# Backend API Design

Aditi edited this page 2 weeks ago · 1 revision

# Backend API Design

This document outlines the design principles and structure of the backend API for the AI-Driven Testing project. The API is responsible for receiving code snippets, interacting with Large Language Models (LLMs) to generate tests, and returning these tests to the client.

*(Note: Ensure file names like `main.py` or `api.py` match your actual project structure. If Pydantic models for request/response bodies are defined in a separate file like `api_models.py`, please mention that.)*

## 1. Framework and Core File

- **Framework:** The backend API is built using FastAPI, a modern, high-performance Python web framework chosen for its speed, ease of use, automatic data validation, and API documentation generation.
- **Core File:** The main FastAPI application instance, along with endpoint definitions, request/response models (using Pydantic), and startup logic, is primarily located in `backend/main.py`. *(If your main API logic is in `backend/api.py` or another file, please adjust this accordingly).*

## 2. Key Endpoints

The API exposes several endpoints to facilitate test generation and provide information:

- `GET /` (Root/Health Check):
  - **Purpose:** Provides a basic health check endpoint to verify that the API server is running and responsive. It also serves as a simple welcome point.
  - **Design:** Returns a static JSON message indicating the API is operational.
- `GET /models`:

  - **Purpose:** Allows clients (like the CLI or a web frontend) to discover which LLMs are currently supported and configured for use by the backend.
  - **Design:** This endpoint reads the list of model identifiers from the `allowed_models.json` file (see Model Configuration Design) at startup and returns this list as a JSON array of strings. This enables client applications to dynamically adapt to the available models.

- `POST /generate-test`:

  - **Purpose:** This is the primary functional endpoint of the API. It accepts a Python code snippet, the name of the desired LLM, and other optional parameters. It then interacts with the chosen LLM to generate a unit test case and streams the result back to the client.
  - **Design Details:**
    - **Request Handling:** Accepts a JSON payload in the request body. This payload is validated against a Pydantic model (`RequestModel`) to ensure all required fields are present and correctly formatted.
    - **LLM Client Initialization:** Delegates the creation and configuration of the appropriate LLM client (e.g., OpenAI, Anthropic, Ollama) to the `llm_manager.py` module (see LLM Manager Design). This promotes separation of concerns.
    - **Prompt Construction:** Dynamically constructs a detailed prompt to send to the LLM. This prompt includes the user's code snippet, any provided dependencies, user preferences for testing frameworks (e.g., pytest, unittest), and a system message guiding the LLM's behavior (see Prompt Design).
    - **Streaming Response:** Initiates a streaming request to the selected LLM. The API then returns a `StreamingResponse` (media type `text/plain`) to the client. This means the client starts receiving the generated test case token by token as the LLM produces it, which is crucial for good user experience with potentially long generation times.

## 3. Data Models (Pydantic)

FastAPI leverages Pydantic models for:

- **Data Validation:** Ensuring incoming request data conforms to the expected schema.
- **Data Serialization:** Converting data to and from JSON.
- **API Documentation:** Automatically generating JSON Schema definitions used in the Swagger UI and ReDoc documentation.

Key Pydantic models (likely defined in `main.py` or a separate `api_models.py` file) include:

- **`RequestModel` (or a similar name for the `/generate-test` input):**

  - Defines the structure of the JSON request body for the `/generate-test` endpoint.
  - Specifies required fields such as `code_snippet` (string) and `model_name` (string).

- Includes optional fields like `dependencies` (list of strings), `testing_framework` (string), `mocking_framework` (string), and `llm_config` (an `LLMConfig` object).
- Provides type hints and validation rules (e.g., `model_name` must be one of the models returned by `GET /models`).

- **`LLMConfig` (nested within `RequestModel`):**
  - An optional Pydantic model allowing clients to provide specific LLM provider settings for a single request, overriding any globally configured defaults.
  - Fields include `provider` (string, e.g., "openai", "anthropic", "ollama"), `api_key` (string), and `base_url` (string).

- **`ErrorResponseModel` (Implicit or Explicit):**
  - While FastAPI often handles this implicitly when `HTTPException` is raised, a standardized error response model typically includes a `detail` field containing a human-readable error message.

## 4. Error Handling

- The API uses FastAPI's built-in `HTTPException` to return appropriate HTTP status codes (e.g., 400, 404, 500) and JSON error messages to the client.
- **Common error scenarios and their typical responses:**
  - Invalid request payload (e.g., missing required fields, incorrect data types): `400 Bad Request` or `422 Unprocessable Entity`.
  - Requested `model_name` not found in `allowed_models.json`: `400 Bad Request`.
  - Missing or invalid API key for an LLM provider when required: Typically handled by `llm_manager.py`, which might raise a `400 Bad Request`.
  - Unexpected errors during LLM interaction or other internal processing: `500 Internal Server Error`.

## 5. Asynchronous Operations

- Given that interactions with external LLM APIs can be time-consuming (I/O-bound), the API is designed to be asynchronous using Python's `async` and `await` keywords.
- Specifically, the `/generate-test` endpoint and its underlying calls to LLM SDKs are implemented asynchronously. This allows the server to handle other requests concurrently while waiting for responses from the LLMs, improving overall throughput and responsiveness.

+ Add a custom footer

+ Add a custom sidebar

## Clone this wiki locally

```
https://github.com/amosproj/amos2025ss04-ai-driven-testing.wiki.git
```