UNDERSTANDING THE CODEBASE GENIUS 📚

A complete Guide to Building code base genius.

#I AM NOT PERFECT ANY MISTAKE IN THE DOCUMENT FEEL FREE TO AIR IT OUT AND IT SHALL BE CORRECTED

TABLE OF CONTENTS

Introduction

1.Prerequisites & Installation

2.Understanding the Architecture

3.Setting Up Your Development Environment

4.Backend Development with Jac

5.Frontend Development with Streamlit

6.Running the Application

*Best Practices

INTRODUCTION

This documentation provides a fully working backend built with JacLang and Streamlit frontend. You will learn how to set up and run the sample project, then apply similar patterns and best practices when implementing your own code documentation pipeline using the byLLM (Large Language Model) task manager.

Project Overview:

Frontend: Streamlit (User Interface)

Backend: JacLang + OpenAI (AI Logic/Brain)

Purpose: Task Manager with AI-powered documentation

PREREQUISITES & INSTALLATION 💻

WINDOWS Installation

Install JacLang

pip install jaclang

Install Python 3

pip install python3

Checking Versions:

jac --version

python3 --version

UBUNTU/LINUX Installation

Installation Syntax:

sudo apt update && sudo apt upgrade -y

Install Python3:

sudo apt install python3 -y

Checking Versions:

python3 --version

OR if you installed python directly:

python --version

⚠️ IMPORTANT NOTE:

After python or python3, there is a space then --version (with two dashes)

UNDERSTANDING THE ARCHITECTURE 🏗️

Project Structure

CodebaseGenius/

├── frontend/

│ └── app.py (Streamlit UI)

├── backend/

│ ├── RepoMapper.jac

│ ├── CodeAnalyzer.jac

│ └── DocGenie.jac

├── requirements.txt

├── .env

└── README.md

Components Breakdown:

FRONTEND (40%) - What the user interacts with

Stack: Streamlit

Purpose: User interface, input forms, display results

BACKEND (60%) - Brain of the project (AI Logic)

Stack: JacLang & OpenAI

Purpose: Process requests, analyze code, generate documentation

Configuration Files:

requirements.txt - Lists all Python dependencies

.env - Stores environment variables (API keys)

README.md - Project documentation

SETTING UP YOUR DEVELOPMENT ENVIRONMENT 🛠️

Required Tools:

VS Code (Integrated Development Environment) - For editing files

Ubuntu/WSL (Linux Terminal) - For running commands

💡 POINT OF CLARITY:

VS Code manages your external files (file explorer, editing)

Ubuntu/WSL is the terminal where you write and execute commands

Step 1: Open Ubuntu Terminal

You should see:

user@admin:~$

Understanding the prompt:

user - Your username

admin - Your hostname (computer name)

~ - Current directory (home directory, like /home/user)

$ - Indicates you're a regular user (not root/administrator)

Step 2: Link WSL and VS Code

Install WSL (if not already installed)

wsl --install

Navigate to your desired location and open VS Code

code .

In VS Code:

Press Ctrl + Shift + P (Opens command palette)

Type: >Remote-WSL: New Window

Or type: >Terminal: Select Default Profile and set WSL as default

Step 3: Create Project Directory

Check current directory

pwd

List files in current directory

ls -la

Create project directory

mkdir CodebaseGenius

Navigate into the directory

cd CodebaseGenius

Verify you're in the correct location

pwd

Expected output:

/home/user/CodebaseGenius

BACKEND DEVELOPMENT WITH JAC 🧠

Creating Backend Files

From VS Code or terminal:

Create backend folder

mkdir backend

cd backend

Create the three main Jac files

touch RepoMapper.jac

touch CodeAnalyzer.jac

touch DocGenie.jac

⚠️ REMEMBER: Capitalization matters! RepoMapper.jac ≠ repomapper.jac

1. RepoMapper.jac 🗺️

Purpose: Maps the repository structure and identifies files

```
walker RepoMapper {
    has repo_path: str;
    has file_list: list = [];

    can map_repository with entry {
        # Logic to traverse directory structure
        # Identifies all code files
        # Returns structured file tree
        print("Mapping repository structure...");
    }
}
```

What it does:

Scans your project directory

Creates a hierarchical map of all files

Identifies file types (Python, JavaScript, etc.)

Returns a structured list for analysis

2. CodeAnalyzer.jac 🔍

Purpose: Analyzes code quality, complexity, and patterns

```
walker CodeAnalyzer {
    has file_path: str;
    has analysis_results: dict = {};

    can analyze_code with entry {
        # Reads file content
```

```
        # Performs static code analysis
        # Identifies functions, classes, imports
        # Calculates complexity metrics
        print("Analyzing code structure...");
    }
}
```

What it does:

Reads source code files

Identifies functions, classes, and variables

Analyzes code complexity

Detects patterns and potential issues

Generates metrics (lines of code, cyclomatic complexity)

3. DocGenie.jac 📝

Purpose: Generates AI-powered documentation using OpenAI

```
import:py from openai { OpenAI }

walker DocGenie {
    has code_content: str;
    has api_key: str;
    has documentation: str = "";

    can generate_docs with entry {
        # Connects to OpenAI API
        # Sends code for analysis
        # Receives AI-generated documentation
        # Formats and returns docs
        print("Generating documentation with AI...");
    }
}
```

What it does:

Takes analyzed code as input

Uses OpenAI API to generate human-readable documentation

Creates descriptions for functions, classes, and modules

Formats documentation in Markdown

Returns structured documentation

CREATING PYTHON VIRTUAL ENVIRONMENT

Why Use a Virtual Environment?

Isolates project dependencies

Prevents version conflicts

Makes project portable

Easy to recreate on other machines

Step-by-Step Setup:

```
# Navigate to project root
cd ~/CodebaseGenius

# Create virtual environment
python3 -m venv venv

# Activate virtual environment (Linux/WSL)
source venv/bin/activate

# Your prompt should now show (venv)
(venv) user@admin:~/CodebaseGenius$

# Install dependencies
pip install jaclang streamlit openai python-dotenv

# Create requirements.txt
```

pip freeze > requirements.txt

To deactivate:

deactivate

SETTING UP OPENAI API KEY 🔑

Step 1: Get Your API Key

Visit: https://platform.openai.com

Sign up or log in

Navigate to: API Keys section

Click: Create new secret key

Copy the key immediately (you won't see it again!)

Step 2: Create .env File

In project root directory

touch .env

Edit .env file (using nano or VS Code):

nano .env

Add your API key:

OPENAI_API_KEY=sk-proj-your-actual-api-key-here

Save and exit (Ctrl + X, then Y, then Enter in nano)

Step 3: Load Environment Variables in Code

In your Python/Jac files:

import:py from dotenv { load_dotenv }

import:py import os

Load environment variables

load_dotenv();

Access API key

api_key = os.getenv("OPENAI_API_KEY");

FRONTEND DEVELOPMENT WITH STREAMLIT 🎨

Create Frontend File

Create frontend folder

mkdir frontend

cd frontend

Create Streamlit app

touch app.py

Basic Streamlit Structure (app.py)

```
import streamlit as st
import requests

st.title("🤖 CodeBase Genius")
st.subheader("AI-Powered Code Documentation Generator")

# File uploader
uploaded_file = st.file_uploader("Upload your code file", type=['py', 'js', 'java'])

if uploaded_file:
    # Read file content
    content = uploaded_file.read().decode()

    # Display code
    st.code(content, language='python')

    # Generate documentation button
    if st.button("Generate Documentation"):
        with st.spinner("Analyzing code..."):
            # Call backend API
            response = requests.post(
```

```
            "http://localhost:8000/generate-docs",
            json={"code": content}
        )

        # Display results
        st.success("Documentation generated!")
        st.markdown(response.json()["documentation"])
```

RUNNING THE APPLICATION 🚀

Step 1: Start the Backend Server

```
# Make sure you're in the backend directory
cd ~/CodebaseGenius/backend

# Start Jac server
jac serve task_manager.jac
```

Expected output:

```
Starting Jac server on http://localhost:8000
Server is running...
```

Step 2: Start the Frontend

Open a new terminal tab/window:

```
# Navigate to frontend directory
cd ~/CodebaseGenius/frontend

# Activate virtual environment
source ../venv/bin/activate

# Run Streamlit app
streamlit run app.py
```

Expected output:

```
You can now view your Streamlit app in your browser.
```

Local URL: http://localhost:8501

Network URL: http://192.168.x.x:8501

Step 3: Access the Application

Open your browser and go to: http://localhost:8501

USEFUL TERMINAL COMMANDS 📋

Navigation Commands:

```
pwd              # Print working directory (shows current location)
ls               # List files in current directory
ls -la           # List all files (including hidden) with details
cd <directory>   # Change directory
cd ..            # Go up one directory level
cd ~             # Go to home directory
mkdir <name>     # Make directory
touch <filename> # Create empty file
rm <filename>    # Remove file
rm -r <directory> # Remove directory recursively
```

File Operations:

```
cat <filename>   # Display file contents
nano <filename>  # Edit file with nano editor
vim <filename>   # Edit file with vim editor
cp <source> <dest>  # Copy file
mv <source> <dest>  # Move/rename file
```

Process Management:

```
Ctrl + C         # Stop running process
Ctrl + Z         # Suspend process
ps aux           # List running processes
kill <PID>       # Kill process by ID
```

BEST PRACTICES ☑

1. Code Organization

Keep backend and frontend separate

Use clear, descriptive file names

Follow consistent naming conventions (PascalCase for classes, snake_case for variables)

2. Environment Management

Always use virtual environments

Keep .env file in .gitignore

Document all environment variables in README

3. API Security

Never commit API keys to Git

Use environment variables for sensitive data

Rotate keys regularly

4. Version Control

Initialize Git repository

git init

Create .gitignore

echo "venv/" >> .gitignore

echo ".env" >> .gitignore

echo "_pycache_/" >> .gitignore

Make first commit

git add .

git commit -m "Initial commit"

5. Documentation

Write clear README files

Comment complex code sections

Keep documentation up-to-date

Use emojis to make docs more engaging! 🎉

TROUBLESHOOTING 🔧

Common Issues:

1. "jac: command not found"

Reinstall jaclang

pip install --upgrade jaclang

Check if it's in PATH

which jac

2. "Port already in use"

Find process using port 8000

lsof -i :8000

Kill the process

kill -9 <PID>

3. "Module not found"

Make sure virtual environment is activated

source venv/bin/activate

Reinstall dependencies

pip install -r requirements.txt

4. "OpenAI API Error"

Verify API key is correct in .env

Check you have credits in your OpenAI account

Ensure .env file is in the correct directory

CREATING ATTRACTIVE README FILES 🎨

Using Emojis

Windows:

Press Windows + . (period) to open emoji picker

Or Windows + ; (semicolon)

Mac:

Press Cmd + Ctrl + Space

Linux/Ubuntu:

Press Ctrl + . or Ctrl + ;

Or install: sudo apt install ibus-table-emoji

README Template with Emojis:

🤖 CodeBase Genius

> AI-powered code documentation generator

✨ Features

- 🗺️ Repository mapping
- 🔍 Code analysis
- 📝 AI documentation generation
- 🎨 Beautiful UI with Streamlit

🚀 Quick Start

Installation

\\\`bash

pip install -r requirements.txt

\\\`

Usage

\\\`bash

jac serve backend/task_manager.jac

streamlit run frontend/app.py

\\\`

📚 Documentation

See [full documentation](docs/README.md)

🤝 Contributing

Contributions welcome! See CONTRIBUTING.md

📄 License

MIT License - see LICENSE

FINAL CHECKLIST ✔

Before considering your project complete:

[ ] All dependencies installed

[ ] Virtual environment created and activated

[ ] .env file created with API key

[ ] Backend files created (RepoMapper, CodeAnalyzer, DocGenie)

[ ] Frontend app.py created

[ ] Backend server starts without errors

[ ] Frontend loads in browser

[ ] API calls work correctly

[ ] Documentation is clear and complete

[ ] Code is committed to Git

[ ] .gitignore includes sensitive files

NEXT STEPS 🎯

Enhance Features:

Add support for more programming languages

Implement batch processing

Add code quality scoring

Improve UI:

Add dark mode

Create progress indicators

Add export options (PDF, HTML)

Deploy:

Deploy backend to cloud (Heroku, AWS, Azure)

Deploy frontend to Streamlit Cloud

Set up CI/CD pipeline

RESOURCES 📖

JacLang Documentation: https://docs.jac-lang.org

Streamlit Documentation: https://docs.streamlit.io

OpenAI API Reference: https://platform.openai.com/docs

Python Virtual Environments: https://docs.python.org/3/tutorial/venv.html

INSPIRATION 💪

During my journey of doing this project, I faced challenges and confusions. I decided to take my time and write this to help the few out there who are confused.

Remember: It's all about having the right mentality and consistency!

ACKNOWLEDGMENTS 🙏

If this documentation was helpful, you can send thanks to the writer:

Tel: +254717546421

Fun Fact: Did you know that emojis in documentation can increase engagement by up to 30%? Studies show that visual elements make technical docs more approachable! 📊

#GoodLuck fellow coursemates 🍀

Made with ❤️ by a fellow learner, for learners