**GenAI-Powered API Mock Generator — Project Documentation**

**Overview**

This project is a Streamlit-based web app that enables users to:

* Upload Swagger/OpenAPI specification files (YAML/JSON).
* Automatically generate realistic mock responses for API endpoints based on the spec schemas.
* Export generated mocks as Postman collections.
* Enhance Swagger specs by injecting rich fake example data into request and response schemas.
* Run a local Flask-based mock server to serve these mocked endpoints with optional validation and configurable response delay.

The solution uses AI-powered fake data generation heuristics to produce realistic sample data, enhancing API development, testing, and demos.

**File-by-File Breakdown**

**1. app.py**

**Purpose**

Main Streamlit app providing the user interface for uploading specs, generating mocks, exporting collections, enhancing Swagger files, and starting the mock server.

**Key Features & UI**

* **Custom CSS Styling**: Polished UI with a professional color palette, wide tabs bar to prevent wrapping, single-line header, styled buttons, and YAML editor fonts.
* **Tabs**:
  + Upload Swagger File
  + Generate Mock Response
  + Export to Postman
  + Swagger-to-Swagger (Enhanced Spec)
  + Run Mock Server

**Main Components & Functions**

* **Upload Swagger File**:  
  File uploader supporting JSON/YAML, loads and parses Swagger spec using load\_swagger from swagger\_parser.py, extracts endpoints, and stores state.
* **Generate Mock Response**:  
  User selects an endpoint from extracted list, app gets JSON schema for response via get\_response\_schema, generates mock data with generate\_mock\_response from mock\_generator.py, displays JSON, and allows download.
* **Export to Postman**:  
  Builds a minimal Postman collection JSON based on selected endpoint and mock response, downloadable as .json.
* **Swagger-to-Swagger (Enhanced Spec)**:  
  Calls enhance\_swagger\_with\_samples from swagger\_enhancer.py to inject realistic fake examples into request/response schemas and shows enhanced spec YAML with editable text area and download option.
* **Run Mock Server**:  
  Lets user configure port, response delay, and strict validation mode; starts Flask mock server (run\_server.py) in background subprocess and optionally opens endpoint URL in browser.

**2. swagger\_parser.py**

**Purpose**

Utility module for parsing Swagger/OpenAPI specs and extracting useful info for the app.

**Functions**

* load\_swagger(file, filetype=None)  
  Loads and parses a Swagger file object in JSON or YAML format into a Python dict.
* extract\_endpoints(spec)  
  Returns a list of all API endpoints with HTTP methods as strings like "GET /users" from the spec.
* get\_response\_schema(spec, path, method)  
  Retrieves JSON schema for the response of a given API endpoint, falling back to request body schema if response schema is unavailable.

**3. mock\_generator.py**

**Purpose**

Contains logic to generate realistic fake data samples matching a given JSON schema, for use as mock responses or example data.

**Key Functions**

* smart\_mock(field\_name, data\_type, format\_type=None, enum\_list=None)  
  Generates a single fake value based on field name heuristics, data type, format (like email, date), and enum lists, using Faker and custom logic.
* generate\_mock\_response(schema)  
  Recursively generates mock data conforming to the input JSON schema. Supports:
  + Primitive types (string, integer, boolean, number)
  + Arrays (returns list of two sample items)
  + Objects with properties
  + Schema composition keywords: allOf, oneOf, anyOf

**4. swagger\_enhancer.py**

**Purpose**

Enhances a Swagger/OpenAPI spec by injecting realistic fake example data (example fields) into requestBody and response schemas using the mock generator.

**Functions**

* enhance\_swagger\_with\_samples(spec: dict) -> dict  
  Makes a deep copy of the spec and walks all API paths and methods, injecting example fields in relevant schemas.
* \_generate\_example\_for\_schema(schema: dict)  
  Helper that calls generate\_mock\_response to produce a sample instance for a JSON schema.

**5. run\_server.py**

**Purpose**

A Flask-based lightweight mock server that:

* Serves endpoints as defined in the Swagger spec, returning mock responses generated from the schemas.
* Supports configurable response delay for simulating latency.
* Optional strict validation of query parameters, headers, and request body using JSON Schema validation (jsonschema library).
* Auto-registers all endpoints in the spec with appropriate HTTP methods and paths.

**Key Details**

* **Argument parsing**: Takes Swagger file path, port, delay, and --strict flag from CLI.
* **Schema resolution**: Resolves $ref references from components/schemas for requests and responses.
* **Validation**: Checks presence of required query/header params, enum constraints, and validates JSON request bodies.
* **Dynamic route registration**: Adds routes programmatically based on Swagger paths and methods.

**Features Summary**

* **Swagger File Support**: Upload JSON/YAML OpenAPI/Swagger 3.0+ specs.
* **Dynamic Endpoint Extraction**: Extracts all defined API endpoints and methods for user selection.
* **Mock Response Generation**: Auto-generates realistic fake data from schemas, including nested and composite types.
* **Swagger Enhancement**: Injects example fields with realistic fake samples into spec for better documentation and mocking.
* **Postman Export**: Creates Postman collection JSON from generated mocks.
* **Configurable Mock Server**: Runs a local Flask mock server with configurable port, response delay, and strict validation modes.
* **Professional Streamlit UI**: Clean, responsive tabs, polished controls, editable enhanced specs, and download options.

**How It Works: Flow Summary**

1. User uploads a Swagger spec file.
2. The app parses and lists available endpoints.
3. User selects an endpoint to generate a mock response based on its JSON schema.
4. User can export the mock response as JSON or a Postman collection.
5. User can generate an enhanced Swagger spec with fake example fields injected.
6. User can run a local mock server that serves these mocked responses with optional validation and delay.

**How To Run**

* Install required packages (Streamlit, Flask, PyYAML, Faker, jsonschema, etc.)
* Run streamlit run app.py to start the web UI.
* Use the UI to upload specs, generate mocks, and start mock server.
* Mock server will start as a subprocess running run\_server.py with given settings.

**Tips for Code Walkthrough**

* Explain the modular separation: UI (app.py), parsing (swagger\_parser.py), mock data gen (mock\_generator.py), Swagger enhancement (swagger\_enhancer.py), and mock server (run\_server.py).
* Emphasize the recursive and heuristic logic in mock generation to produce realistic data.
* Show how the mock server dynamically registers endpoints and validates input based on the spec.
* Highlight UI polish for smooth user experience and flexibility.
* Be ready to demo uploading a spec, generating a mock response, exporting to Postman, and running the mock server.