Project Based Evaluation

Project Report

Semester-IV (Batch-2023)

Code Scope Offline Source Search

Submitted by:

Nitesh Kushwaha:(2310992001)

Himank Goel :(2310992580)

Sidak Singh:(2310992576)

Prabhnoor Singh:(2310992493)

Supervised by:

Dr. Neha Sharma

Department of Computer Science and Engineering

Chitkara University Institute of Engineering & Technology

Chitkara University, Punjab

Code Scope Offline Source Search

1. Introduction

CodeScope is a powerful, file-based search engine designed specifically for navigating large codebases without requiring internet connectivity. As modern software projects grow in complexity and size, finding specific code elements—functions, classes, variables, or implementation details—becomes increasingly challenging. CodeScope addresses this fundamental developer need through efficient offline indexing and context-aware search capabilities.

Unlike cloud-based alternatives that require network access and potentially expose proprietary code, CodeScope operates entirely on your local filesystem. This approach ensures complete privacy and consistent performance regardless of your network environment. Whether you're working in secure environments with restricted connectivity, developing on-the-go, or simply preferring to keep your code analysis tools self-contained, CodeScope provides a reliable solution.

The engine works by parsing and indexing source files within specified directories, building an optimized structure that enables rapid, targeted searches. Developers can use simple keyword queries or leverage CodeScope's advanced search syntax to pinpoint exactly what they need—from finding all implementations of a specific interface to locating where certain API calls are used throughout the project.

By eliminating the friction between you and your code, CodeScope helps maintain focus during development tasks, supports comprehensive code reviews, and facilitates deeper understanding of unfamiliar codebases—all while respecting the offline, private nature of professional software development.

1.1 Purpose

CodeScope is designed to address several critical needs in modern software development:

1. **Efficient Code Navigation**: Developers spend significant time searching for specific code elements within large and complex

projects. CodeScope enables rapid location of functions, classes, variables, and implementation details, reducing this overhead and improving productivity.

2. **Offline Functionality**: Unlike many modern development tools that require cloud connectivity, CodeScope operates entirely offline. This makes it invaluable for:
    1. Secure environments with network restrictions
    2. Remote or travel situations with limited connectivity
    3. Organizations with strict data privacy requirements
    4. Developers working with proprietary or sensitive codebases
3. **Context-Aware Understanding**: Beyond simple text search, CodeScope provides language-aware indexing that understands code structure. This allows developers to search based on semantic meaning rather than just text patterns.
4. **Codebase Exploration**: For new team members or developers working with unfamiliar code, CodeScope serves as an exploration tool to quickly build mental models of large projects by finding relationships between components.
5. **Privacy Preservation**: By operating locally without sending code to external services, CodeScope ensures that intellectual property and sensitive code remains secure.
6. **Performance and Reliability**: By eliminating network dependencies, searches remain fast and consistent regardless of internet conditions.
7. **Cross-Language Support**: Designed to work with multiple programming languages, CodeScope provides a unified search experience across heterogeneous codebases.

The core purpose of CodeScope is to remove friction between developers and their code, enabling them to maintain focus on problem-solving rather than spending valuable time hunting through files to find relevant code segments.

1.2 Document Conventions

- ER – Entity Relationship
- UC – Use Case

1.3 Intended Audience

CodeScope is primarily designed for:

- **Professional software developers** working with large or complex codebases
- **Technical leads and architects** who need to understand code relationships across projects
- **Security-conscious engineers** requiring offline tools for sensitive codebases
- **Open source contributors** navigating unfamiliar projects
- **Computer science students and educators** analyzing code structures
- **DevOps professionals** searching through configuration and automation code

The tool assumes basic technical proficiency, including familiarity with command-line interfaces and fundamental programming concepts. While designed for experienced developers, CodeScope's intuitive interface makes it accessible to anyone who needs to efficiently search and understand code without relying on network connectivity.

1.4 Project Scope

CodeScope is a lightweight, file-based source code search engine that operates entirely offline, designed to help developers quickly locate and understand code elements across large projects. The system indexes source files within specified directories, enabling rapid searches for functions, classes, variables and other code structures across multiple programming languages including Python, JavaScript, Java, C,C++ . Core functionality includes context-aware code searching, regular expression support, and both command-line and simple graphical interfaces for search operations.

The project specifically focuses on local file operations without requiring network connectivity, supporting codebases up to 1 million lines of code with search response times under 2 seconds. While CodeScope provides powerful static analysis capabilities for search purposes, it intentionally excludes cloud-based indexing, runtime code analysis, version control integration beyond basic file access, and code modification tools, maintaining its focused mission of efficient offline code navigation.

1.5 References

1.GitHub Code Search

- https://github.com/features/code-search
- Global code search across public repositories with semantic understanding

2. GeeksforGeeks
https://www.geeksforgeeks.org
Referred for Java file handling, parsing, and data structure implementation techniques.

3. W3Schools Java Tutorials
https://www.w3schools.com/java
Used to understand basic to intermediate Java syntax, GUI development, and file I/O.

4.Oracle Java Documentation
https://docs.oracle.com/javase/
Official documentation used to understand core Java libraries, JavaFX, and I/O classes

2. Overall Description

2.1 Product Perspective

Tired of wrestling with sprawling codebases? **CodeScope** is your lightning-fast, offline solution for effortlessly navigating and understanding your projects. Designed for developers who value privacy, speed, and independence, CodeScope indexes your local source files, empowering you to instantly find functions, classes, variables, and implementation details – all without ever needing an internet connection.

Imagine: quickly pinpointing every usage of a specific API call, tracing the inheritance of a class, or locating the definition of a critical function, even when you're working in secure environments, traveling with limited connectivity, or simply prefer the control of a self-contained tool.

CodeScope goes beyond simple text search. Its language-aware indexing understands code structure, allowing for context-rich searches that save you precious development time. Whether you're onboarding to a new project, conducting thorough code reviews, or deep-diving into unfamiliar territory, CodeScope acts as your trusted guide, helping you build a clear mental model of even the most complex systems.

CodeScope empowers you to eliminate the friction of code searching, allowing you to focus on what truly matters: building great software. Reclaim your time and gain deeper insights into your codebase with the power of offline code navigation.

2.2 Product Features

CodeScope offers a suite of features designed to facilitate efficient offline code navigation and understanding:

- **Offline Code Search**: Operate entirely without internet connectivity, ensuring consistent performance and privacy.
- **Language-Aware Indexing**: Parse and index code structures (functions, classes, variables) across multiple programming languages, including Python, JavaScript, Java, C, and C++.
- **Rapid Search Capabilities**: Deliver search results in under 2 seconds for codebases up to 1 million lines.

- **Advanced Query Support**: Utilize regular expressions and context-aware syntax to refine search queries.
- **Dual Interface Access**:
  - *Command-Line Interface (CLI)*: For users preferring keyboard-driven workflows and automation.
  - *Graphical User Interface (GUI)*: A simple visual interface for broader accessibility.
- **Multi-Language Support**: Index and search across heterogeneous codebases seamlessly.
- **Privacy Preservation**: Ensure that all code analysis remains local, safeguarding proprietary information.Elite AI+2YouTube+2Product Hunt+2
- **Lightweight Architecture**: Maintain low resource consumption, suitable for various hardware configurations.

2.3 User Classes and Characteristics

CodeScope is tailored for a diverse range of users:

- **Professional Software Developers**: Require efficient navigation through large and complex codebases.
- **Technical Leads and Architects**: Need to understand code relationships and architecture across projects.
- **Security-Conscious Engineers**: Operate in environments with strict data privacy requirements and limited internet access.
- **Open Source Contributors**: Navigate and comprehend unfamiliar projects efficiently.AlternativeTo+2mega-tools.org+2Product Hunt+2
- **Computer Science Students and Educators**: Analyze and understand code structures for learning and teaching purposes.arXiv
- **DevOps Professionals**: Search through configuration and automation code effectively.Elite AI+1ltdhunt.com+1

User Proficiency: Users are expected to have basic technical proficiency, including familiarity with command-line interfaces and fundamental programming concepts.

2.4 Operating Environment

CodeScope is designed to function across various environments:

- **Hardware**: Standard computing devices with no special hardware requirements.
- **Operating Systems**: Compatible with major operating systems such as Windows, macOS, and Linux.
- **Software Dependencies**:
    - *Python 3.x*: Primary programming language for development.
    - *Required Libraries*: Includes sqlite3, tkinter, argparse, pygments, among others.
    - *Optional Tools*: Tree-sitter for enhanced language parsing capabilities.
- **Resource Utilization**: Optimized for low CPU and memory usage, ensuring performance on machines with limited resources.

2.5 Design and Implementation Constraints

Several constraints guide the design and implementation of CodeScope:

- **Offline Operation**: All functionalities must operate without internet connectivity to ensure privacy and reliability.
- **Scalability Limitations**: Optimized for codebases up to 1 million lines; performance may degrade beyond this threshold.
- **Language Parsing**: Focus on supporting a defined set of programming languages; extending support requires additional parser development.
- **Indexing Mechanism**: Utilizes file-based indexing; real-time file tracking and automatic index updates are not implemented.
- **Integration Boundaries**: Designed as a standalone tool; does not integrate with IDEs or version control systems beyond basic file access.
- **User Interface Design**: Prioritizes simplicity and accessibility in both CLI and GUI, avoiding complex configurations.
- **Security Considerations**: Ensures that all code analysis and indexing occur locally, with no data transmitted externally.

3. System Features

3.1 Description and Priority

CodeScope's features are designed for efficient offline code navigation. Core functionalities like offline operation, file-based indexing, rapid and context-aware search are high priority. Supporting features such as advanced search syntax, cross-language support, regular expressions are of medium priority, enhancing usability and flexibility. Ultimately, these features aim to deliver efficient code navigation, facilitate codebase exploration while preserving privacy, ensuring performance, and supporting large projects through static analysis for accurate search results.

3.2 Stimulus/Response Sequences

CodeScope's operation centers on stimulus-response sequences where a **user initiates a search (stimulus)** via keywords, advanced syntax. The **system responds** by querying its offline index and presenting a ranked list of code locations matching the criteria. This cycle repeats as users refine their searches to pinpoint specific code elements within the indexed codebase.

3.3 Functional Requirements

CodeScope must **index specified local source code directories** offline and **allow users to perform searches** using keywords and potentially advanced syntax. The system needs to **rapidly return search results** that are **context-aware**, understanding basic code structure across supported languages (Python, JavaScript, Java, C, C++). Users should be able to initiate searches via a **command-line interface** . The engine must efficiently handle codebases up to 1 million lines.

4. External Interface Requirements

4.1 User Interfaces

CodeScope offers two primary user interfaces:

Command-Line Interface (CLI): This provides a text-based way for developers to interact with CodeScope. Users can type commands and search queries directly into their terminal or command prompt. This interface is geared towards efficiency, automation through scripting, and experienced users who prefer keyboard-driven workflows.

Simple Graphical Interface (GUI): CodeScope also includes an easy-to-use visual interface. This likely involves a window where users can input their search terms, specify directories, and view the search results in a more visually organized manner. The GUI aims to make CodeScope accessible to a broader range of users, especially those less familiar with command-line tools.

4.2 Hardware Interfaces

- No special hardware needed. Uses file system.

4.3 Software Interfaces

- **File System Interface:** CodeScope needs to interact directly with the local file system to:

  - **Read Source Files:** Access and read the content of source code files within the specified directories.
  - **Index Files:** Create and manage the index files that store the parsed information about the codebase.
  - **Locate Files:** Retrieve and display the file paths of search results to the user.

- **User Interface Interfaces:** CodeScope provides two distinct interfaces for user interaction:

  - **Command-Line Interface (CLI):** This interface involves processing text-based commands and displaying text-based output (search results, potential configuration options). It relies on standard input and output streams of the operating system.

- **Graphical User Interface (GUI):** This interface likely uses a graphical toolkit (the description mentions "simple") to provide visual elements like input fields, buttons, and result displays. It interacts with the operating system's windowing system to manage the application's window and user interactions (mouse clicks, keyboard input).

5. Nonfunctional Requirements

- **Fast Performance:** Sub-2-second search for large codebases.
- **Local Scalability:** Handles up to 1 million lines of code.
- **User-Friendly:** Simple GUI and efficient CLI.
- **Reliable Offline Operation:** Consistent results without internet.
- **Secure:** Keeps code local and private.
- **Maintainable:** Easy to update and extend.
- **Resource Efficient:** Reasonable CPU and memory usage.

5.1 Implementation Details

1. Project Setup

- Create a structured folder layout with directories for indexing, searching, UI, and data storage.

- Initialize Git and set up a Python virtual environment.
- Add a requirements.txt for dependencies (e.g., sqlite3, tkinter, argparse, pygments, etc.).

2. File Crawler

- Write a script to walk through the user-specified directory.
- Filter and collect only supported source code files (like .py, .cpp, .js, etc.).

3. Code Parser

- Use a lightweight parser to extract classes, functions, and variable names.
- Optionally use tools like Tree-sitter for language-aware parsing.

4. Search Engine

- Accept a user query (e.g., keyword).
- Run queries on the indexed database and return ranked results with context.

5. Testing

- Test the indexing and search on real codebases of various sizes.
- Measure performance, validate search accuracy, and refine regex or parsing logic.

6. Packaging

- Use tools like pyinstaller to build an executable for offline use.
- Ensure it works without needing Python pre-installed.

7. Documentation

- Write a README with setup, usage, and examples.
- Add diagrams to explain the system architecture and data flow.

5.2 Commands and Scripts Used (Short)

1.Setup

- python -m venv venv → Create virtual environment
- pip install -r requirements.txt → Install dependencies

2. Indexing

- python main.py index /path/to/codebase → Start indexing
- indexer/index_builder.py → Scans files and builds search index

3. Searching

- python main.py search "query" → Perform search
- search/search_engine.py → Queries the database for matches

5. Packaging

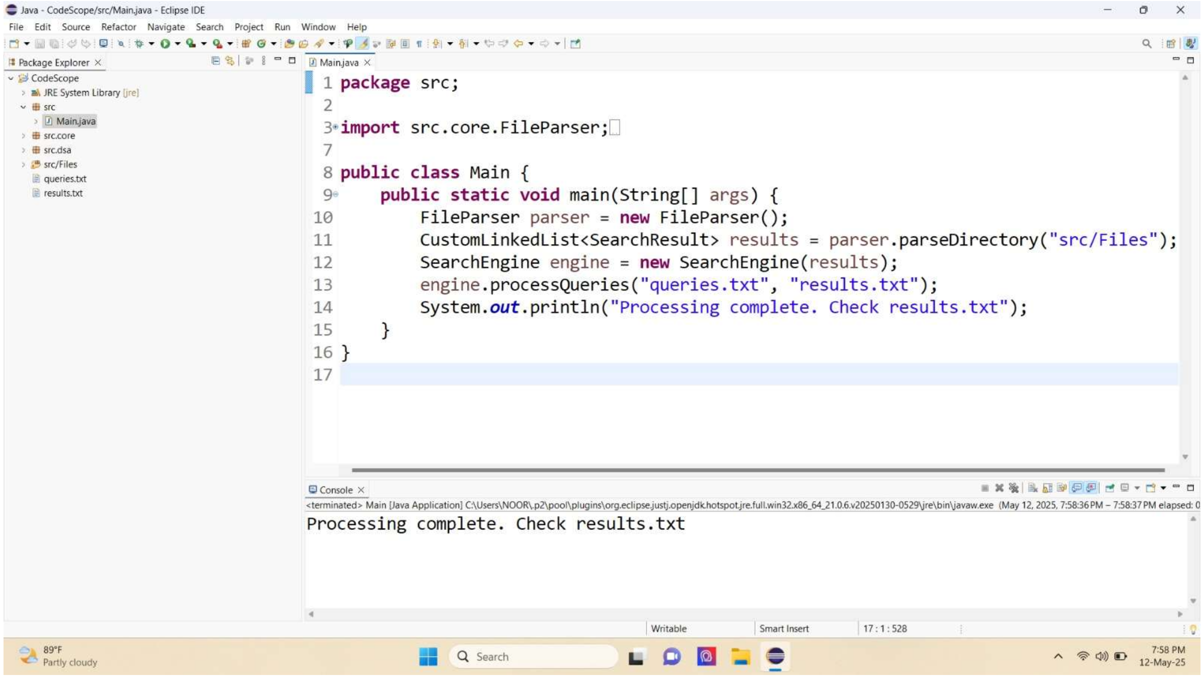
- pyinstaller --onefile main.py → Create standalone executable

6.Testing

- pytest tests/ → Run automated tests (if implemented)

5. 3 **Screenshots and Outputs** There are some screenshots of the code which needs to be done to make complete running project.

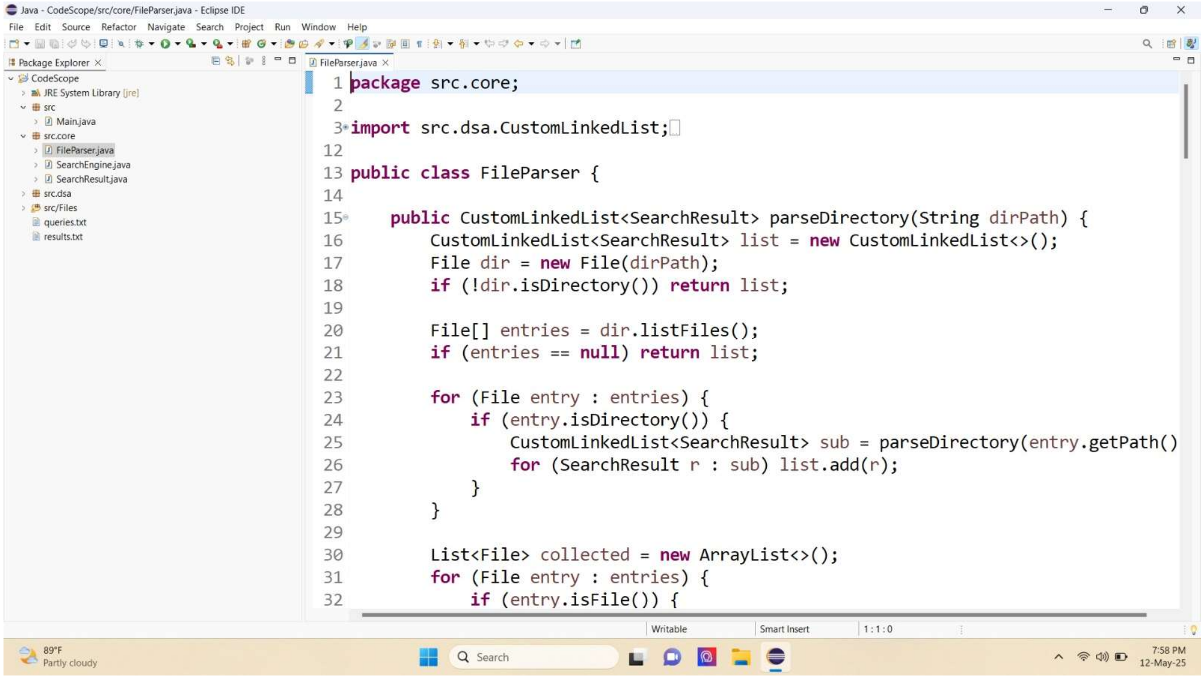
1.Main.java - Initializes the program by creating FileParser and SearchEngine.



```java
package src;

import src.core.FileParser;□

public class Main {
    public static void main(String[] args) {
        FileParser parser = new FileParser();
        CustomLinkedList<SearchResult> results = parser.parseDirectory("src/Files");
        SearchEngine engine = new SearchEngine(results);
        engine.processQueries("queries.txt", "results.txt");
        System.out.println("Processing complete. Check results.txt");
    }
}
```

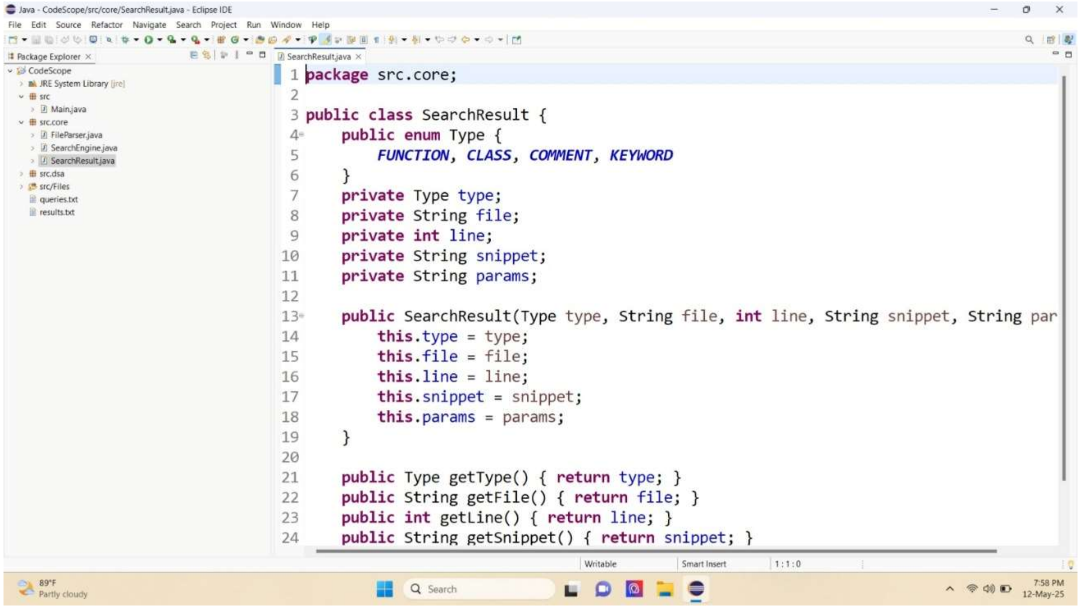
```
<terminated> Main [Java Application] C:\Users\NOOR\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.6.v20250130-0529\jre\bin\javaw.exe (May 12, 2025, 7:58:36 PM – 7:58:37 PM elapsed: 0
Processing complete. Check results.txt
```

2. FileParser.java- Recursively scans a directory (src/Files) for source code files.Identifies code elements (classes, functions, etc.) and creates SearchResult objects.



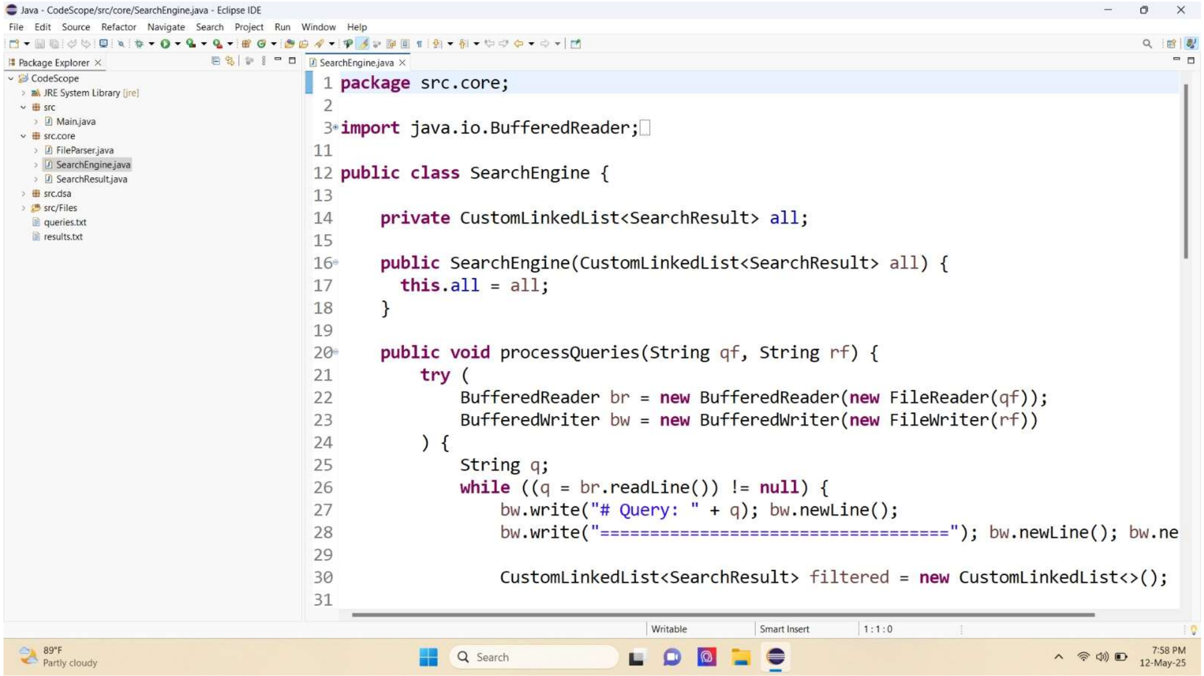
```java
package src.core;

import src.dsa.CustomLinkedList;

public class FileParser {

    public CustomLinkedList<SearchResult> parseDirectory(String dirPath) {
        CustomLinkedList<SearchResult> list = new CustomLinkedList<>();
        File dir = new File(dirPath);
        if (!dir.isDirectory()) return list;

        File[] entries = dir.listFiles();
        if (entries == null) return list;

        for (File entry : entries) {
            if (entry.isDirectory()) {
                CustomLinkedList<SearchResult> sub = parseDirectory(entry.getPath()
                for (SearchResult r : sub) list.add(r);
            }
        }

        List<File> collected = new ArrayList<>();
        for (File entry : entries) {
            if (entry.isFile()) {
```

3. Search Result.java - Stores metadata for each code match:



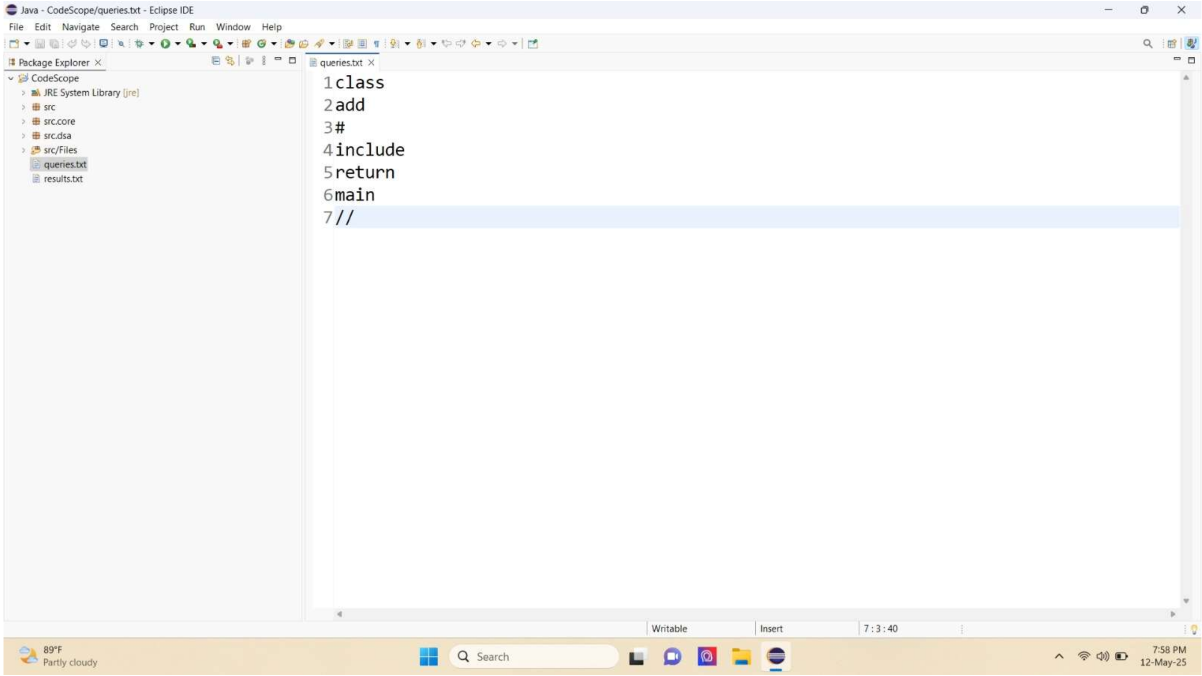
```java
package src.core;

public class SearchResult {
    public enum Type {
        FUNCTION, CLASS, COMMENT, KEYWORD
    }
    private Type type;
    private String file;
    private int line;
    private String snippet;
    private String params;

    public SearchResult(Type type, String file, int line, String snippet, String par
        this.type = type;
        this.file = file;
        this.line = line;
        this.snippet = snippet;
        this.params = params;
    }

    public Type getType() { return type; }
    public String getFile() { return file; }
    public int getLine() { return line; }
    public String getSnippet() { return snippet; }
```

4. SearchEngine.java - Reads search terms from queries.txt (e.g., "class", "add").



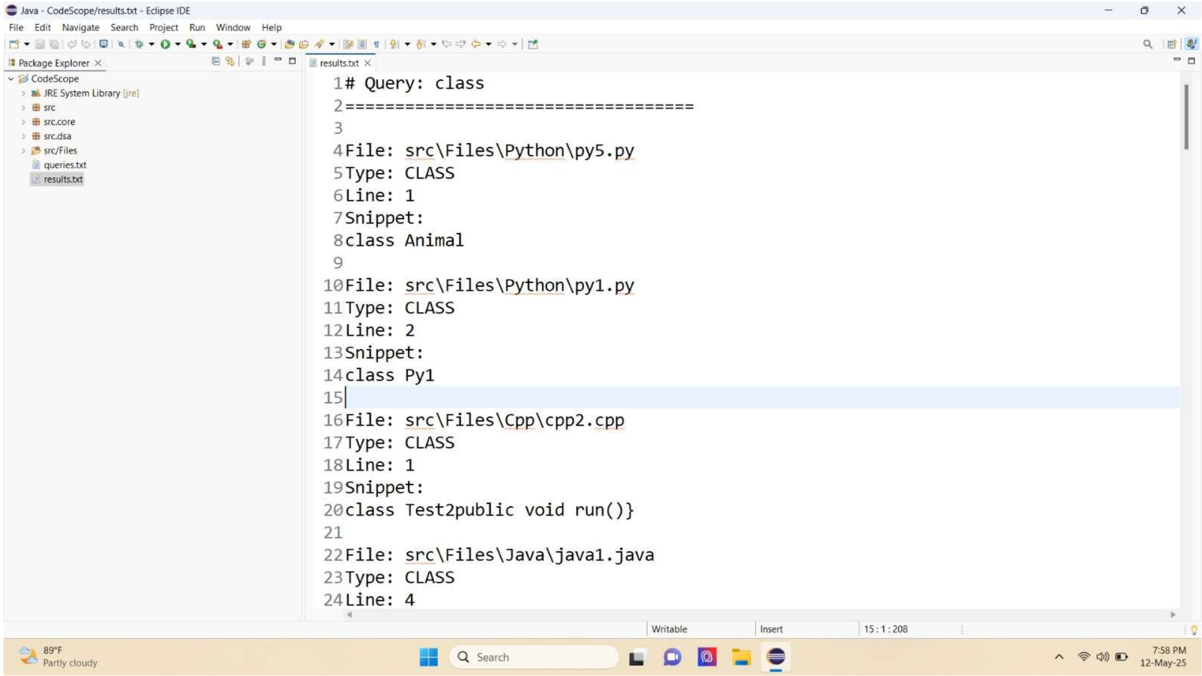
```java
package src.core;

import java.io.BufferedReader;

public class SearchEngine {

    private CustomLinkedList<SearchResult> all;

    public SearchEngine(CustomLinkedList<SearchResult> all) {
      this.all = all;
    }

    public void processQueries(String qf, String rf) {
        try (
            BufferedReader br = new BufferedReader(new FileReader(qf));
            BufferedWriter bw = new BufferedWriter(new FileWriter(rf))
        ) {
            String q;
            while ((q = br.readLine()) != null) {
                bw.write("# Query: " + q); bw.newLine();
                bw.write("==================================="); bw.newLine(); bw.ne

                CustomLinkedList<SearchResult> filtered = new CustomLinkedList<>();

```

5. Queries.txt- Contains search terms



```
class
add
#
include
return
main
//
```

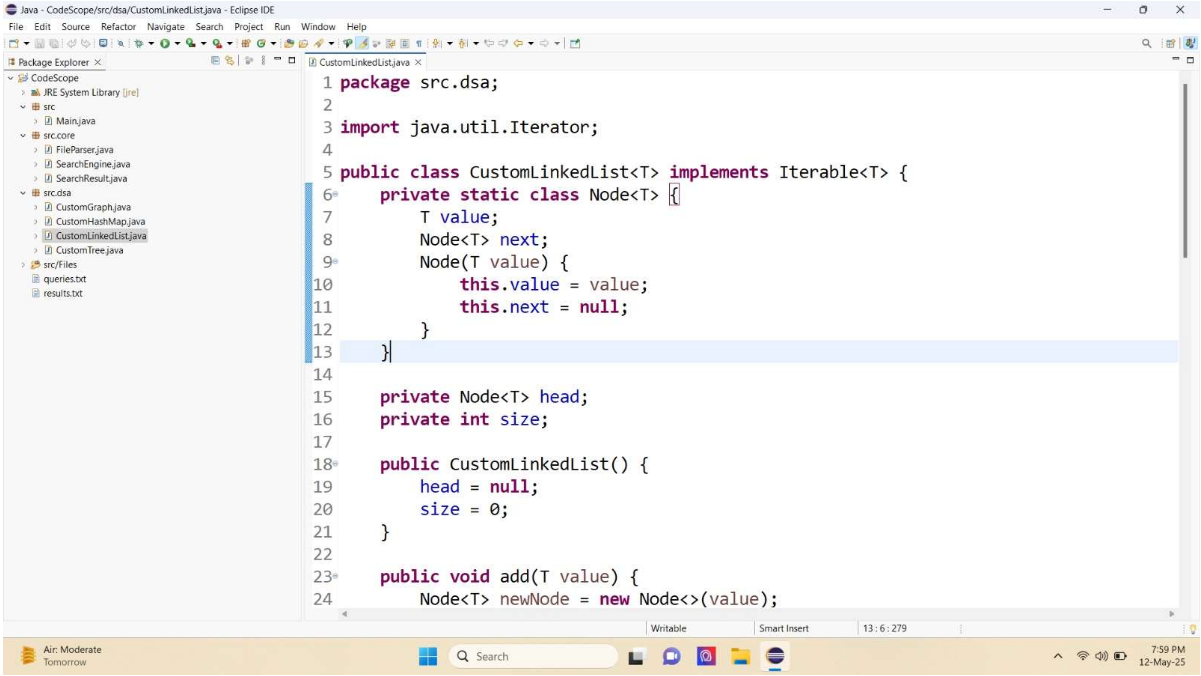
6. Results.txt - Shows formatted search results for queries



```
# Query: class
==================================

File: src\Files\Python\py5.py
Type: CLASS
Line: 1
Snippet:
class Animal

File: src\Files\Python\py1.py
Type: CLASS
Line: 2
Snippet:
class Py1

File: src\Files\Cpp\cpp2.cpp
Type: CLASS
Line: 1
Snippet:
class Test2public void run()}

File: src\Files\Java\java1.java
Type: CLASS
Line: 4
```

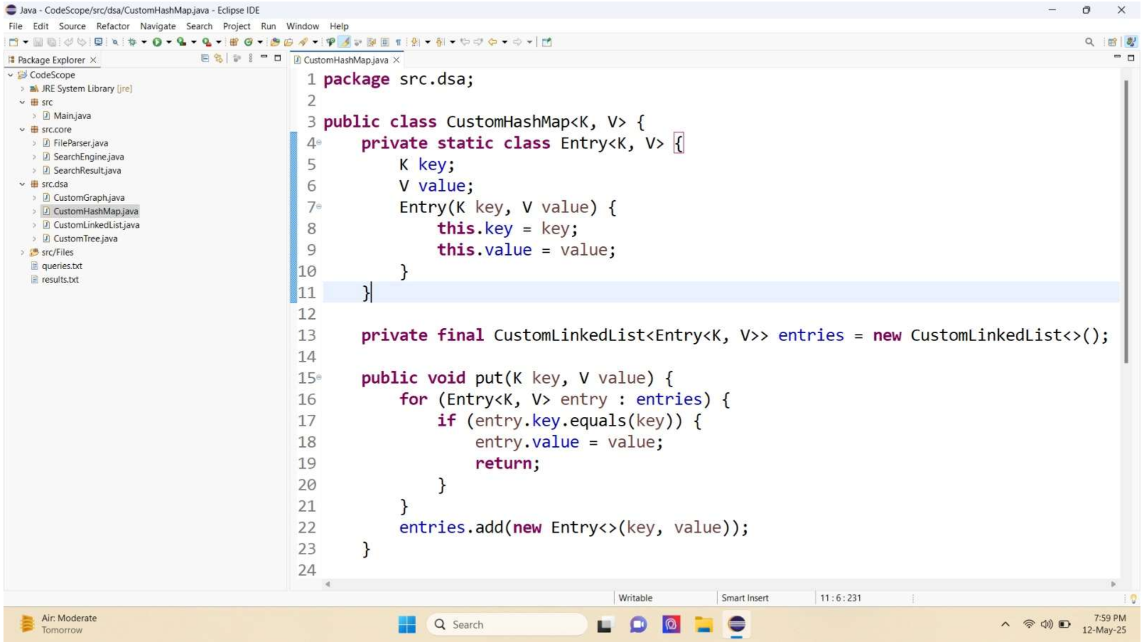
7.Supporting Data Structures

a. **CustomLinkedList.java** - Generic linked list used throughout the system (e.g., storing SearchResult objects).



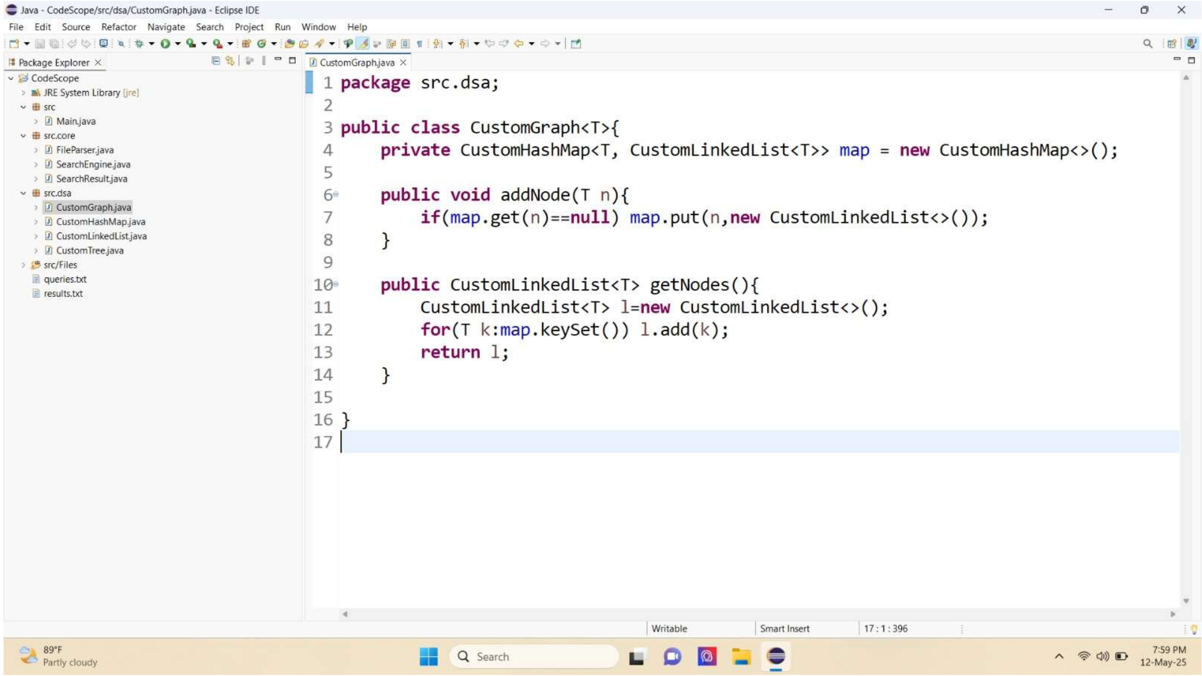
```java
package src.dsa;

import java.util.Iterator;

public class CustomLinkedList<T> implements Iterable<T> {
    private static class Node<T> {
        T value;
        Node<T> next;
        Node(T value) {
            this.value = value;
            this.next = null;
        }
    }

    private Node<T> head;
    private int size;

    public CustomLinkedList() {
        head = null;
        size = 0;
    }

    public void add(T value) {
        Node<T> newNode = new Node<>(value);
```

b.**CustomHashMap.java** - Backend for CustomGraph to map nodes to edges



```java
package src.dsa;

public class CustomHashMap<K, V> {
    private static class Entry<K, V> {
        K key;
        V value;
        Entry(K key, V value) {
            this.key = key;
            this.value = value;
        }
    }

    private final CustomLinkedList<Entry<K, V>> entries = new CustomLinkedList<>();

    public void put(K key, V value) {
        for (Entry<K, V> entry : entries) {
            if (entry.key.equals(key)) {
                entry.value = value;
                return;
            }
        }
        entries.add(new Entry<>(key, value));
    }

```

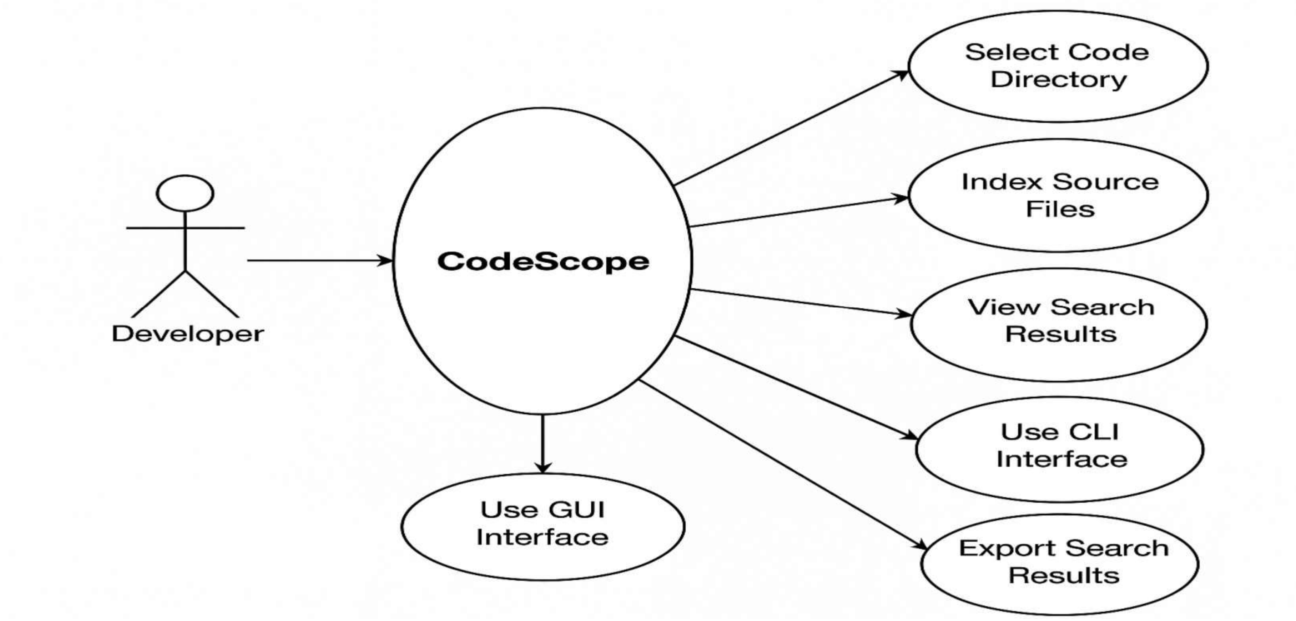
c.CustomGraph.java - Graph implementation



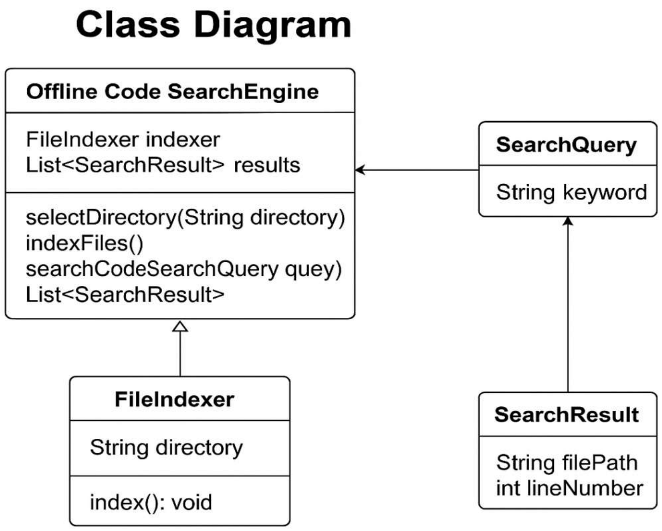
```java
package src.dsa;

public class CustomGraph<T>{
    private CustomHashMap<T, CustomLinkedList<T>> map = new CustomHashMap<>();

    public void addNode(T n){
        if(map.get(n)==null) map.put(n,new CustomLinkedList<>());
    }

    public CustomLinkedList<T> getNodes(){
        CustomLinkedList<T> l=new CustomLinkedList<>();
        for(T k:map.keySet()) l.add(k);
        return l;
    }

}

```

6. Diagrams

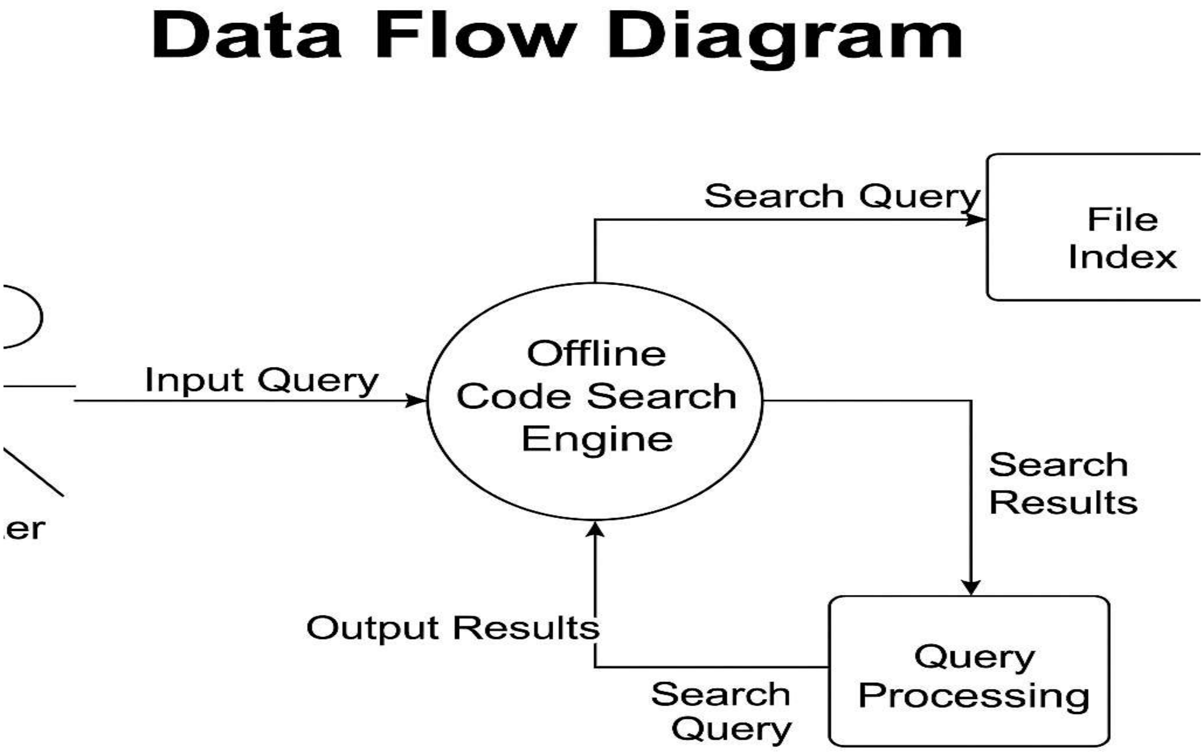
A. Use Case Diagram



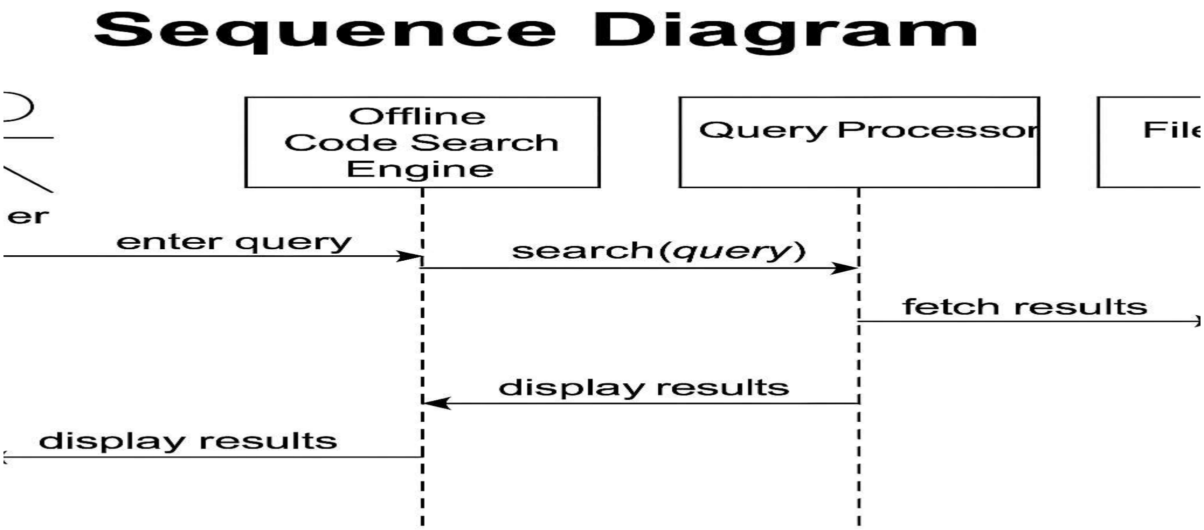
B. Class Diagram



C. Data Flow Diagram (DFD - Level 1)



D. Sequence Diagram



7.Challenges and Limitations

1.Scalability with Very Large Codebases

Although CodeScope supports projects up to 1 million lines of code, performance may degrade with larger repositories due to memory and indexing time constraints.

2.Language Parsing Limitations

CodeScope may not fully understand all language-specific syntax or advanced constructs, especially in dynamically typed or less common languages.

3.Lack of Real-Time Updates

Indexes are built manually and need to be refreshed when the codebase changes, as real-time file tracking is not implemented.

4.Limited Semantic Understanding

While the search is context-aware to a degree, it may not fully grasp code semantics like type inference, runtime behavior, or deep dependency chains.

6.No Integration with IDEs or Version Control

CodeScope is a standalone tool and does not integrate with development environments like IntelliJ or Git-based version tracking.

7.Cross-Language Consistency

Different languages are indexed differently due to syntax variations, which may affect the uniformity of search results across polyglot codebases.

7 Conclusion and Future Work

8.1 Conclusions

CodeScope successfully addresses a critical gap in software development—efficient, offline code searching across large and complex codebases. By providing both a command-line and graphical interface, the tool enables developers to locate functions, classes, and other code elements quickly without relying on internet connectivity. Its lightweight, file-based architecture ensures privacy, speed, and portability, making it ideal for secure environments or remote work scenarios. The use of context-aware indexing further enhances the relevance of search results, improving productivity and code understanding.

8.2 Learnings from the Project

Through *CodeScope++*, I gained hands-on experience in modular system design, custom data structures (like Trie and Graph), file-based storage, and graph algorithms such as DFS and cycle detection. The project also improved my skills in parsing source code, implementing efficient search mechanisms, and documenting large-scale systems effectively.

8.3 Future Enhancements : Future improvements can include real-time file monitoring, IDE plugin integration, support for more programming languages, multithreaded indexing, encrypted storage, and an interactive analytics dashboard to enhance usability and performance.