InvestigaODS

Arquitectura, Modelo de Datos, Planes (Student Basic/Pro), API y Fases de Trabajo

1) Resumen ejecutivo

Objetivo. Plataforma de capacitación ambiental con modalidades autodidacta y guiada, evaluaciones y certificación, con integración futura de un asistente inteligente (chatbot).

Stack y enfoque: Frontend (React + Vite + Tailwind), Backend (NestJS + TypeORM), Base de Datos (MySQL), todo orquestado con Docker. El chatbot no se implementa aún, pero se reserva el endpoint /chat/message y/o un canal WebSocket/SSE para su conexión posterior.

Segmentación de estudiantes: Se incorporan dos niveles—Student Basic y Student Pro—para habilitar acceso diferenciado a cursos y funcionalidades. Se diseña escalabilidad hacia pasarela de pagos para cursos arancelados y convenios futuros.

2) Roles y permisos (RBAC)

- Roles: ADMIN, INSTRUCTOR, STUDENT (con plan BASIC o PRO).
- ADMIN: gobierno del sistema (usuarios/roles, catálogos, auditoría, certificados globales).
- INSTRUCTOR: crea y gestiona sus cursos, módulos, lecciones, quizzes; ve progreso de sus alumnos; emite certificados de sus cursos.
- STUDENT: se inscribe, consume contenidos, rinde exámenes, descarga sus certificados.
- Autorización en NestJS: JwtAuthGuard + RolesGuard + políticas por propietario (ownerId).

3) Planes de estudiante (Basic vs Pro)

Comparativa funcional por plan:

Funcionalidad	Student Basic	Student Pro
Acceso a cursos autodidacta (self-paced)	Sí	Sí
Acceso a cursos guiados (cohortes/plazos)	No	Sí
Certificados verificables	No	Sí
Clases sincrónicas (en vivo)	No	Sí
Desafíos con puntaje / gamificación	No	Sí
Evaluaciones avanzadas / proyectos	No	Sí
Soporte prioritario / comunidad privada	No	Sí

Preparado para pasarela de pagos*	N/A	Sí

*Activación cuando existan convenios/comercialización.

4) Arquitectura técnica

Frontend

- React + Vite + Tailwind; TanStack Query para data fetching/cache y Zustand para estado UI/autenticación.
- Rutas públicas: /, /login, /register, /courses, /courses/:slug.
- Rutas privadas estudiante: /dashboard, /learn/:courseId/:lessonId, /certificates.
- Rutas privadas instructor: /instructor, /instructor/courses, /instructor/courses/:id/builder, /instructor/courses/:id/students.
- Rutas privadas admin: /admin, /admin/users, /admin/catalog.
- Gating por plan: badges FREE/BASIC/PRO y callouts de upgrade cuando el plan no cumple requisitos.
- Widget ChatDock (UI) deshabilitado por ahora, listo para integrarse a /chat/message.

Backend

- NestJS modular: Auth, Users, Courses, Content (Modules/Lessons), Enrollments, Progress, Quizzes, Attempts, Certificates, Chat (placeholder), Admin, Storage, Audit.
- TypeORM con MySQL, class-validator, documentación con Swagger, testing con Jest.
- RBAC con guards y políticas de propiedad; CORS y cookies httpOnly para refresh token.

Infraestructura

- Docker Compose (dev) con servicios: db (mysql:8), phpmyadmin, backend (NestJS), frontend (Vite).
- Puertos: MySQL 3306, phpMyAdmin 8081, backend 3000, frontend 5173.
- Variables críticas: VITE_API_BASE_URL=http://localhost:3000 (frontend); conexión DB via env (backend).

5) Modelo de datos (MySQL + TypeORM)

Entidades núcleo:

- User: email (uniq), passwordHash, firstName, lastName, avatarUrl?, role (ADMIN|INSTRUCTOR|STUDENT).
- Course: title, slug (uniq), summary, description, thumbnailUrl?, level, language, visibility (PUBLIC|PRIVATE), modality (SELF_PACED|GUIDED), ownerId (User).
- Cohort (solo GUIDED): courseId, name, startAt, endAt, capacity?.
- Module: courseId, index, title, summary?.

- Lesson: moduleId, index, title, content (MD/HTML), videoUrl?, durationMin?, resources (JSON[]).
- Enrollment: userId, courseId, cohortId?, status (ACTIVE|COMPLETED|DROPPED), enrolledAt.
- LessonProgress: userId, lessonId, completed, progressPct (0-100), lastViewedAt.
- Quiz: courseId?/lessonId?, title, type (QUIZ|EXAM), passScore, attemptLimit?, timeLimitSec?, weight?.
- Question: quizId, type (MCQ|TRUE_FALSE|OPEN), prompt, points, metadata?.
- Option: questionId, text, isCorrect, explanation?.
- Attempt: quizId, userId, startedAt, submittedAt?, score, status (IN_PROGRESS|SUBMITTED|GRADED).
- Answer: attemptId, questionId, optionId?/openText?, isCorrect?, awardedPoints?.
- Certificate: userId, courseId, cohortId?, serial (uniq), pdfUrl, hashSha256, issuedAt.
- Tag y CourseTag para etiquetado.
- AuditLog para trazabilidad.

Extensiones para planes y funcionalidades Pro:

- MembershipPlan: id, code (BASIC|PRO), name, features JSON, status, createdAt.
- Subscription: id, userId, planId, startAt, endAt?, status (ACTIVE|CANCELLED|EXPIRED).
- Course.tierRequired: FREE|BASIC|PRO + flags: hasCertificate, supportsLive, supportsChallenges.
- LiveClass: id, courseId/cohortId?, title, startAt, endAt, meetingUrl, recordingUrl?, capacity, timezone.
- Challenge: id, courseId/lessonId, title, description, points, rules JSON; ChallengeSubmission: userId, challengeId, artifactUrl?, score, status.
- UserPoints: userId, courseId?, points (leaderboard).
- Placeholders de pagos: PaymentProvider/PaymentIntent (a futuro).

6) API (contratos resumidos)

Autenticación

- POST /auth/register → { email, password }
- POST /auth/login → { accessToken, user }
- POST /auth/refresh (cookie httpOnly)
- POST /auth/logout

Usuarios

- GET /users/me, PATCH /users/me
- GET /admin/users, PATCH /admin/users/:id (Admin)

Cursos y contenido

- GET /courses (filtros: q, tag, modality, owner, tier)
- POST /courses (Instructor/Admin), GET|PATCH|DELETE /courses/:id
- GET /courses/:id/outline, POST /courses/:id/modules, POST /modules/:id/lessons

Inscripciones y progreso

- POST /courses/:id/enroll (Student)
- GET /me/enrollments, GET /courses/:id/students (Instructor/Admin)
- POST /lessons/:id/progress, GET /me/courses/:cid/progress

Quizzes y exámenes

- POST /lessons/:id/quizzes (Instructor/Admin), GET /quizzes/:id
- POST /quizzes/:id/attempts, POST /attempts/:id/answers, POST /attempts/:id/submit, GET /attempts/:id/result

Certificados

- POST /courses/:id/certificates/issue (Instructor/Admin)
- GET /me/certificates
- GET /certificates/verify?serial=...

Planes y gating (nuevo)

- GET /plans → planes disponibles (BASIC, PRO) y features
- GET /me/subscription → plan y estado actuales del estudiante
- POST /subscriptions/upgrade → { planCode } (mock por ahora)
- Guardas: si course.tierRequired=PRO y el usuario no es Pro → 403
- Live classes: GET /courses/:id/live-classes (Pro), POST (Instructor)
- Challenges: CRUD para instructor; submissions e intentos (Pro)

Chat (placeholder)

- POST /chat/message (se conectará a servicio IA más adelante)
- WebSocket/SSE reservado: /chat/stream

7) Fases de trabajo

- Fase A — Infra y cimientos: Docker Compose, conexión NestJS+TypeORM+MySQL, seeds (Admin).
- Fase B — Identidad y RBAC: Auth (login/refresh/logout), RolesGuard, políticas por propietario.
- Fase C — Cursos y Contenido: CRUD de Course/Module/Lesson; catálogo y detalle en frontend; course builder básico.

- Fase D — Inscripciones y Progreso: Enrollments y LessonProgress; dashboard estudiante.
- Fase E — Evaluaciones: Quizzes/Questions/Attempts/Answers; auto-calificación; runner en frontend.
- Fase F — Certificación: generación de PDF y hash; emisión y verificación pública.
- Fase G — Planes y gating: MembershipPlan, Subscription, tierRequired por curso, página Planes y UI de upgrade.
- Fase H — Clases en vivo y Desafíos (Pro): LiveClass; Challenges + puntos + leaderboard.
- Fase I — QA y Despliegue: pruebas, seguridad (CORS/headers), logging y despliegue.

8) Criterios de aceptación

MVP general:

- Autenticación con JWT + refresh; roles verificados por guards.
- Catálogo y detalle de cursos; inscripción; progreso por lección.
- Exámenes con corrección automática y visualización de resultados.
- Emisión y descarga de certificados con hashSha256 verificable.
- UI responsive con paleta de GREEN DREAMS.

Específicos de planes:

- Student tiene plan BASIC o PRO con Subscription.ACTIVE.
- Cursos tier=BASIC se consumen con BASIC o PRO; tier=PRO solo con PRO.
- Clases en vivo, desafíos y certificados avanzados se limitan a PRO.
- Instructores/Admin gestionan sin restricciones de plan.

9) Siguientes pasos inmediatos

- Confirmar enums y campos finales (incl. course.level y flags).
- Implementar backend: módulos Auth, Users y Courses con Swagger.
- Exponer /plans y /me/subscription (mock) y conectar el gating en frontend.
- Integrar catálogo real y login en frontend; pruebas end-to-end en Docker.