Software Design Description (SDD)

Based on IEEE 1016 – Adapted for Student Projects

Project Title: Python code analyzer

Team Name & Members:

Zahra Bader Hayyan 2240006156

Hadeel abduallah alqhtani 2240003327

Noor Almuhsen 2240003505

Yomna Al moslem 2240003445

Alzahraa alabbad 2240006089

Eman Alnajem 2240002468

Wedad Mohammed AL-hussaini 2240002853

Advisor: Rahmah alzhrani

Version: 1.0

Date: 24-11-2025

Contents

1. Introduction

This Software Design Description (SDD) outlines the design of the Python Code Analyzer Tool, a system that processes user-uploaded Python (.py) files and generates a structured HTML report. Its purpose is to provide a detailed yet clear description of the system's architecture, internal components, and design decisions to ensure accurate and consistent implementation and testing.

The scope of the SDD covers the system's modules, data flow, processing logic, and interactions between components responsible for extracting code metrics such as line count, classes, functions, imports, and syntax-related issues.

The intended audience includes developers implementing the tool, testers validating functionality, and the course instructor reviewing the design for completeness and compliance.

This document is prepared in alignment with the Software Requirements Specification (SRS) and the Software Project Management Plan (SPMP), which define the functional requirements, constraints, and project structure. These documents serve as the basis for the design decisions presented here.

2. System Overview

2.1 High-Level Description

The Python Code Analyzer is a web-based static analysis system designed to evaluate the quality and structure of user-uploaded Python source files (.py). The system extracts essential code metrics including the number of lines, functions, classes, and imports and identifies syntax issues or formatting inconsistencies. The output is clear and readable HTML report that assists beginner programmers and developers in understanding, reviewing, and improving the quality of their code. A detailed Context Diagram showing the system's interaction with external entities is provided in Section 7,

2.2 Design Strategy

The system follows a plan-driven design strategy where the overall architecture is defined before implementation. The design process identifies main components, specifies their responsibilities, and defines interaction patterns. This approach emphasizes modularity, allowing each component such as file upload, static analysis, and report generation, to be developed and tested independently while maintaining

well-defined interfaces.

2.3 Architectural Style

The Python Code Analyzer adopts a web-based client–server architecture that separates the user interface from the backend processing logic.

- Frontend Interface: A browser-based interface responsible for handling user interactions, file uploads, and displaying the analysis report.

- Backend Server: It manages the core application workflow: receiving and validating uploaded files, performing static code analysis and syntax checks, handling temporary file storage, and generating the final HTML report.

Communication between the frontend and backend occurs through standard and a clear ,interoperability ,ensuring modularity ,HTTP/HTTPS protocols separation of concerns.

3. Design Considerations

3.1 Assumptions

- Users will access the tool through a supported web browser (Chrome, Firefox, Safari, Edge).
- Users will upload valid Python .py files through the website interface.
- Users have a stable internet connection while using the tool.
- The server should have Python 3.x installed and all required libraries available for analysis.
- Users have basic knowledge of Python and can understand the results shown in the report.

3.2 Constraints

- The server must run Python 3.x and use open-source libraries such as Ast, Inspect, Radon, Jinja2, etc.
- The tool should only analyze uploaded Python files and must not run or execute the code, to ensure safe and secure operation.
- File size for uploaded Python files may be limited to ensure performance and security.

- The system must run on standard web technologies such as HTML, CSS, and JavaScript for the user interface.
- The web application must be compatible with common browsers.
- The project must be completed within the timeline of the course.

3.3 Design Goals

- Provide a simple web interface where users can upload a Python file easily.
- Generate a clear HTML based analysis report that displays code structure, metrics, issues, and suggestions.
- Ensure the system is responsive and fast, even for medium sized Python files.
- Make the tool accessible without requiring users to install any software.
- The backend should be flexible and modular so new features can be added easily, like analyzing multiple files or uploading a ZIP project.
- Prioritize safety by enforcing static analysis to avoid running untrusted code.

3.4 Trade-offs

- The system performs static analysis only, which increases security by preventing the execution of untrusted python code. However, it limits the tool's ability to detect runtime errors or dynamic behavior.
- Making the tool web-based improves accessibility without requiring installation, but it also creates a dependency on internet connection and server availability.
- Generating reports directly in HTML is simple and browser friendly, but it offers less customization compared to a full graphical interface or downloadable PDF formats.
- Keeping the interface simple helps beginners and improves usability for students, but reduces the level of control and configuration options for advanced users.
- The project uses only open-source Python libraries to keep the tool free and easy to maintain, but this limits access to certain advanced features available in commercial code analysis tools

- Restricting uploaded file size improves performance but limits handling of very large projects.

4. Architectural Design

The Python Code Analyzer system follows a web-based client–server architecture. Users upload a Python file through the web interface, after which the backend processes the file, analyzes the code structure, detects syntax errors, and generates a structured HTML report.

Major System Component

Frontend Interface

The client-side interface provides an intuitive web-based environment through which users upload Python source files and later access the generated analytical reports. It handles all user interactions and ensures seamless communication with the backend services.

Upload Module

This module is responsible for receiving incoming .py files, performing format validation, and conducting preliminary integrity checks. It ensures only valid and secure inputs proceed to the analysis pipeline.

Code Analyzer

A core backend component that performs structural and static code analysis using Python's standard libraries (e.g., ast, inspect). It extracts essential metrics including line count, function and class definitions, imported modules, and additional syntactic constructs present in the source code.

Syntax Error Handler

This submodule isolates and detects syntactic anomalies by leveraging Python's compile() function within controlled exception-handling blocks. It ensures robust error detection without interrupting system execution or compromising stability.

Report Generator

A dedicated component that synthesizes the extracted analysis results into a structured, human-readable HTML report. It organizes metrics, syntax feedback, and code summaries into a coherent and visually accessible format.

Backend Server

The backend orchestrates the entire workflow, coordinating interactions between modules, managing temporary file storage, invoking analysis routines, and delivering the final report to the frontend. It enforces system logic, security constraints, and lifecycle management of uploaded files.

Component Interaction:

1. The user initiates the process by uploading a Python source file through the frontend interface.
2. The Upload Module validates the file format and integrity, ensuring that only legitimate .py files are forwarded to the analysis pipeline.
3. Upon successful validation, the Code Analyzer processes the file, extracting structural metrics and performing a detailed examination of functions, classes, imports, and other code elements.
4. Simultaneously, the Syntax Error Handler evaluates the file for syntactic correctness, capturing exceptions and recording relevant error diagnostics.
5. The Report Generator aggregates all collected information, formatting it into a comprehensive HTML report that encapsulates metrics, syntax results, and insights.
6. The Backend Server returns the finalized report to the frontend, where it is rendered and presented to the user for review.

Component diagram

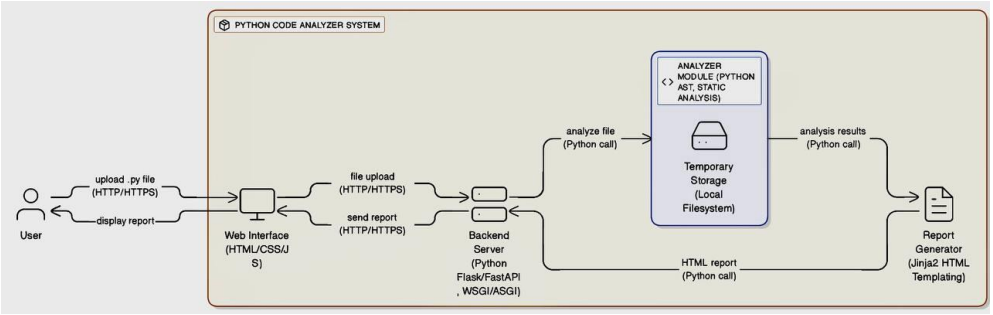

Figure 1 component Diagram

Deployment diagram

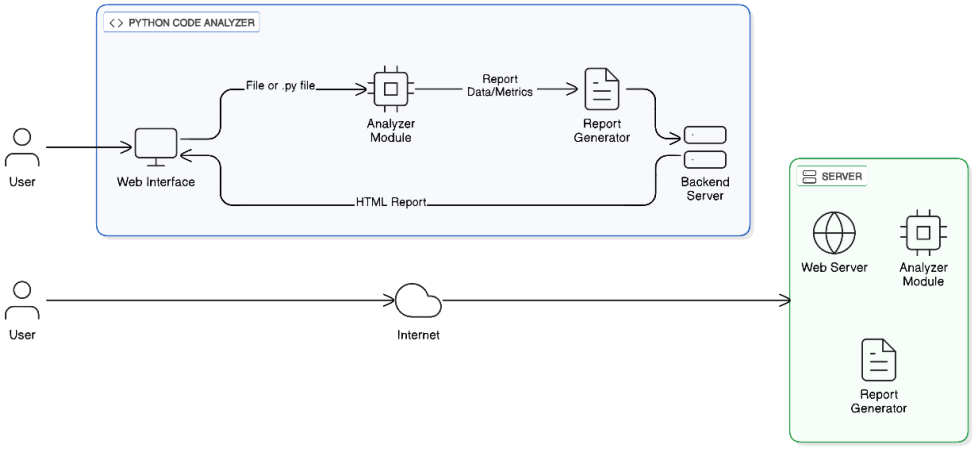

Figure 2 Deployment Diagram

5. Detailed Design

Module name	Responsibilities	Inputs/outputs	Notes / diagrams
File analysis module	1. Receive and read the uploaded Python file. 2. Extract number of lines. 3. Identify all functions and classes. 4. Detect imported libraries.	Inputs: Python file (.py) Outputs: line_count, functions_list, classes_list, imported_libraries	A sequence diagram can be added to illustrate file parsing.
Report generation module	1. Generate a structured HTML report. 2. Calculate complexity and maintainability metrics.	Inputs: analysis results from file analysis module Outputs: HTML report	A class diagram may be added to show report structure.

	3. Provide recommendations for code improvement.		
Error handling module	1. Detect syntax errors in the file. 2. Identify invalid file formats. 3. Ensure the system continues running without crashing.	Inputs: uploaded file Outputs: error messages, warnings, valid/invalid status	An activity diagram can illustrate the error flow.
User interaction module	1. Allow user to upload Python files. 2. Display the generated report. 3. Provide download functionality. 4. Allow re-analysis after modifications.	Inputs: user actions (upload, view, download, re-analyze) Outputs: HTML pages, notifications, downloadable report	A sequence diagram can represent user actions and system responses.

Table 1 table of modules

5.1 UML Diagrams

5.1.1 Sequence Diagram – File Analysis Module

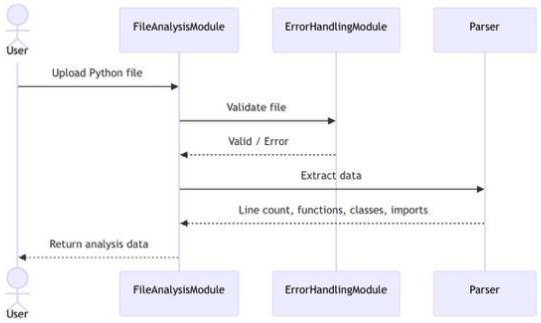

Figure 3 Sequence Diagram of File Analysis Module

5.1.2 Class Diagram – Report Generation

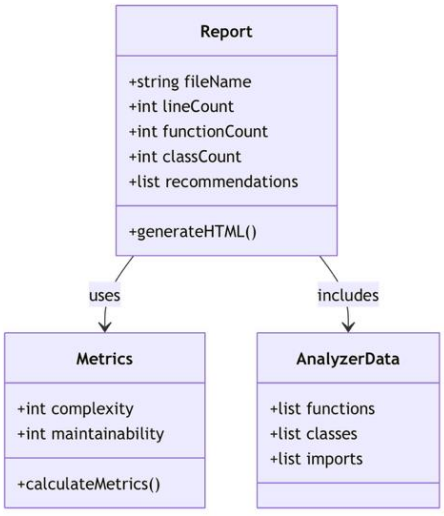

Figure 4 Class Diagram of Report Generation

5.1.3 Activity Diagram – Error Handling

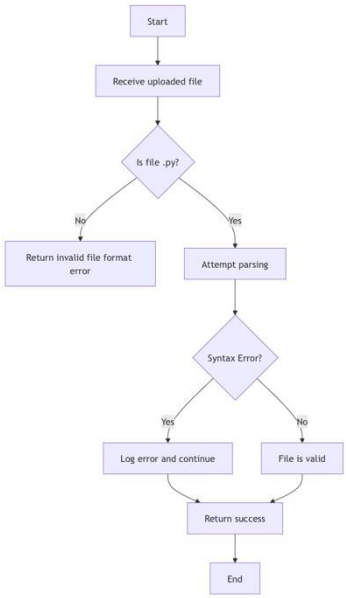

Figure 5 Activity Diagram for Error Handling

5.1.4 Sequence Diagram – User Interaction

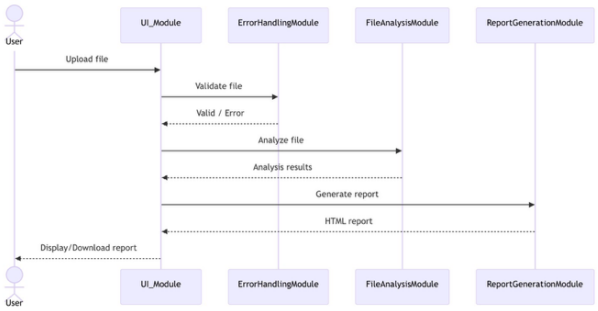

Figure 6 Sequence Diagram of the User Interaction

6. Data Design

The data design for the Python Code Analyzer is fundamentally driven by the need to
efficiently store, manage, and retrieve the results of the static code analysis. Given the
project's scope which focuses on extracting metrics and identifying structural
elements and simple errors the data architecture must support both the transient, in memory processing of the source code and the persistent storage of the final analysis
reports.

6.1 Data Structures for In Memory Analysis

the system processes the Abstract Syntax Tree (AST) of the Python source file. The
intermediate, in-memory data structures are designed to mirror the hierarchical nature
of the code, facilitating easy extraction and aggregation of metrics. A primary nested
dictionary structure is employed to hold the analysis results before persistence.

- Analysis Root Object: This top-level structure contains metadata about the
analysis run, such as the file name, analysis timestamp, and a high-level summary.
- File Metrics Object: A dedicated structure to store quantitative data about the
file, including total_lines, code_lines, comment_lines, and file_size.
- Constructs List: This list aggregates all identified structural elements. Each
element is an object representing a class, function, or imported module.
- Class Object: Contains class_name, start_line, end_line, and a nested list of its
methods.
- Function Object: Contains function_name, parameters, cyclomatic_complexity
(if calculated), and line_count.
- Issues List: A collection of objects detailing detected errors or style violations.

Each issue object includes issue_type (e.g., Syntax Error), line_number, and a

detailed description.

6.2 Database Schema Definition

The schema is composed of four primary entities, linked by one-to-many relationships originating from the central AnalysisReport entity.

AnalysisReport Entity

it is the core entity, representing a single execution of the code analyzer on a specific file. It serves as the parent record for all subsequent detailed data.

Attribute	Data Type	Description	Constraints
ReportID	INTEGER	Unique identifier for the analysis report.	Primary Key, Auto-Increment
FileName	VARCHAR(255)	The name of the analyzed Python file.	Not Null
AnalysisTimestamp	DATETIME	The date and time the analysis was performed.	Not Null
ProjectName	VARCHAR(255)	Optional field to group reports by project.	Nullable

Table 2 Analysis Report

FileMetrics Entity

it stores the quantitative, file-level statistics, maintaining a one-to-one relationship with the AnalysisReport.

Attribute	Data Type	Description	Constraints
MetricID	INTEGER	Unique identifier for the metric record.	Primary key, Auto-Increment

ReportID	INTEGER	Foreign key linking to the parent report.	Foreign key (AnalysisReport)
TotalLines	INTEGER	Total number of lines in the file.	Not Null
CodeLines	INTEGER	Number of lines containing executable code.	Not Null
CommentLines	INTEGER	Number of lines containing only comments.	Not Null

Table 3 File Metrics

CodeConstruct Entity

it captures the structural elements of the code, such as classes and functions, allowing for detailed structural reporting.

Attribute	Data Type	Description	Constraints
ConstructID	INTEGER	Unique identifier for the code construct.	Primary Key, Auto-Increment
ReportID	INTEGER	Foreign key linking to the parent report.	Foreign Key (AnalysisReport)
ConstructType	VARCHAR(50)	Type of construct (e.g., 'Class', 'Function', 'Import').	Not Null

ConstructName	VARCHAR(255)	The name of the class or function.	Not Null

Table 4 CodeConstruct

Issue Entity

it records all detected issues, including syntax errors, which are a core requirement of the project.

Attribute	Data Type	Description	Constraints
IssueID	INTEGER	Unique identifier for the issue record.	Primary Key, Auto-Increment
ReportID	INTEGER	Foreign key linking to the parent report.	Foreign Key (AnalysisReport)
IssueType	VARCHAR(50)	Category of the issue (e.g., 'Syntax Error', 'Style Violation').	Not Null
LineNumber	INTEGER	The line number where the issue was found.	Not Null
Description	TEXT	Detailed message describing the issue.	Not Null

Table 5 Issue Entity

6.3 Entity-Relationship Diagram (ERD)

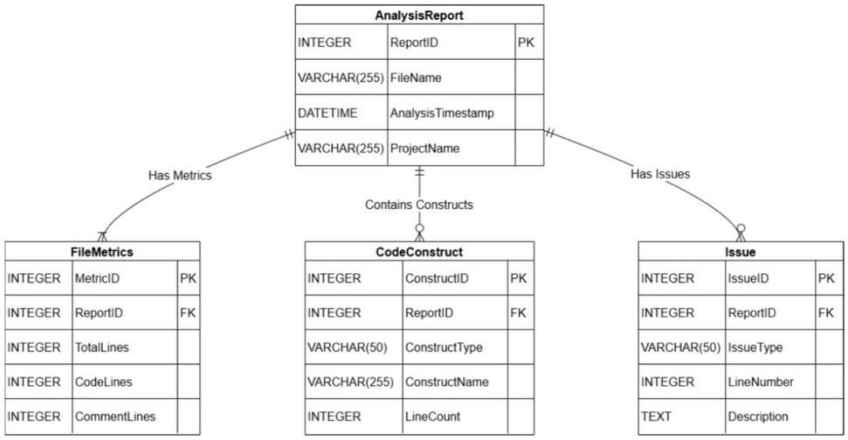

Figure 7 Entity-Relationship Diagram (ERD)

The central AnalysisReport entity maintains a one-to-one relationship with the FileMetrics entity, ensuring that each report has a unique set of file-level statistics. Conversely, the AnalysisReport entity maintains one-to-many relationships with both the CodeConstruct and Issue entities. This structure allows a single analysis report to be associated with multiple code constructs (classes, functions) and multiple detected issues, providing a comprehensive and structurally sound representation of the analyzed source code.

7. External Interfaces

This section describes how the *Python Code Analyzer System* interacts with external systems, tools, and components. The system relies on several external entities to process user inputs, analyze Python files, and deliver the final HTML report.

7.1 User Interface

The user interacts with the system through a web browser. The user uploads a Python (.py) file, requests analysis, and later views or downloads the generated HTML report.

7.2 Web Browser Interface

The web browser sends **HTTP requests** to the web server using:
- **HTTP POST** for file uploads
- **HTTP GET** for retrieving the analysis report

The browser also displays the results returned from the system.

7.3 Web Server Interface

The web server receives requests from the browser, forwards the uploaded file to the analyzer, executes the analysis code, and returns the final report back to the client.
It acts as the communication bridge between the user and the backend system.

7.4 Python Interpreter Interface

The web server uses the **Python Interpreter** to execute the analyzer module.
The interpreter runs all internal logic, including code parsing, complexity analysis, linting, and report generation.

7.5 External Python Libraries

The system communicates with several external Python libraries to complete the analysis:
- **AST** – Parses Python code structure
- **Radon** – Calculates complexity and maintainability metrics
- **Pylint** – Detects style issues and unused imports
- **Jinja2** – Generates the HTML report using templates

These libraries receive code or data from the system and return structured results.

7.6 Communication Protocols

The system uses:
- **HTTP/HTTPS** for communication between browser and server
- **Internal function calls** between the interpreter and external Python libraries
- **Multipart/form-data** for file upload

No external APIs or third-party online services are used.

7.7 Hardware Interfaces
The system does not require specialized hardware. The user interface can be accessed using Laptops or Desktop PCs.

Context Digram

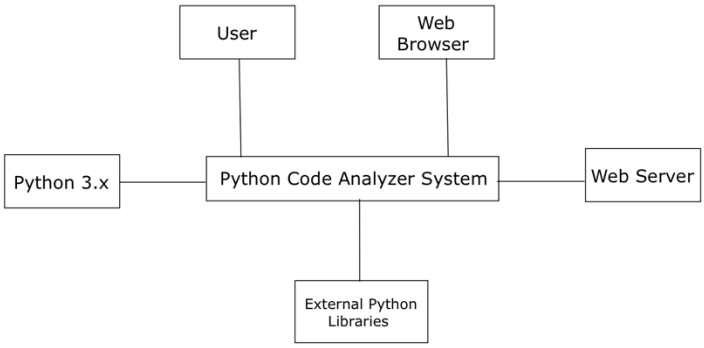

Figure 8 Context Diagram

Sequence Diagram

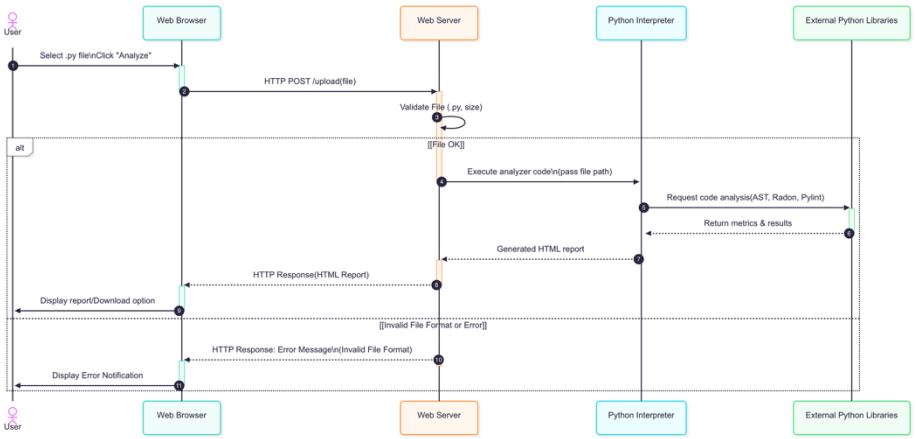

Figure 9 Sequence Diagram

8. Appendices

Glossary of Terms:

Term	Definition
SDD	A document that explains the structure and design of the system.
SRS	Defines functional and non-functional requirements of the system.
Static Analysis	Examining code without running it to detect structure, metrics, and errors.
HTML Report	A structured report generated in HTML format that displays the results of analyzing the Python file.
Code Analysis	The process of examining a Python file to extract information such as line count, functions, classes, and imports.
Syntax Error	An error that occurs when the code violates Python's syntax rules.
Code Metrics	Measurable characteristics of code (line count, functions, classes, etc.).
Import Statement	A command used to include external libraries or modules in Python.
Maintainability Metrics	Measurements that indicate how easy code is to read or modify.
ERD (Entity–Relationship Diagram)	A diagram showing relationships between system data entities.
Activity Diagram	A UML diagram showing workflow or processes within the system.
Deployment Diagram	A diagram showing how system components are deployed on hardware.
Class Diagram	A UML diagram that shows classes and their relationships.

Table 6 of Glossary

Supporting Diagrams:

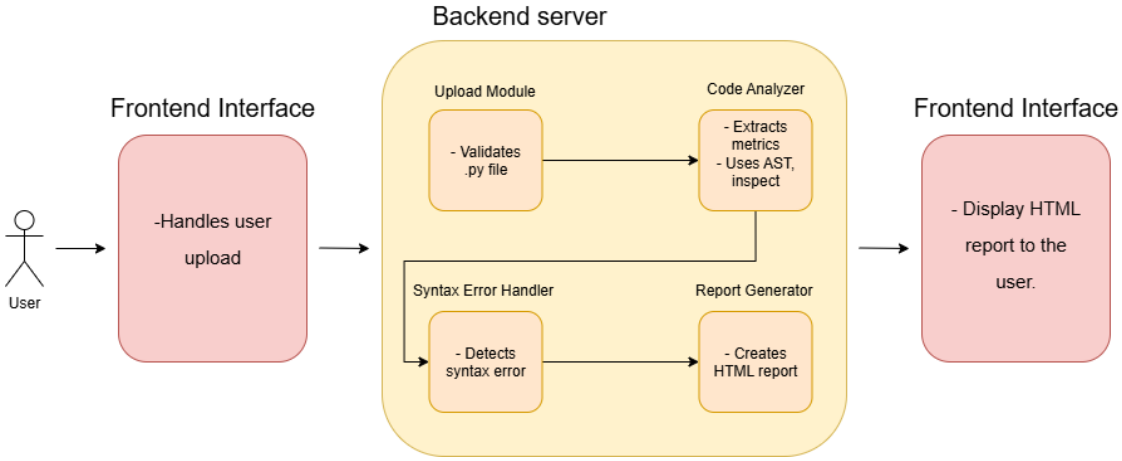

Figure 10 System Architecture Diagram – Python Code Analyzer Tool

References :

1. CSC305_5FA1_Group4_ProjectIdea
2. CSC305_5FA1_group4_projectProposal
3. CSC305_5FA1_Group4_SPMP
4.CSC305_5FA1_Group4_SRS

Evaluation

To be completed by the instructor or supervisor.

6. Project SDD Report (36 points / 6 marks) – Week 13

Section	Excellent (Full Points)	Good (75%)	Fair (50%)	Poor (25% or below)	Points
1. Introduction (3 pts)	Purpose, scope, audience, and references are clearly explained and aligned with SRS.	Covers most parts but some vague.	Minimal explanation of purpose/scope.	Missing or irrelevant.	/3
2. System Overview (4 pts)	High-level architecture and design strategy clearly described with diagrams (e.g., context, layered).	Basic overview given, limited diagrams.	Vague overview, no clear design rationale.	Very poor or missing.	/4
3. Design Considerations (5 pts)	Assumptions, constraints, goals, and trade-offs are wellidentified and justified.	Most elements present, some weak.	Limited mention of goals or constraints; unclear tradeoffs.	Missing or irrelevant.	/5
4. Architectural Design (8 pts)	Comprehensive architecture description with diagrams (component, deployment). Clear decomposition and interactions.	Most elements present but incomplete.	Limited diagrams; weak explanation of interactions.	Very poor or missing.	/8

5. Detailed Design (8 pts)	Each module/component described in detail (data, algorithms, I/O, interactions). UML diagrams included (class, sequence, activity).	Most components covered, some lack detail.	Few components described; diagrams minimal.	Missing or irrelevant.	/8
6. Data Design (3 pts)	Data structures, schema, and relationships clearly defined (ERD/schema included).	Some data design described, missing	Minimal mention of data structures or	Missing.	/3

Section	Excellent (Full Points)	Good (75%)	Fair (50%)	Poor (25% or below)	Points
		relationships.	schema.		
7. External Interfaces (3 pts)	Interfaces with systems/components clearly specified (APIs, libraries, protocols).	Interfaces covered but vague.	Limited or unclear description.	Missing.	/3
8. Appendices & Supporting Info (2 pts)	Glossary, references, and supporting diagrams included.	Some appendices included.	Minimal (e.g., only glossary).	None provided.	/2