Arquitectura de software

La **arquitectura de software** que usarás es un **monolito modular con Arquitectura Hexagonal (Puertos y Adaptadores)**. Eso significa: una sola app (fácil de construir y desplegar) pero **ordenada en capas** con reglas claras y piezas intercambiables.

**Qué es cada capa (en simple)**

1. **Interfaces (UI/API) – “la puerta”**  
   Controladores HTTP (REST). Validan lo básico y convierten JSON ⇄ DTO. *No hacen reglas de negocio.*
2. **Aplicación (Servicios de casos de uso) – “el orquestador”**  
   Métodos como iniciarSesion(), responder(), obtenerResultado().  
   Abren **transacciones**, llaman repositorios y entidades del dominio, y devuelven DTOs.  
   Aquí también aplicas **autorización por rol**.
3. **Dominio – “las reglas del juego”**  
   Entidades y lógica: Usuario, Categoria, Modulo, Ejercicio, **Palabra**, SesionPractica, Intento, Resultado.  
   Invariantes: una **Sesión FINALIZADA no se edita**, un **Módulo PUBLICADO** solo con ejercicios válidos, **Ejercicio–Palabra** es relación de uso (M:N).  
   *No sabe de BD, HTTP ni caché.*
4. **Infraestructura (Adaptadores) – “los enchufes”**  
   Implementaciones de puertos (interfaces) para hablar con **PostgreSQL** (repos), **Redis** (caché/colas opcionales) y **almacenamiento de objetos** (medios).  
   Si mañana cambias la BD, cambias estos adaptadores y listo.

**Cómo viaja una petición (ejemplo real)**

**UC07: Responder un ejercicio**

1. **API** recibe POST /sesiones/{id}/intentos con la respuesta.
2. **Aplicación** (PracticaService) valida: la sesión es tuya y está *EN\_CURSO*.
3. Pide a **Infra** (repo) el **Ejercicio** desde Postgres.
4. **Dominio** ejecuta Ejercicio.evaluar(respuesta) → correcto/puntaje.
5. **Aplicación** guarda el **Intento** (repo) dentro de una **transacción** y responde con el feedback.
6. Si era el último, **cierra la sesión** y genera **Resultado**.

**Por qué así (beneficios concretos)**

* **Rápido de construir** (MVP) y **fácil de cambiar**: nuevas pantallas o tipos de ejercicio no rompen el núcleo.
* **Reglas claras** en el Dominio (menos bugs).
* **Intercambiable**: si cambias Postgres o sumas un buscador, tocas solo **adaptadores**.
* **Escalable**: replicas de la app y de la BD, y cacheas catálogo en Redis.

**Dónde encajan tus datos (lo clave de tu modelo)**

* **Catálogo**: Categoria → Modulo → Ejercicio
* **Vocabulario**: **Palabra** pertenece a una Categoria y se relaciona **M:N** con Ejercicio (tabla puente en BD).
* **Práctica**: SesionPractica → Intento → Resultado
* **Progreso**: Estadistica, Logro

**Estructura mínima del proyecto (mapa mental)**

/interfaces # controladores REST y DTOs

/aplicacion # servicios de casos de uso + puertos (interfaces)

/dominio # entidades y reglas (sin dependencias externas)

/infraestructura # repos Postgres, Redis, storage (adaptadores)

/tests # unit/integration/e2e

**Qué programas primero (orden recomendado)**

1. Entidades de **Dominio**: SesionPractica, Intento, Ejercicio.evaluar(), Resultado.
2. **Puertos** (interfaces): ICatalogoRepo, IPracticaRepo, IUsuarioRepo.
3. **Servicios de Aplicación**: PracticaService, CatalogoService, ResultadosService.
4. **Adaptadores Postgres** (repos) y **controladores REST**.
5. **Redis** para cachear catálogo (luego).

Tecnologias

**Frontend (Web / PWA)**

* React + TypeScript (con Vite)
* UI: Tailwind CSS + shadcn/ui
* Estado y data: React Query (fetching) + Zustand (estado global)
* Formularios/validación: React Hook Form + Zod
* Ruteo: React Router
* Tests: Vitest + React Testing Library
* E2E: Cypress

**Móvil (opcional, fase 2)**

* React Native con Expo (o mantener solo PWA en MVP)

**Backend (API)**

* Node.js + NestJS
* ORM: Prisma
* Autenticación/autorización: JWT (Passport) o Auth0/Supabase Auth
* Documentación: OpenAPI/Swagger
* Tests: Jest + Supertest

**Datos y archivos**

* Base de datos: PostgreSQL
* Cache/colas: Redis (para jobs y rate limiting)
* Almacenamiento de medios (audio/imágenes): S3 compatible (AWS S3 o Cloudflare R2)

**Funcionalidades “edtech”**

* STT (voz→texto) y TTS (texto→voz): servicios gestionados (Google Cloud Speech, Azure Speech, Amazon Polly)
* (Opcional) Generación/corrección de ejercicios: API de LLM

**Infra & DevOps**

* Monorepo con pnpm/Turborepo
* CI/CD: GitHub Actions
* Contenedores: Docker
* Deploy: Vercel (frontend), Railway/Fly.io/Render (API + DB)
* Observabilidad: Sentry (errores) + PostHog (analítica de producto)
* Logs: cualquier proveedor gestionado (p. ej., Logtail)

**Calidad de código**

* ESLint + Prettier + Husky (pre-commits)