Code Analysis Endpoint ? Flask Application

Table of Contents

- --

Overview

A single Flask route (`/code-analyze`) accepts a code file upload, infers its language from the extension, builds a minimal context, sends it to an LLM endpoint for analysis, and renders the result on the main page. It is designed as a quick demo for single?file code analysis.

- --

Architecture & Flow

```
Client (POST /code-analyze)
        ?
        ?
Flask Route ? Validate file & form data
        ?
        ?
Infer language ? Build context dict
        ?
        ?
Generate LLM messages via build_code_assistant_messages()
        ?
        ?
POST to LM Studio chat/completions endpoint
```

?
?
Parse LLM response ? Render index.html with result
```

Key components:
- **Flask** ? web framework.
- **W

- --

## Key Functions & Helpers

| Function | Purpose |
|---|---|
| `code_analyze()` | Main route handling file upload, context creation, LLM call, and rendering. |
| `build_code_assistant_messages(context, goal)` | (External) Builds the conversation payload for the LLM. |
| `secure_filename()` | Sanitizes uploaded file names. |

- *Constants (assumed global)**
- `LM

- --

## Request Handling

- Accepts **POST** with `multipart/form-data`.
- Exp

  - Optional form fields: `"code_goal"` and `"tech_stack"`.
- Validates presence of a file; returns **400** if missing.

- --

## Language Detection

```python
ext = filename.rsplit(".", 1)[-1].lower() if "." in filename else ""

```
language_map = { ... }   # mapping of extensions to language names
language = language_map.get(ext, "text")
```

- Very naive: relies solely on file extension.

- Fall

- --

Context Construction

```python
context = {
    "project_name": "Ad-hoc upload project",
    "project_tech_stack": tech_stack,
    "project_notes": "Single-file upload via Recall AI demo.",
    "project_summary": "...",
    "active_file": {
        "path": filename,
        "language": language,
        "content": code_content,
    },
    "selection_snippet": "",
    "related_files": [],
}
```

- Minimal, single?file context.

- Des

- --

LLM Interaction

```python
messages = build_code_assistant_messages(context, user_goal or "Explain this code file in detail.")
payload = {"model": CHAT_MODEL, "messages": messages, "temperature": 0.2}
resp = requests.post(f"{LM_STUDIO_BASE_URL}/chat/completions", json=payload)
```

- Uses a low temperature for deterministic output.

- Exp

- --

Response Rendering

```python
return render_template(
    "index.html",
    chunks_count=len(DOCUMENT_CHUNKS),
    code_answer=code_answer,
    code_goal=user_goal,
)
```

- Passes the LLM answer, goal, and chunk count to the template.

- No

- --

Pros & Cons

Pros

- **Simplicity** ? Easy to understand and extend.

- **R

Cons

- **Naive language detection** ? Extension?only approach can misclassify.

- **N

- --

Suggested Improvements

Area	Recommendation
Language Detection	Use a library like `guesslang` or parse file headers for more accurate inference.
File Validation	Enforce size limits, MIME type checks, and optional syntax validation.
Multi?file Support	Accept a ZIP archive or repository clone; build a full project context.

Context Enrichment	Pull metadata from `pyproject.toml`, `package.json`, or other config files.
Error Handling	Catch exceptions from `requests` and provide user?friendly messages.
Caching	Cache LLM responses for identical inputs to reduce latency and cost.
Rate Limiting	Protect the endpoint from abuse (e.g., Flask?Limiter).
Testing	Add unit tests for language detection, context building, and LLM payload creation.
Documentation	Generate API docs (e.g., Swagger/OpenAPI) for the route.

- --

Overall Review

The `code_analyze` endpoint is a solid foundation for a single?file code analysis demo. It cleanly separates concerns: file handling, context creation, LLM interaction, and rendering. However, to move from a prototype to production?grade service, it needs stronger validation, richer context handling, and robust error management. Implementing the suggested improvements will make it more reliable, secure, and extensible for real?world use cases.