last refresh timestamp.

Student/Mentor Mini-Dashboards (App UI)

- Lightweight widgets showing trends and goals, not raw data tables.

Accessibility

- High-contrast themes; keyboard navigation; downloadable CSVs with headers.

9) Notifications & Automation

Event	Audience	Channel	Rule
SLA Missed (refresh)	Data Analyst/Admin	Email/In-app	Immediate

Event	Audience	Channel	Rule
KPI Breach (e.g., completion < 40%)	Director	In-app	After 2 consecutive days
Finance Daily Pack	Finance	Email (PDF + CSV)	07:30 daily
Cohort Weekly Report	Director	Email (PDF)	Mondays 08:00

10) Security & Compliance

- **RLS/CLS** enforced at warehouse and BI layers; student PII masked unless privileged role.
- **PII minimization** in exports; watermarked downloads; access logs per export.
- **Consent lineage**: reports involving profiling traits require consent scope check.
- **Audit** all dataset creations, metric changes, and schedule runs.
- **Retention**: facts retained indefinitely; raw staging purged after 90 days (configurable).
- **Compliance**: Botswana DPA/GDPR alignment; subject access requests supported via controlled exports.

11) Technical Dependencies

Component	Tech
Warehouse	PostgreSQL (OLAP extensions) or Snowflake/BigQuery (if scale); dbt for ELT
Orchestration	Airflow/Prefect; CI via GitHub Actions
BI	Metabase/Power BI embedded
Event Bus	Kafka or Redis Streams (from app)
Storage	S3 for export artifacts & PDF renders
Observability	OpenTelemetry; Prometheus/Grafana; Datafold/Great Expectations for DQ
Security	Vault/KMS for creds; SSO/OIDC for BI

12) Error Handling & Exceptions

Case	Handling
DQ failure	Quarantine dataset, mark last good snapshot as current; alert
Late-arriving facts	Backfill last N partitions; re-compute dependents
Schema drift	Auto-create nullable columns (non-breaking); open ticket for model update
Export timeout	Stream in chunks; retry once; notify requester with link
Dashboard render fail	Fallback to last cached PDF; flag report

13) Future Enhancements

- **Feature store** for ML (mentor matching, success prediction).
- **What-if simulators** for cohort capacity/pricing (connect Billing + Outcomes).
- **Natural-language queries** (semantic search over metrics).
- **Data contracts** with upstream services (schema + SLAs enforced in CI).

Registries & Shared Services

Missions / Competency / Role Registry (MCRR)

Technical Specification (Developer-Ready)

1) Module Overview

The **MCRR** is the canonical catalog for **roles → competencies → missions → artifacts** used across OCH.
It standardizes how programs map learning activities, assessments, portfolio evidence, and outcomes to industry roles (SOC Analyst, Cloud Sec Eng, DFIR, etc.). All other modules (Profiling, LRS, Portfolio, Mentorship, Gamification, Analytics) reference these keys.

Value: one source of truth for definitions, rubrics, and mappings—enabling consistent progress tracking, readiness scoring, and employer search.

2) Objectives

1. Model **roles** (target jobs/profiles), **competencies** (skills/knowledge/abilities), and **missions** (learning/assessment units).
2. Provide **rubrics** (criteria/levels/weights) and **coverage maps** (mission→competency).
3. Version, approve, and retire items without breaking dependent data.
4. Expose APIs/SDKs for lookup, validation, and analytics joins.
5. Support localization, tags, frameworks (MITRE/NIST/NICE/ISO), and evidence types.

3) Domain Model & Definitions

- **Role**: target outcome (e.g., "SOC Analyst L1").
- **Competency**: measurable ability within a domain (e.g., "Log Analysis – Windows").
  - Attributes: type (knowledge/skill/ability/attitude), level scale (e.g., 0–4), references (NICE/NIST/ATT&CK).
- **Mission**: structured learning/assessment unit aligned to real incidents or workflows (e.g., "Detect Suspicious RDP").
  - Contains steps, inputs, tools, expected outputs, verification mode, and evidence artifacts.
- **Rubric**: scoring criteria with descriptors and weights for portfolio/assessments.
- **Coverage**: mapping `mission → competencies (weight%)`, `role → competencies (target level)`.

4) Roles & Permissions

Role	Capabilities
Registry Admin	Approve/publish/retire items; manage versions; import/export
Curriculum Architect	Propose/edit missions, competencies, rubrics; create drafts
Program Director	Bind missions to tracks/cohorts; define program coverage targets
Mentor/Reviewer	View rubrics; suggest edits; annotate gaps
Student	Read-only public descriptions (when exposed); see mapping on their items
Employer (Portal)	Read selected role/competency definitions (public catalog)

---

5) Core Functional Requirements

5.1 Registry CRUD & Versioning

- Draft → review → approved → published → retired.
- **Immutable published versions**; changes create new version with `supersedes`.
- Backwards-compatibility window: old keys remain resolvable for N months.

5.2 Roles Catalog

- Fields: `name`, `level (L1/L2/L3)`, `description`, `industry_refs`, `expected_outcomes`, `entry_requirements`, `tools_stack`, `tags`.
- **Role→Competency targets**: `{competency_id, target_level, weight%}`.

5.3 Competency Library

- Fields: `key`, `name`, `type`, `domain`, `subdomain`, `scale (0..4 or 0..5)`, `descriptors per level`, `framework_refs (NICE, ATT&CK, ISO)`; `evidence_types`.
- **Hierarchy**: domain → competency group → competency.
- **Obsolescence**: mark deprecated with replacement link.

5.4 Missions Registry

- Fields: `key`, `title`, `summary`, `context/threat`, `tools`, `prereqs`, `steps[]`, `expected_outputs`, `assessment_mode (auto/manual/hybrid)`, `time_estimate`, `difficulty`, `evidence_schema (files/links/logs)`, `safety_notes`.
- **Coverage**: `{competency_id, weight%}` (sum 100).
- **Verification**: auto signals (LRS events), manual rubric, or blended.
- **Assets**: templates (pcap, log sets), instructions, solution outlines (mentor-only).

5.5 Rubrics & Criteria

- **Rubric**: `{criterion_key, description, levels[{score, descriptor}], weight%}`.
- Supports **min-bar** (must-pass) criteria; **auto-check** hooks (regex/JSON schema for evidence).
- Bind rubrics to **mission** or **role capstones**.

5.6 Program/Track Bindings

- Allow directors to compose **track blueprints** by selecting missions and setting min thresholds per competency.
- Export **coverage matrix**: cohort target vs available missions (gap analysis).

5.7 Tags, Frameworks & References

- Attach multi-tags (e.g., `cloud`, `dfir`, `windows`, `aws`, `identity`, `zero_trust`).
- Link to **framework references** (NICE tasks codes, ATT&CK techniques, NIST controls).

5.8 Localization & Accessibility

- Localized titles/descriptions/rubrics; left-to-right/right-to-left readiness.
- Markdown support with safe rendering.

5.9 Integrity & Impact Analysis

- Show **blast radius** on change: which programs/cohorts/missions/competencies are impacted.
- Required approval if change affects published cohorts.

6) Data Model (PostgreSQL)

```
roles(
  id UUID PK, version INT, key TEXT unique, name TEXT, level TEXT,
  description TEXT, industry_refs JSONB, expected_outcomes TEXT,
  tools_stack JSONB, tags TEXT[], status
ENUM[draft,approved,published,retired],
  supersedes UUID null, created_by UUID, created_at, updated_at
)

competencies(
  id UUID PK, version INT, key TEXT unique, name TEXT, type
ENUM[knowledge,skill,ability,attitude],
  domain TEXT, subdomain TEXT, scale_max INT, descriptors JSONB,
  framework_refs JSONB, evidence_types TEXT[], tags TEXT[],
  status ENUM[draft,approved,published,retired], supersedes UUID null,
  created_by UUID, created_at, updated_at
```

```
)

roles_competencies_targets(
  role_id UUID FK, competency_id UUID FK, target_level INT, weight INT,
  PRIMARY KEY (role_id, competency_id)
)

missions(
  id UUID PK, version INT, key TEXT unique, title TEXT, summary TEXT,
  context TEXT, tools JSONB, prereqs JSONB, steps JSONB,
  expected_outputs JSONB, assessment_mode ENUM[auto,manual,hybrid],
  time_estimate_mins INT, difficulty ENUM[beginner,intermediate,advanced],
  evidence_schema JSONB, safety_notes TEXT, tags TEXT[],
  status ENUM[draft,approved,published,retired], supersedes UUID null,
  created_by UUID, created_at, updated_at
)

missions_competencies(
  mission_id UUID FK, competency_id UUID FK, weight INT,
  PRIMARY KEY (mission_id, competency_id)
)

rubrics(
  id UUID PK, key TEXT unique, title TEXT, version INT, criteria JSONB,
  min_bar JSONB, status ENUM[draft,approved,published,retired], supersedes
UUID null,
  created_by UUID, created_at, updated_at
)

mission_rubrics(
  mission_id UUID FK, rubric_id UUID FK, PRIMARY KEY (mission_id, rubric_id)
)

bindings_tracks_missions(   -- track blueprint composition
  id UUID PK, track_id UUID, mission_id UUID, mandatory BOOL, order_idx INT
)

bindings_tracks_targets(    -- track-level competency targets
  id UUID PK, track_id UUID, competency_id UUID, target_level INT
)
```

Indexes:

- `roles(key, status)`, `competencies(key, status)`, `missions(key, status)`.
- GIN on `tags`, `framework_refs`, `descriptors`.

7) API Surface (prefix `/api/v1/registry`)

Public/Read

- `GET /roles?status=published&tag=`
- `GET /roles/{key}`
- `GET /competencies?domain=&tag=`

- `GET /competencies/{key}`
- `GET /missions?tag=&difficulty=&status=`
- `GET /missions/{key}`
- `GET /rubrics/{key}`
- `GET /coverage/mission/{key}` → competencies + weights
- `GET /coverage/role/{key}` → competencies + targets

Admin/Write

- `POST /roles` / `PUT /roles/{key}` / `POST /roles/{key}/publish` / `POST /roles/{key}/retire`
- Same for `competencies`, `missions`, `rubrics`.
- `POST /tracks/{id}/blueprint` (bind missions + targets).
- `POST /lint` — validate a bundle (see below).
- `POST /import` / `GET /export` (JSON bundles, versioned).

Validation (Lint) Response Example

```
{
  "valid": false,
  "errors": [
    {"path":"missions[ctf_windows_rdp].coverage","message":"weights must sum
to 100"},
    {"path":"roles[soc_l1].targets","message":"unknown competency
'cloud_iam_basics'"}
  ],
  "warnings": [{"path":"competencies[win_logs]","message":"no framework
refs"}]
}
```

8) Integrations

Module	How it uses the Registry
Profiling	Emits readiness per `competency_key`; aligns assessments to competency targets by role/track
LRS (Learning)	Maps course completions to `mission_key` → updates competency coverage
Portfolio	Each artifact binds to `mission_key` + `competency_keys`; reviewers use `rubric_key`
Mentorship	Goals/outcomes reference `competency_keys`; session templates pull mission steps
Gamification	Awards points/badges for mission completion or competency thresholds
Program/Cohort Mgmt	Track blueprints and gap analyses rely on mission/competency coverage
Analytics	Conformed dims: `dim_role`, `dim_competency`, `dim_mission`; facts join via keys

Module	How it uses the Registry
Employer Portal	Search candidates by role/competency; show evidence mapped to missions

9) Workflows

9.1 Authoring a New Mission

1. Architect drafts mission (steps, evidence schema, coverage, rubric).
2. Lint check (weights=100, valid competency refs, rubric consistency).
3. Peer review → changes requested → approve.
4. Publish v1; notify Program Directors; log in Audit.

9.2 Updating a Competency

1. Create **v2** with refined descriptors or scale.
2. System computes **impact set** (roles, missions, tracks affected).
3. Directors choose **adopt now** (new cohorts) or **grandfather** (current cohorts stay on v1).
4. Publish; deprecated version stays resolvable for N months.

9.3 Building a Track Blueprint

1. Director selects missions; sets **mandatory** flags and order.
2. Define target levels for key competencies.
3. Export blueprint to Program/Cohort Mgmt; schedule calendars accordingly.

10) UI/UX Requirements

- **Catalog explorer** with filters: domain, framework, difficulty, tags.
- **Diff viewer** for versions (what changed).
- **Coverage matrix**: mission × competencies heatmap; role × competencies target table.
- **Impact analyzer** sidebar during edits.
- **Rubric editor** with live preview; criteria weights must total 100.
- **Localization panel** for translations.
- **Safe previews** of mentor-only content (solution notes gated).

Accessibility: WCAG 2.1 AA; keyboard-first editing.

11) Governance & Quality Controls

- Mandatory **framework mapping** for security domains (e.g., ATT&CK/NICE) when applicable.
- **Peer review quorum** (≥2 approvers) before publish.
- Automated **broken link** and **asset checksum** checks.

- **Content SLAs**: review cycle every 12 months; expiry warnings.
- **Ownership**: each item has an **owner** and **backup owner**.

12) Security & Compliance

- Read is public/internal per item flag; mentor-only content behind RBAC.
- All writes audited (who/when/what changed).
- Evidence schemas avoid PII; any sample data sanitized.
- Exports/imports signed and verified; optional allowlist for import provenance.

13) Tech & Ops

Component	Choice
Backend	FastAPI/NestJS
DB	PostgreSQL 15 (JSONB for descriptors/rubrics/steps)
Search	Meilisearch/OpenSearch on titles/tags/descriptions
Storage	S3 for mission assets (pcap/logs), with signed URLs
Caching	Redis for hot lookups (keys → payload vX)
Observability	OTel, Prometheus/Grafana, Sentry
CI	Schema lint, contract tests, content tests (weights sum, refs exist)

14) Error Handling

Case	Handling
Coverage weights ≠ 100	422 validation error with suggested fix
Unknown competency/role key	404; lint lists closest matches
Publishing with unresolved impacts	Block publish; show impacted list
Asset checksum mismatch	409; require re-upload
Attempt to edit published version	409; instruct to create new version

15) Future Enhancements

- **Auto-mapping assistant** (suggest competencies from mission text using NLP).
- **Cross-provider alignment** (map Coursera/THM modules → missions at scale).
- **Employer skill ontology sync** (import external taxonomies).
- **Adaptive role profiles** (regional variants: Botswana/Namibia/Zimbabwe).
- **Badge schemas** auto-generated from rubrics (OpenBadges/VC).

Scheduling & Calendar Hub (SCH)

Technical Specification (Developer-Ready)

1) Module Overview

The **Scheduling & Calendar Hub (SCH)** is the unified time-management backbone for the entire OCH Student Management Platform.
It coordinates **mentorship sessions**, **program calendars**, **assessments**, **cohort events**, **payment renewals**, and **notifications** across different time zones.

It provides:

- **Central calendar service** with time-zone awareness, recurring rules (RFC 5545/ICAL-compatible), and conflict resolution.
- **Availability management** for mentors, students, and programs.
- **Booking engine** for 1-to-1, 1-to-many, and multi-track events.
- **Integration** with external calendars (Google, Outlook, ICS).
- **APIs** for scheduling, rescheduling, reminders, and attendance tracking.

Value: ensures smooth coordination of all activities across OCH programs while maintaining a unified source of time-bound truth for analytics, notifications, and billing.

2) Objectives

1. Provide a **shared calendar layer** across all SMP modules.
2. Manage **time-zone-aware scheduling** with recurrence, reminders, and cancellations.
3. Enable **availability & slot booking** for mentors, students, and staff.
4. Sync with **external calendars** (Google, Outlook) for bi-directional updates.
5. Trigger **event-based notifications**, reminders, and attendance updates.
6. Feed **time-series analytics** (participation, completion, adherence rates).

3) Scope & Boundaries

In-scope:

- Event creation, update, cancellation.
- Availability windows, slot generation, conflict checks.
- Resource locking (rooms, mentors, students).
- Integrations with Notifications (NAS) and Mentorship modules.
- Time zone normalization (IANA TZ).

Out-of-scope:

- Video-conferencing (will integrate via providers like Zoom, Teams, Meet).
- HR/Payroll attendance management.

4) Roles & Permissions

Role	Capabilities
Scheduler Admin	Create/manage program calendars, override conflicts, manage time zones
Mentor	Manage availability, approve/reschedule bookings, sync external calendar
Student	Book/reschedule allowed sessions, view program calendar, sync reminders
Program Director	Define academic calendars, blackout dates, bulk bookings
System Service	Sync and trigger reminders, push attendance
External (Employer)	View or book demo sessions (if permitted)

5) Core Functional Requirements

5.1 Calendar & Event Management

- Create, update, cancel, and duplicate events.
- Event metadata:
- `id, title, description, start_time, end_time, time_zone, recurrence_rule (RFC5545),`
- `organizer_id, participant_ids[], location/meeting_link, visibility (public/private),`
- `category (mentorship, assessment, class, review, payment, general), color_tag`
- Support **recurring events** with `RRULE`, `RDATE`, `EXDATE`.
- Auto-handle **DST changes**.
- Sync real-time to external calendars via webhooks.

5.2 Availability & Slot Booking

- Mentor/Trainer defines **availability blocks** (e.g., Mon-Fri 14:00–18:00 CAT).
- System generates **bookable slots** with customizable duration (e.g., 45 min).
- Slot states: `available`, `held`, `booked`, `expired`.
- Prevent double booking and time overlaps.
- Allow recurring availability templates (weekly patterns).
- Integrate with Mentorship module for mentorship session booking.

5.3 Conflict Detection & Resolution

- Cross-check overlapping events for the same user/resource.

- Warn or auto-resolve using priority logic:
  `Program Events > Mentorship > Assessment > Personal/Other`.
- Optional "soft holds" for tentative events awaiting confirmation.

5.4 External Calendar Integration

- OAuth2-based connection for Google/Outlook.
- Two-way sync (create/update/delete).
- Privacy controls for event visibility (show busy only, or full details).
- Sync interval: every 10 min or via webhook push if supported.

5.5 Reminders & Notifications

- Pre-event reminders (e.g., 24h, 2h, 15min).
- Post-event follow-ups (e.g., feedback forms, attendance check-ins).
- Event-triggered notifications → NAS (email/SMS/in-app).
- Support digest format (daily/weekly summary of upcoming events).

5.6 Attendance & Check-in

- In-event "Join" links (unique, expiring).
- Attendance captured automatically via link click or manual confirmation.
- Attendance statuses: `present`, `late`, `absent`, `excused`.
- Attendance analytics integrated with Reporting layer.

5.7 Time Zones & Localization

- All times stored in UTC; convert for display via user profile time zone.
- Auto-detect from IP or account settings.
- Handle daylight saving changes automatically.
- Locale-based formatting (en-GB, en-US, fr-FR, etc.).

5.8 Program & Cohort Calendars

- Each **program/cohort** has a master calendar combining:
  - Training sessions
  - Mentorship slots
  - Deadlines & events
- Exportable via `.ics` or embedded calendar view.
- Color-coded per category.

5.9 Resource Management

- Register and assign shared resources:
  - Meeting rooms, Zoom links, or virtual resource pools.

- Allocate automatically or manually per event.
- Prevent overuse or duplication.

5.10 Audit & Versioning

- Log all create/update/cancel/reschedule actions with reason codes.
- Keep change history (who/when).
- Capture invite/accept/decline responses (ICS semantics).

6) Data Model (PostgreSQL)

```
calendars(
  id UUID PK, owner_type ENUM[user,program,cohort], owner_id UUID,
  name TEXT, color TEXT, visibility ENUM[private,shared,public],
  created_at, updated_at
)

events(
  id UUID PK, calendar_id UUID FK, title TEXT, description TEXT,
  start_time TIMESTAMPTZ, end_time TIMESTAMPTZ, time_zone TEXT,
  recurrence_rule TEXT, recurrence_exceptions TEXT[],
  organizer_id UUID, location TEXT, meeting_link TEXT,
  category ENUM[mentorship,assessment,class,review,payment,other],
  visibility ENUM[private,shared,public], color_tag TEXT,
  status ENUM[scheduled,cancelled,completed], created_at, updated_at
)

participants(
  id UUID PK, event_id UUID FK, user_id UUID, role
ENUM[organizer,mentor,student,guest],
  response ENUM[pending,accepted,declined,tentative], check_in_at TIMESTAMPTZ
null,
  check_out_at TIMESTAMPTZ null, attendance_status
ENUM[present,late,absent,excused] null
)

availabilities(
  id UUID PK, mentor_id UUID, day_of_week INT, start_time TIME, end_time
TIME,
  time_zone TEXT, recurrence ENUM[weekly,biweekly,custom], valid_from DATE,
valid_to DATE,
  status ENUM[active,inactive], created_at, updated_at
)

slots(
  id UUID PK, availability_id UUID FK, start_time TIMESTAMPTZ, end_time
TIMESTAMPTZ,
  status ENUM[available,held,booked,expired], booked_by UUID null, event_id
UUID null,
  created_at, updated_at
)

resources(
```

```
  id UUID PK, type ENUM[room,zoom_link,virtual_pool], name TEXT, capacity
INT,
  status ENUM[active,inactive], metadata JSONB, created_at
)

event_resources(
  event_id UUID FK, resource_id UUID FK, PRIMARY KEY(event_id, resource_id)
)

reminders(
  id UUID PK, event_id UUID FK, type ENUM[email,sms,in_app],
  send_at TIMESTAMPTZ, status ENUM[pending,sent,failed], created_at
)

calendar_integrations(
  id UUID PK, user_id UUID, provider ENUM[google,outlook,ics], access_token
TEXT encrypted,
  refresh_token TEXT encrypted, expires_at TIMESTAMP, sync_enabled BOOL,
  last_sync_at TIMESTAMP, created_at
)
```

Indexes:

- `events(start_time, end_time, calendar_id)`
- `participants(user_id, event_id)`
- `slots(mentor_id, status)`
- `availabilities(mentor_id, day_of_week)`

7) APIs (prefix `/api/v1/schedule`)

Calendars

- `GET /calendars?owner_type=&owner_id=`
- `POST /calendars` / `PUT /calendars/{id}` / `DELETE /calendars/{id}`

Events

- `GET /events?calendar_id=&from=&to=&category=`
- `POST /events` / `PUT /events/{id}` / `DELETE /events/{id}`
- `POST /events/{id}/participants` (invite)
- `POST /events/{id}/cancel`
- `POST /events/{id}/reschedule`
- `GET /events/{id}/ics` → `.ics` download

Availability & Slots

- `GET /availability?mentor_id=`
- `POST /availability`
- `GET /slots?mentor_id=&from=&to=`
- `POST /slots/{id}/book` / `POST /slots/{id}/cancel`

Integrations

- `POST /integrations/connect` (OAuth2 handshake)
- `POST /integrations/sync`
- `GET /integrations/status`

Reminders & Attendance

- `POST /events/{id}/checkin` / `POST /events/{id}/checkout`
- `GET /attendance?cohort_id=&from=&to=`
- `POST /reminders/schedule` / `GET /reminders?event_id=`

8) Process Flows

8.1 Mentorship Booking Flow

1. Mentor sets weekly availability.
2. Student selects slot → `POST /slots/{id}/book`.
3. System checks conflicts → creates event in both calendars.
4. NAS sends confirmation + reminders.
5. Event syncs to external calendar (Google/Outlook) if connected.

8.2 Program Calendar Setup

1. Program Director defines term dates and blackout days.
2. Bulk imports training events via CSV or API.
3. Events distributed across cohort calendars.
4. Automated reminders & attendance tracking per session.

8.3 Reschedule Flow

1. Organizer requests new slot → `/events/{id}/reschedule`.
2. Notify all participants; handle auto-accept rules.
3. Update event times in SCH + external calendars.

8.4 Attendance Tracking

1. Student joins via secure "Join" link → check-in auto-logged.
2. If absent beyond grace period, system auto-marks as `absent`.
3. Mentors manually adjust status post-session if needed.
4. Analytics module aggregates attendance stats.

9) Integrations

Module	How it Integrates
Mentorship	Schedules mentorship sessions, syncs feedback forms & attendance
Program Mgmt	Maintains cohort calendars and academic timetables
Payment	Adds renewal/payment reminders to calendar
Notifications (NAS)	Sends event reminders & updates
Analytics	Exports attendance and punctuality metrics
Portfolio	Tags portfolio submissions with completion dates
External Calendars	Bi-directional sync for personal convenience

10) UI/UX Requirements

- **Calendar Views:** Day / Week / Month toggle.
- Color-coded events (mentorship, training, review, etc.).
- **Slot Picker:** grid showing mentor availability; timezone auto-adjust.
- Drag-and-drop rescheduling (with permission).
- **Filter:** category, mentor, cohort, student.
- **Attendance Pane:** mark present/absent; export CSV.
- **Reminder Configurator:** select reminder times and channels.
- **Mobile View:** responsive, minimal taps for booking.

Accessibility: keyboard navigation, high contrast mode, screen reader support.

11) Security & Compliance

- JWT-secured APIs (scope: `schedule.*`).
- Only event participants can view private events.
- Tokenized external calendar credentials; encrypted at rest.
- Audit trail for bookings/reschedules/cancellations.
- Retention policy: attendance logs kept 5 years (per CPPC).
- PII minimization: only participant names/emails (no sensitive data in descriptions).

12) Tech Stack

Component	Choice
Backend	FastAPI / NestJS
DB	PostgreSQL 15 (with timezone-aware TIMESTAMPTZ)
Cache	Redis (slot caching, reminders queue)
Job Queue	Celery / BullMQ for reminders & sync jobs
Integrations	Google Calendar API, Microsoft Graph API

Component	Choice
ICS	icalendar / fullcalendar libraries
Observability	OpenTelemetry, Prometheus, Grafana, Sentry
Frontend	React/Next.js with FullCalendar.js component
Notifications	Integrates via NAS microservice
CI/CD	Test recurring patterns, DST correctness, API contract tests

13) Error Handling

Scenario	Handling
Time conflict	409 with conflicting event IDs
Slot expired	410; suggest next available slot
External sync failure	Retry with exponential backoff; mark sync_failed; alert user
Invalid timezone	400 with list of valid IANA zones
Event overlap limit	Soft warning (user override allowed for admin)
Reminder failure	Retry 3× then mark failed in logs
API timeout	Queue job and respond 202 Accepted

14) Future Enhancements

- **AI-assisted scheduling:** suggest optimal times based on availability & engagement history.
- **Group auto-matching:** automatically form study or project groups.
- **Predictive attendance alerts:** detect likely absentees.
- **Smart load balancing:** distribute sessions evenly across mentors.
- **ICS feed generator:** cohort-specific calendar feeds.
- **Integration with Zoom/Meet APIs** for auto-link creation.