Integrated Development Environment (IDE)

1. Introduction

This document outlines the specifications and conceptual design for a custom Integrated Development Environment (IDE)-like desktop application. The primary objective is to create a simplified IDE with a rich user interface built using Python and the PySide6 library, providing key functionalities such as file exploration, code editing with tabs, an outline/symbol view, and an integrated terminal.

The deliverables include:

- A set of detailed functional and non-functional requirements.
- The tools and frameworks to be employed.
- A use case diagram illustrating user interactions.

2. Project Overview

Goal:

To develop a desktop-based IDE application that allows users to browse files, open and edit source code in multiple tabs, view code structure (symbols/outline), and run commands in an integrated terminal.

Core Features:

File Explorer (Left Panel):

Displays a tree view of the filesystem directory chosen by the user.

Allows basic file operations: open, rename, delete (with confirmation), and create new files/folders.

Tab-based Code Editor (Center Panel):

Users can open multiple files simultaneously.

Each opened file is represented by a tab at the top of the central area.

Basic text editing functionalities (open, save, cut, copy, paste).

Syntax highlighting (potentially implementable via a third-party Python syntax highlighting library if desired).

Outline/Symbol Window (Right Panel):

Displays a structural overview of the currently active file (e.g., functions, classes, variables).

Updates dynamically when the active file changes.

Clicking on a symbol in this panel navigates to that symbol's definition in the code editor.

Integrated Terminal (Bottom Panel):

Embedded terminal window for running commands.

3. Detailed Specifications

3.1 Functional Requirements

FR1: File Management

FR1.1: The user shall be able to navigate through the directory structure using the file explorer.

FR1.2: The user shall be able to open files from the file explorer by double-clicking.

FR1.3: The user shall be able to create, rename, and delete files/folders via context menus in the file explorer.

FR1.4: Opened files should appear as tabs in the central editor panel.

FR2: Editor Functionality

FR2.1: The user shall be able to edit files in the central editor.

FR2.2: The user shall be able to save changes to a file (Ctrl+S shortcut).

FR2.3: The editor shall support basic editing (cut, copy, paste, undo, redo).

FR2.4: Optionally, the editor may provide syntax highlighting (depending on implementation scope).

FR3: Outline/Symbol Window

FR3.1: The outline panel shall display the hierarchical structure of the current file's symbols (e.g., classes, methods).

FR3.2: Clicking a symbol in the outline shall move the cursor to the corresponding code location in the editor.

FR3.3: The outline shall refresh dynamically when switching tabs or when the file structure changes.

FR4: Integrated Terminal

FR4.1: The user shall have access to a terminal panel at the bottom of the application.

FR4.2: The terminal shall allow running commands, such as "python filename.py" or "git status."

FR4.3: The terminal shall support basic I/O functionality, scrolling, and resizing.

FR5: Window Layout and Docking

FR5.1: The main window shall have a left dock (file explorer), center area (editor tabs), right dock (outline), and bottom dock (terminal).

FR5.2: Docks may be resizable by dragging boundaries.

FR5.3: The user interface shall be responsive and adapt to various screen sizes.

3.2 Non-Functional Requirements

NFR1: Performance

The IDE application should load and display the initial interface within 3 seconds on a standard development machine.

Opening a moderately sized file (up to a few thousand lines) should be performant and not significantly lag.

NFR2: Usability

The interface shall be intuitive, with commonly used features accessible via menus, buttons, and standard keyboard shortcuts.

The layout should follow familiar IDE paradigms so users can easily understand where to find tools.

NFR3: Reliability & Stability

The application should gracefully handle errors such as attempting to open a non-existent file or losing access to a directory.

Automatic recovery (e.g., saving the list of currently opened files) could be considered for future versions.

NFR4: Compatibility

The application shall run on major desktop platforms (given that is a python application it will run :D).

4. Tools and Technologies

Programming Language: Python (Version 3.9 or later recommended)

GUI Framework: PySide6 (Chosen for its Qt-based cross-platform compatibility and modern Python bindings.)

Code Editor Integration:

Native text editing capabilities of PySide6 (QPlainTextEdit or QScintilla for advanced syntax highlighting, if desired).

Outline/Symbol Generation:

Python's built-in ast module or a custom parser to identify classes, functions, and methods in Python files.

For other languages, a language server or simple regex-based symbol extraction can be used depending on scope (or even lexer parser if I have time).

Terminal Integration:

A QTermWidget (if available) or a suitable PySide6-compatible terminal emulator widget. Alternatively, a subprocess interface combined with a text widget for basic terminal I/O.

Version Control: GitHub for source code repository and version control.

IDE for Development: Visual Studio Code (for developing this project itself). The final product is a standalone application.

5. Use Case

Below is a textual description of the use case.

Actors:

User (the person interacting with the IDE).

Sorry but that's all 😊.

Use Cases:

Open a Project Directory

User selects a directory to load into the file explorer.

System displays the directory structure in the file explorer.

Open a File

User double-clicks a file in the file explorer.

System opens the file in a new tab in the central editor.

Edit and Save File

User edits text in the opened file's tab.

User triggers save (Ctrl+S or via File > Save).

System writes changes to the file on disk.

View Outline

User selects an open file tab.

System analyzes the file's structure and displays symbols in the outline panel.

User clicks on a symbol.

System moves editor's cursor to that symbol's position.

Use Terminal

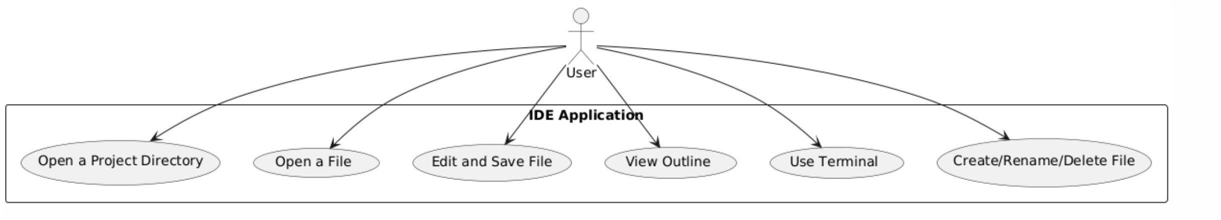
User types a command into the terminal panel.

System executes the command and displays output in the terminal panel.

Create/Rename/Delete File

User right-clicks in file explorer, selects “New File”/“Rename”/“Delete.”

System performs the requested file operation and updates the file explorer view.



6. Conclusion

This report presents a comprehensive overview of the proposed IDE application, including its core functionalities, chosen technologies, and a use case diagram. The outlined specifications and requirements serve as a foundation for development, ensuring the final product meets user expectations for basic IDE capabilities.