PROJECT ABSTRACT

REST API Testing Workbench

A Postman-style API testing workbench for the browser with a request builder, header/body/auth tabs, a simulated e-commerce API, syntax-highlighted responses, collections and history. Delivered built-to-order with the web app, report, PPT and viva kit.

01. Title

REST API Testing Workbench

02. Introduction / Background

Testing a REST API is a daily engineering task: developers fire GET, POST, PUT and DELETE requests, inspect status codes and JSON bodies, and iterate. Desktop tools do this well but need installation and accounts; students in a lab often just need to understand methods, status codes, headers and request bodies. This project builds ReqForge, a REST API testing workbench that runs as a single web page: a request builder with method/URL/headers/body/auth tabs, a mock e-commerce API simulated in the browser (products and orders with full CRUD), syntax-highlighted JSON responses with timing, a saved collection and a request history — teaching real API-testing workflow with zero setup.

03. Problem Statement

Students learn REST theory (methods, status codes, JSON) but rarely practice the request/response loop because lab machines lack installed API clients and public test APIs are rate-limited or unreliable. The project addresses this with a self-contained workbench: the request builder, a deterministic mock API, response inspection and history all in one page, so every HTTP concept in the syllabus is demonstrated live and repeatably.

04. Objectives

- Provide a request builder: HTTP method, URL, headers, JSON body and bearer-token auth tabs.
- Simulate a realistic e-commerce API in-browser (products + orders, full CRUD, proper status codes: 200/201/400/404/422).
- Render responses with JSON syntax highlighting, status line, timing and request-id metadata.
- Ship a prebuilt request collection (list/get/create/update/delete flows) and a clickable history.
- Include an API reference view documenting every mock endpoint, parameters and error codes.
- Deliver the complete student kit: web app, report, PPT and viva Q&A.;

05. Proposed Methodology

The workbench has three parts. (1) Request builder UI: method selector, URL bar, tabbed headers/body/auth panels, and a Send button with a loading state. (2) Mock API router: a JavaScript function parses the URL path and method, matches routes (collection vs item,

products vs orders), validates JSON bodies, and returns response objects with realistic status codes and latency simulation. (3) Response pipeline: the JSON is pretty-printed with a tokenizer-based syntax highlighter, and each exchange is appended to history for one-click replay. An in-page docs view is generated from the same route table.

06. System Architecture / Working

On Send, the builder assembles the request (attaching the bearer token as an Authorization header) and passes it to the mock router after a short simulated latency. The router matches method+path against the route table: GET /products supports limit and category filters, item routes do lookup by id, POST validates required fields (422 on failure, 400 on malformed JSON), PUT merges partial updates, DELETE returns the removed record. The response renderer shows the status code with its reason phrase, elapsed milliseconds, payload size and a generated request id, then the highlighted JSON body.

07. Hardware / Software Requirements

- Any modern web browser — no installation, no backend server, no accounts
- Single HTML/CSS/JS file (vanilla JavaScript, no dependencies)
- Text editor for code walkthrough (VS Code recommended)
- Report: LibreOffice/MS Word; presentation: PowerPoint-compatible slides

08. Expected Outcomes

- A working workbench where all 7 mock endpoints respond with correct status codes and JSON.
- Demonstrated CRUD flows: create a product, read it back, update it, delete it.
- Visible teaching of 200/201/400/404/422 semantics through the response panel.
- A complete report, PPT and viva Q&A; on REST, HTTP methods, status codes and API testing practice.

09. Applications

- Web-technology lab sessions on HTTP and REST without external dependencies.
- Frontend students can test UI code against the mock API's predictable responses.
- Demonstrating request/response debugging in project reviews.

10. Future Scope

- Environment variables and request chaining (use one response in the next request).
- Import/export collections in a portable JSON format.
- WebSocket testing tab and GraphQL query panel as optional extensions.