📁 Architecture_documentation_generator/app.py

- Main (): This is the entry point of the Streamlit application.
    - **Configuration Setup**: Sets up the Streamlit page with title, icon, and wide layout
    - **Sidebar Configuration**:
        - Creates input fields for Gemini API key and GitHub token
        - The Gemini API key has a default value (which should be removed in production)
        - GitHub token is optional for higher rate limits
    - **Main Interface:**
        - Two-column layout: main analysis area and examples sidebar
        - Input field for GitHub repository URL
        - Checkboxes for analysis options:
        - Include Mermaid Diagrams
        - Analyze Design Patterns
        - Analyze Dependencies
        - Include Data Flow Analysis
    - **Analysis Process:** When "Analyze Repository" is clicked:
    - **Validation**: Checks if URL and API key are provided
    - **Client Initialization**: Sets up GitHub and Gemini API clients
    - **Repository Validation**: Verifies the repository exists and is accessible
    - **File Fetching**: Downloads all relevant repository files
- **AI Analysis**: Uses Gemini to analyze the codebase
- **Results Storage**: Saves results in Streamlit session state
- **Results Display:** Creates tabbed interface showing:
- Overview: Repository statistics and summary
- Architecture: High-level design and diagrams
- Modules: Component breakdown
- Data Flow: How data moves through the system
- Design Patterns: Identified patterns
- Export: PDF/DOCX download options

📁 Architecture_documentation_generator/github_analyzer_functions.py

- **Global Variables:** Stores initialized clients for reuse across functions
    - gemini_client = None
    - github_session = None
    - github_headers = {}
- **initialize_clients**(gemini_api_key, github_token)
    - Sets up API clients for both GitHub and Gemini services.
- **File Classification Constants:** Defines which files to analyze, prioritize, or skip

-
    - CODE_EXTENSIONS = {'.py', '.js', '.ts', ...}
    - PRIORITY_FILES = {'package.json', 'requirements.txt', ...}
    - SKIP_PATTERNS = {'test', 'node_modules', '.git', ...}
- **should_skip_path(path)**
    - Determines if a file/directory should be ignored:
    - Checks against skip patterns (test directories, build artifacts, etc.)
    - Skips binary files (images, executables, etc.)
- **validate_repository(github_url)**: Validates GitHub URL and fetches repository metadata:
    - Uses regex to extract owner/repo from URL
    - Makes GitHub API call to get repository info
    - Returns dictionary with repo details (name, description, language, stars, etc.)
- **get_file_type(filename):** Maps file extensions to readable file types (e.g., '.py' → 'Python')
- **fetch_repository_contents(owner, repo_name, path)**
    - Makes GitHub API calls to get directory/file contents for a specific path.
- **fetch_all_repository_files(github_url):** Core Function. This is the main repository scanning function:
    - **Initialization**: Sets up data structure to store results
    - **Breadth-First Traversal**: Uses queue to explore all directories
    - **Path Filtering**: Skips test directories, build files, etc.
    - **File Processing**: For each file:
        - Categorizes by type
        - Counts statistics
        - Downloads content for analysis (if relevant and under 100KB)
    - **Content Fetching**: Downloads actual file contents for important files
    - **Returns**: Dictionary containing:
        - files: List of all files with metadata
        - directories: List of all directories
        - key_files: Dictionary of file paths → content
        - statistics: File counts, languages, etc.
- **build_analysis_prompt(repo_structure, options)**
    - **Creates the prompt sent to Gemini AI:**
    - **Repository Summary**: File counts, directories
    - **Key File Contents**: Up to 25 most important files (3000 chars each)
    - **Analysis Requirements**: Based on user options
    - **Output Format**: Structured sections for consistent parsing
    - **File Prioritization**:
    - Priority files (package.json, README, etc.) come first
    - Then sorted by file size

    - Truncates large files to first 15000 characters
- **parse_gemini_response(response_text)**
    - Parses AI response into structured sections using regex:
    - Overview
    - Architecture
    - Modules
    - Data Flow
    - Design Patterns
    - Mermaid Diagrams
- **analyze_repository_with_ai(repo_structure, options):** Main AI analysis function
    - Builds comprehensive prompt
    - Calls Gemini AI with specific configuration
    - Parses response into structured format
    - Adds repository statistics

📁 Architecture_documentation_generator /document_generator_functions.py

- **setup_pdf_styles()**
    - Creates custom PDF styles using ReportLab:
    - Title style (large, centered, green)
    - Heading styles (different sizes, colors)
- Code style (monospace, gray background)
- **generate_pdf_document (documentation, repo_info**): Creates professional PDF documentation:
    - **Title Page**: Repository name, description, generation date
    - **Statistics Table**: File counts, languages, GitHub stats
    - **Content Sections**: Overview, architecture, modules, data flow, design patterns
    - **Diagrams**: Mermaid code blocks for diagrams
    - **PDF Features**:
        - Professional styling with colors
        - Tables for statistics
        - Code blocks for diagrams
        - Proper spacing and formatting
- **generate_docx_document(documentation, repo_info)**
    - **Creates Word document with similar content:**
    - Uses python-docx library
    - Table for statistics
    - Structured headings
    - Code blocks styled as quotes

How the System Works End-to-End:

1. **User Input**: User provides GitHub URL and selects analysis options

2. **Repository Scanning**:
    - Validates repository exists
    - Downloads all file/directory information via GitHub API
    - Filters out irrelevant files (tests, builds, binaries)
    - Downloads content for important files (up to 25 files, 3000 chars each)
3. **AI Analysis**:
    - Builds comprehensive prompt with file contents and structure
    - Sends to Gemini AI with specific formatting requirements
    - AI analyses code patterns, architecture, data flow, design patterns
4. **Result Processing**:
    - Parses AI response into structured sections
    - Adds repository statistics
    - Stores in session state for display
5. **Documentation Display**:
    - Shows results in tabbed interface
    - Displays Mermaid diagrams
    - Provides export options
6. **Export**:
    - Generates professional PDF or Word documents
    - Includes all analysis sections, statistics, and diagrams

Problem: Take Time:
- Solution 1: Can be used RAG Based Approach
- Solution 2: Try to summarize the code then pass to LLM to generating Architecture and design documentation.

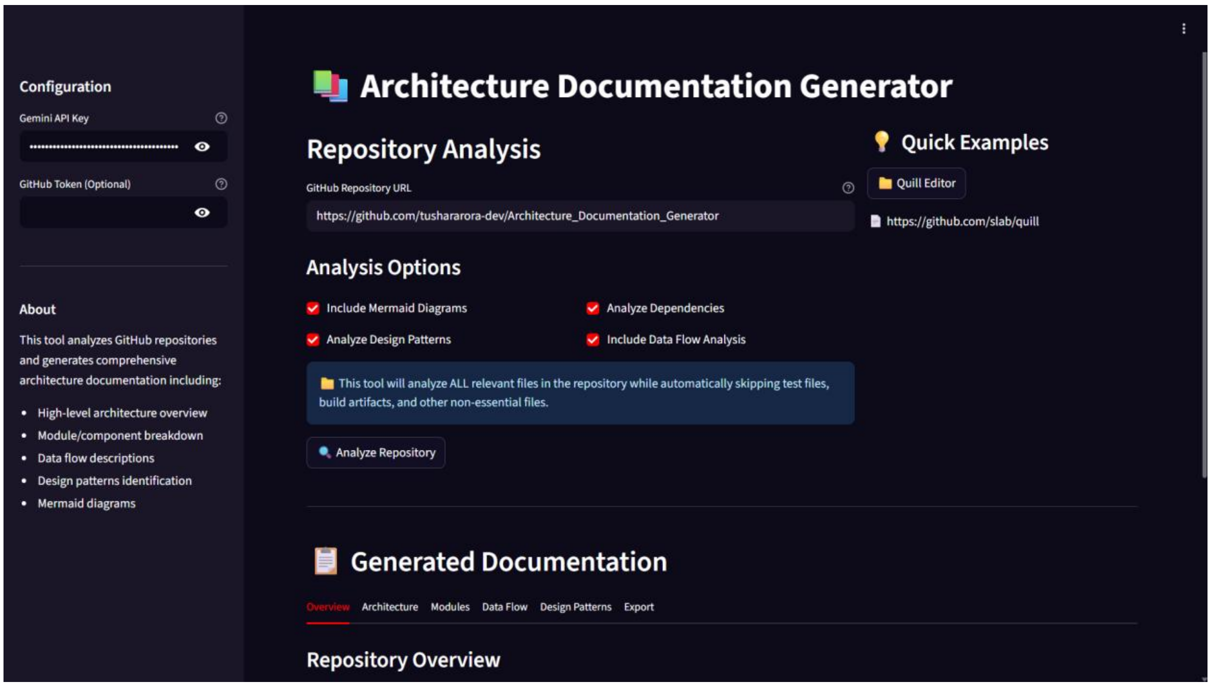
Due to time and future cost cutting, I used top 25 files with first 3000 characters and less than 100kb size file.
- **1 token ≈ 4 characters** in English text (including spaces & punctuation).
- So, **3000 characters ÷ 4 ≈ 750 tokens**.
- 25×3000=75,000 characters then 75,000÷4=18,750 tokens

Let take an example of this project (Take 3 Main Files):
app.py: Total characters: 10263
document_generator_functions.py: Total characters: 10763
github_analyzer_functions.py: Total characters: 16380
10263 + 10763 + 16380 = 37,406 characters
37,406÷4=9,351.5

I change 3000 to 15000 Character. Will change approach like to fetch starting, middle and end.

Readme.md of Architecture_documentation_generator/app.py:
https://github.com/tushararora-dev/Architecture_Documentation_Generator/blob/main/readme.md

GitHub URL:
https://github.com/tushararora-dev/Architecture_Documentation_Generator



Configuration

Gemini API Key: ••••••••••••••••••••••••••••••••••••

GitHub Token (Optional):

About

This tool analyzes GitHub repositories and generates comprehensive architecture documentation including:

- High-level architecture overview
- Module/component breakdown
- Data flow descriptions
- Design patterns identification
- Mermaid diagrams

Architecture Documentation Generator

Repository Analysis

GitHub Repository URL: https://github.com/tushararora-dev/Architecture_Documentation_Generator

Quick Examples

Quill Editor

https://github.com/slab/quill

Analysis Options

- ✓ Include Mermaid Diagrams
- ✓ Analyze Dependencies
- ✓ Analyze Design Patterns
- ✓ Include Data Flow Analysis

This tool will analyze ALL relevant files in the repository while automatically skipping test files, build artifacts, and other non-essential files.

Analyze Repository

Generated Documentation

Overview | Architecture | Modules | Data Flow | Design Patterns | Export

Repository Overview