PROJECT ABSTRACT

Developer API Documentation Portal with Live Try-It Console

A developer docs portal that renders endpoint reference from an OpenAPI 3.1 spec, with a live try-it console, API key management and changelog. Delivered built-to-order with the working portal, demo API, report, PPT and viva kit.

01. Title

Developer API Documentation Portal with Live Try-It Console

02. Introduction / Background

Every backend project needs documentation, yet student APIs ship with a README paragraph. Professional API docs are generated from a machine-readable spec, stay in sync with the code, and let developers execute calls without leaving the page. This project builds that experience: an OpenAPI 3.1 spec drives endpoint pages, code samples and the try-it console's parameter forms, so docs and playground can never drift apart — plus the operational side developers expect: keys, versions, changelog and an error catalog.

03. Problem Statement

Hand-written API docs rot: parameters change, examples go stale, and 'try it' means pasting curl into a terminal. The project addresses this with spec-driven rendering — one OpenAPI file as the single source of truth for pages, samples and console forms — and a sandbox proxy so the console executes real requests without ever exposing secrets to page JavaScript.

04. Objectives

- Render endpoint pages (parameters, schemas, errors) from a single OpenAPI 3.1 spec file.
- Provide a live try-it console that sends real requests through a sandbox proxy and pretty-prints responses.
- Generate copy-paste code samples (curl, Python, JavaScript) from the same spec.
- Manage API keys: issue, revoke, rotate, with per-key monthly usage meters (keys stored hashed).
- Support multiple spec versions with a switcher, breaking-change flags and a changelog.
- Deliver the complete student kit: portal + demo API source, sample spec, report, PPT and viva Q&A.;

05. Proposed Methodology

The system is developed in three stages. (1) Spec pipeline: the OpenAPI YAML/JSON is parsed into endpoint, schema and error models at startup. (2) Rendering: a docs renderer

turns each endpoint into a page with parameter tables, request/response examples and error tables; the console builds its forms from the same parameter definitions. (3) Proxy and keys: the browser posts to the portal's sandbox proxy, which attaches the demo key server-side, forwards, and returns status/headers/body; keys are hashed at rest with usage counters per call.

06. System Architecture / Working

Express serves the portal and the sandbox proxy; a demo orders/payments API is the executable target. Publishing a spec version snapshots the previous one for the version switcher. The console's send flow goes browser → proxy → sandbox API, so no secret reaches page JS; responses render with syntax highlighting, timing and size, and errors deep-link into the error catalog. Redis rate-limits the sandbox (100 req/min per key, design target).

07. Hardware / Software Requirements

- Node.js 18+ with Express
- OpenAPI 3.1 spec (YAML/JSON) as content source
- PostgreSQL (keys, usage, spec versions)
- Redis (rate limiting, response cache)
- Demo orders/payments API (included)
- Docker + docker-compose

08. Expected Outcomes

A docs site where swapping the OpenAPI file re-renders every page, sample and console form with no code changes; a console that executes real sandbox calls with sub-50 ms proxy overhead (design target); and a documented key-security and proxy design the student can defend, including idempotency-key handling on the demo orders API.

09. Applications

- Documenting any student-built REST API for its users
- College API projects and hackathon backend submissions
- SaaS-style developer portals for startup MVPs
- Internal API references for placement-project portfolios

10. Future Scope

- Typed SDK generation (TypeScript, Python) from the spec
- WebSocket and GraphQL documentation support
- Interactive OAuth2 authorization-code flow in the console
- Usage-based billing meters per key