**Phase 0 — Repository & Local Baseline**

1. Clone & run quick health check
   * Action: git clone https://github.com/nimamakhmali/academic-data-platform.git && cd academic-data-platform then docker compose up --build or cd backend && python -m app.
   * Acceptance: backend /health returns OK (per README). [GitHub](https://github.com/nimamakhmali/academic-data-platform/blob/main/README.md)
2. Establish developer environment files
   * Action: add .env.example → .env, pin Python version, create backend/requirements.txt virtualenv instructions in README.
   * Acceptance: pip install -r backend/requirements.txt works locally.
3. Create issue/milestone board derived from docs
   * Action: Convert RFP/SRS items into issues & milestones in repo (backlog, sprint 1, ...).
   * Acceptance: All SRS features exist as issues with priority and estimate. [GitHub](https://github.com/nimamakhmali/academic-data-platform/tree/main/docs)

**Phase 1 — Requirements → Backlog → Implementation Plan**

1. Review RFP & SRS and produce prioritized feature list
   * Action: Map each SRS requirement to 1–3 user stories and acceptance criteria.
   * Acceptance: Document docs/implementation-plan.md with prioritized backlog.
2. Map HLA/Architecture → components & APIs
   * Action: For each high-level component (auth, users, students, faculty, courses, enrollments, research, ML, reporting, dashboard) create component spec & API contracts.
   * Acceptance: Each component has an owner, API surface, and dependencies listed.

**Phase 2 — Database & Data Models (ERD → Models → Migrations)**

1. Convert ERD → SQLAlchemy models
   * Action: Implement models in backend/app/models/ (suggested: User, Role, Student, Faculty, Course, Enrollment, Department, ResearchProject, Publication, AuditLog, Token).
   * Acceptance: Models reflect ERD attributes & relationships; model tests for relationships exist.
2. Add Pydantic schemas
   * Action: Create backend/app/schemas/ (request/response DTOs) with examples and field validations.
   * Acceptance: OpenAPI shows correct schemas.
3. Set up migrations (Alembic)
   * Action: Add Alembic config, create initial migration: alembic revision --autogenerate -m "initial" then alembic upgrade head.
   * Acceptance: DB schema matches models after migration; seed script adds admin user.
4. Add DB indexes and constraints
   * Action: Add composite indexes for common queries (e.g., (department\_id, program)) and appropriate FK cascades.
   * Acceptance: Explain indexing rationale in docs/er-performance.md.

**Phase 3 — Backend API Implementation (FastAPI)**

1. Implement API router skeletons
   * Action: backend/app/api/v1/ routers: auth.py, users.py, students.py, faculty.py, courses.py, enrollments.py, research.py, reports.py, ml.py, health.py, metrics.py.
   * Acceptance: Each router registers under /api/v1/ and appears in OpenAPI.
2. CRUD + service layer
   * Action: Implement backend/app/crud/ (DB operations) and backend/app/services/ (business logic). Keep controllers thin.
   * Acceptance: Unit tests for CRUD ops and service logic.
3. Centralized error handling & validation
   * Action: Add exception handlers for HTTPException, validation errors, DB errors; consistent error response format.
   * Acceptance: All endpoints return consistent error schema.
4. API documentation & examples
   * Action: Ensure every endpoint has summary, tags, request/response examples. Publish OpenAPI JSON to docs/openapi.json.
   * Acceptance: Swagger UI shows comprehensive docs. [GitHub](https://github.com/nimamakhmali/academic-data-platform/tree/main/docs)

**Phase 4 — Authentication & Authorization (OAuth2 / JWT / RBAC)**

1. Implement secure auth core utilities
   * Action: backend/app/core/security.py — password hashing (bcrypt), token creation (access, refresh), token expiry.
   * Acceptance: Auth unit tests verifying token generation & validation.
2. OAuth2/JWT endpoints
   * Action: POST /api/v1/auth/token (password grant), POST /api/v1/auth/refresh, POST /api/v1/auth/revoke.
   * Acceptance: Tokens include roles/claims; protected endpoints reject invalid/expired tokens.
3. Implement Role-Based Access Control (RBAC)
   * Action: Role model + permission table; dependency get\_current\_user returns user with roles; decorator or dependency require\_role("admin").
   * Acceptance: Endpoints protected by role; tests for role restrictions.
4. Admin & bootstrap users
   * Action: Seed an admin user and role during migrations or via seed script.
   * Acceptance: Admin can create other users & assign roles.

**Phase 5 — Caching, Background Jobs & Async Tasks**

1. Integrate Redis for caching & broker
   * Action: Add Redis client usage for query caching and rate limiting.
   * Acceptance: Cache hit/miss logic present; rate limiter working.
2. Implement task queue (Celery or Dramatiq)
   * Action: celery with Redis broker for background tasks: report generation, ML training triggers, email/notifications, ETL.
   * Acceptance: Background tasks run reliably; retry configured.
3. Email & Notification subsystem
   * Action: Add template-based emails and queued deliveries.
   * Acceptance: Notifications are queued and delivered (or simulated in dev).

**Phase 6 — Machine Learning / Analytics Pipeline**

1. Create reproducible ML pipeline
   * Action: ml/ with data\_prep.py, train.py, evaluate.py, infer.py, and an artifacts directory; use MLflow or simple model registry.
   * Acceptance: Training script produces saved model artifact and evaluation metrics.
2. Integrate model training into background jobs
   * Action: Trigger train via Celery task with dataset snapshot ID.
   * Acceptance: Task logs metrics and stores model metadata.
3. Add inference endpoint & monitoring
   * Action: POST /api/v1/ml/infer for predictions; model versioning in responses.
   * Acceptance: Inference latency acceptable; metrics tracked.

**Phase 7 — Frontend (React) — new folder frontend/**

1. Create React app (TypeScript recommended)
   * Action: npx create-react-app frontend --template typescript or Vite + React; folder frontend/.
   * Acceptance: npm start runs dev server.
2. Implement auth flow & token storage
   * Action: Login page, token storage via secure cookie or memory + refresh token flow.
   * Acceptance: Authenticated API calls succeed; route guards in place.
3. Implement main pages & components
   * Action: Pages: Dashboard, Students, Faculty, Courses, Enrollment, Research, Reports, Admin; components: Table, Form, Modal, Chart, Filters, Pagination.
   * Acceptance: Core CRUD flows work end-to-end.
4. Data visualization & dashboards
   * Action: Use charting library; implement interactive filters, export to CSV/PDF.
   * Acceptance: Dashboard shows KPIs described in docs.
5. Responsive UI & accessibility checks
   * Action: Ensure mobile view and basic a11y compliance.
   * Acceptance: No critical accessibility violations.

**Phase 8 — Reporting, Exports & Scheduled Tasks**

1. Implement scheduled reporting tasks
   * Action: Weekly/monthly reports via Celery; store generated files in storage (S3-compatible).
   * Acceptance: Reports accessible via GET /api/v1/reports/{id}/download.
2. Add ad-hoc report builder
   * Action: UI to pick filters and generate report (CSV/PDF).
   * Acceptance: Exports match backend filters and column selections.

**Phase 9 — Testing Strategy (Unit / Integration / E2E)**

1. Backend unit tests (pytest)
   * Action: Add backend/tests/unit/ and backend/tests/integration/ with fixtures and a test DB (docker-compose or sqlite).
   * Acceptance: pytest -q passes for core modules.
2. Integration & contract tests vs OpenAPI
   * Action: Tests assert endpoint behavior matches OpenAPI contract.
   * Acceptance: Contract tests pass.
3. Frontend tests & E2E
   * Action: Jest/unit tests and Cypress or Playwright for full flows (login → create student → view report).
   * Acceptance: E2E flows stable in CI.

**Phase 10 — CI/CD, Containerization & Deployment**

1. GitHub Actions workflows
   * Action: Add .github/workflows/ci.yml for lint → unit tests → build; cd.yml for deployment to staging.
   * Acceptance: PRs run CI and merge only if passing.
2. Dockerfiles & docker-compose improvements
   * Action: Create backend/Dockerfile, frontend/Dockerfile, update docker-compose.yml with services: postgres, redis, backend, frontend, ml-worker.
   * Acceptance: docker compose up --build boots full stack.
3. Production manifests
   * Action: Kubernetes manifests / Helm chart or cloud service deployment scripts; use secrets manager for credentials.
   * Acceptance: One-click deploy to staging environment.

**Phase 11 — Security, Privacy & Compliance**

1. Threat model & security checklist implementation
   * Action: Implement items from docs/security-privacy (input validation, TLS, CORS, CSP, SQL injection protection, rate-limiting, logging).
   * Acceptance: Security doc checklist marked “done” and basic SAST/Bandit scan passes.
2. Data protection & PII handling
   * Action: Encryption at rest for sensitive fields, data retention & deletion scripts, audit logging for PII access.
   * Acceptance: Data deletion workflow tested and audit logs available.
3. Secrets & config management
   * Action: Move secrets to env vars or Vault; never commit secrets.
   * Acceptance: No secrets in repo; CI reads secrets from secure store.

**Phase 12 — Observability, Monitoring & Reliability**

1. Structured logging & correlation IDs
   * Action: Use JSON logs, include request id, user id in logs.
   * Acceptance: Logs searchable and consistent.
2. Metrics & dashboards
   * Action: Expose Prometheus metrics; create Grafana dashboards for system and business KPIs.
   * Acceptance: Dashboards show key metrics.
3. Error tracking & alerts
   * Action: Integrate Sentry (or equivalent) and configure alerts for high-error rates.
   * Acceptance: Alerting configured for production/staging.

**Phase 13 — Documentation, Handover & Onboarding**

1. Finalize developer docs & runbooks
   * Action: docs/developer-setup.md, docs/deployment-runbook.md, docs/backup-restore.md.
   * Acceptance: New developer can run local dev in <10 steps.
2. API consumer docs & SDKs
   * Action: Host Swagger, add short example SDK (Python) or Postman collection.
   * Acceptance: External system can call APIs with provided examples.
3. Product & user documentation
   * Action: User manual for admin and staff, screenshots, FAQ.
   * Acceptance: Non-dev user can perform core tasks following docs.

**Phase 14 — Final QA, Load Testing & Release**

1. Performance & load testing
   * Action: Run JMeter / k6 scenarios for typical and peak loads; tune DB indexes and queries.
   * Acceptance: Application meets performance targets in docs.
2. Backup/Restore validation
   * Action: Test full DB backup and restore, test config restore.
   * Acceptance: Restore completes and data integrity confirmed.
3. Release & post-release validation
   * Action: Release tag, changelog, update README and docs, onboard first customer/stakeholder.
   * Acceptance: Release checklist passed and stakeholders sign off.

**Phase 15 — Ongoing: Maintenance, Roadmap & Improvements**

1. Roadmap execution
   * Action: Implement roadmap items from docs/roadmap.md in prioritized sprints.
   * Acceptance: Roadmap updated after each release.
2. Technical debt & refactor plan
   * Action: Regularly schedule time for refactors, tests, and documentation updates.
   * Acceptance: Tech-debt tickets identified and tracked.

**Progress estimate (based on repo contents & docs visible)**

* Design & documentation (RFP, SRS, Architecture, ERD, Roadmap): **~60–70% complete**. [GitHub](https://github.com/nimamakhmali/academic-data-platform/tree/main/docs)
* Backend scaffold (FastAPI structure, quickstart, docker-compose): **~20–30% implemented** (skeleton exists but feature-complete CRUD/auth/ML not finished). [GitHub+1](https://github.com/nimamakhmali/academic-data-platform/blob/main/README.md)
* Frontend: **0% or not present in this repo** (React mentioned in README but no frontend folder visible). [GitHub](https://github.com/nimamakhmali/academic-data-platform/blob/main/README.md)
* **Overall rough completion:** **~20–25%** (docs strong, core code scaffold present). [GitHub+1](https://github.com/nimamakhmali/academic-data-platform)

**Notes & next steps I can do for you (pick any, I’ll act immediately)**

* File-by-file audit: I can open and review each backend file and produce per-file comments (security, style, missing tests). *(If you want that, either grant raw file content or paste file(s) here.)*
* Generate a prioritized first 6-week sprint plan (milestones + deliverables) from the above checklist.
* Produce a PR template / CI GitHub Actions starter file / Dockerfile examples.