Docker container python IDE

Youssef Ehab Gamaledin
School of Computing
Asia Pacific University of Technology & Innovation (APU)
Kuala Lumpur, Malaysia
TP047161@mail.apu.edu.my

Lee Kim Keong
School of Computing
Asia Pacific University of Technology & Innovation (APU)
Kuala Lumpur, Malaysia
kimlee@staffemail.apu.edu.my

Imran Medi
School of Computing
Asia Pacific University of Technology & Innovation (APU)
Kuala Lumpur, Malaysia
imran.medi@staffmail.apu.edu.my

***Abstract*—Current IDEs consumes a high amount of disk memory while having poor usability. This project focuses on developing a browser-based Python IDE with a friendly and non-complex user interface developed based on their feedback. Whilst consuming less disk memory by using Docker containers.**

Keywords—Python, integrated development environment, docker, user interface, disk storage, code editor, file viewer, compiler, debugger

I. Introduction

The Docker container Python Integrated Development Environment (IDE) project is about developing and designing a browser-based Python IDE using Docker Containers. The development of the Python IDE engages the integration of various utilities, components, and tools. For an uncomplicated development, integration, and maintenance of the Python IDE, it will be broken into five main components. The five components of the IDE are; user-interface, code editor, files view, compiler, and debugger. After developing the Python IDE, a docker image will be created and uploaded to the Docker container registry. Docker containers are isolated software packages, which bundle their configurations, libraries, and software. Docker is a platform that operates a virtual operating system level to deliver containers.

II. Problem Statement

Various Python IDEs have common essential components such as; user-interface, code editor, files view, compiler, and debugger [1]. These essential components are adequate to construct a powerful Python IDE. Nevertheless, the most powerful Python IDEs from well-reputed organisations have several limitations, which may contribute to a bad experience for the users.

A major limitation in most of the Python IDEs is their unfriendly user interface and complex structure. A clogged structure does not satisfy the user and can increase the churn rate. It may also delay the code development and rise confusion for the user. Furthermore, some of the modern Python IDEs contains outdated graphics that might be misled or misunderstood. Novice programmers find most IDE's user interface complex to understand [2]. Nowadays, Python IDEs should be perfectly balanced between having a friendly user interface and having a powerful development environment; to ensure the best possible experience for both novice and expert programmers. The user interface should have an organised structure that downturn any confusion and get the user straight to code development.

One of the noteworthy limitations in current Python IDEs is their consumption of storage disks [2]. Most Python IDEs consume an enormous amount of disk memory [3]. For example, Microsoft Visual Studio 2019 consume 4GB for its community version [4], and 10GB for its enterprise version [4]. JetBrains' PyCharm consume approximately 3.5GB of disk memory [5]. As a result, the computer's speed is decelerated.

Thus, the problem statement can be summarized as that most Python IDEs have poor usability while consuming a large amount of disk storage.

III. Literature Review

As mentioned before the five main components of the Python IDE are; User interface, Code editor, Files view, Compiler, and Debugger. The user interface is the biggest component among the five, as it characterizes how friendly and noncomplex the user interface of the Python IDE is. The code editor is the crucial and complex component, as this is the main interaction between the user and the Python IDE. The files view enables efficient project management related to the project files organisation. The compiler for building and running the code. The Debugger to perform a debugging process on Python scripts. These five components are the minimum requirements for the Python IDE.

A. Docker

Docker is a tool that enables us to quickly deploy applications in a sandbox for execution on the host operating system [6]. Sandboxes are containers that provide a logical packaging approach for abstracting applications from their actual running environment [7]. Due to this decoupling, container-based applications are deployed with ease and consistency. Eventually, containers lead to increased efficiency, which results in better utilisation of computer's resources [6]. There are various alternatives to Docker, for example, Hyper-V containers [8], LinuxContainers [9], and rkt [10]. So what makes Docker the best choice in containerising the Python IDE?

- Provides application packaging, together with all of its dependencies, into a standardised unit for better utilisation [7],
- Offers a low overhead, allowing for more efficient usage of the underlying system and resources [7], and
- It runs containers without a hypervisor, an emulator that generates and execute virtual machines, making it portable and lightweight [7].

Docker operates as follows:

a) A dockerfile is created which includes the necessary software to run the Python IDE,

b) A Docker image will be built from the dockerfile. Docker images are the application's blueprint that serves as the foundation for containers.

c) A Docker container is created to run the Docker Image.

The interaction with the operating system will be through Docker's background service, Docker daemon [7]. Eventually, the Python IDE's docker image will be uploaded to the Docker registry, which is a repository for Docker images.

B. *User Interface*

Trachsler [11] stated that when designing a user interface for an IDE, an important matter is to keep visual elements at a minimum; providing only the essential functions. According to him, a self-describing user interface is crucial to allow users to use the IDE without any detailed instructions. On the other hand, Staub [12] emphasized that since all browsers are now resizable, the user interface should be auto adjustable to fit the browser's width. Additionally, she added that the code editor and the files view should be resizable by the user to match their preferences.

C. *Code Editor*

This is the crucial component, where all the interactions between the user and the Python IDE go through. Thus, the following features are important for the code editor; line numeration, syntax highlighting, error line highlighting, automatic indentation, and automatic closing brackets or braces [11]. Additionally, another important feature is the ability to highlight the same variable's names when one of them is clicked on or highlighted [13]. Both Trachsler [11] and Staub [12] have stated that the following are the best browser-based open-source code editors: 1) Ace; 2) Codemirror (v6.0) [15]. The Ace is a stand-alone JavaScript code editor while Codemirror is a versatile in-browser code editor. Comparisons between them have been carried out in Table I.

TABLE I. CODE EDITOR COMPARISONS

Code Editor	Advantages	Disadvantages
Ace (v1.4)	– Compatible with other libraries [12], and – Edit and load large files rapidly [11].	– Berkeley Software Distribution licenses [14], and – Support only new versions of browsers [14].
Codemirror (v6.0)	– Excellent documentation, and – Multiple content manipulation methods.	– Text-area-based, not a complete code editor, and – L2 code quality, which only ensures that the interfaces are clean.

Codemirror is chosen over Ace because of its better documentation, which will support its integration with the Python IDE. Moreover, because of its content manipulation methods, which will provide better options in the design phase.

D. *Files View*

Files view are a bullet-point view of files uploaded or created by the user. It enables observation of project files for the user to operate over them. According to Staub [12], developing a web-based files view with JavaScript is recommended for better manipulation of files. Staub [12] have explained how to create the files view component with JavaScript in five steps:

a) Create an event listener for an event of a change to the HTML element "<input>".

b) Filter files selected to allow files only ending with "py".

c) Instantiate JavaScript FileReader to read the files allowed.

d) A Files List is created to store the files.

e) Any existing files in the files list before the event of change is removed and the editor is cleared.

E. *Compiler*

Developing a Python compiler from scratch is beyond the scope of this project. The minimum requirements for a compiler, apart from compiling code, are as follows;

- If an error is found by the compiler it should abort the build [13],
- When an error is found, the compiler should be able to track the name of the file where the error exists [13],
- The compiler should provide error messages that are easily comprehended [16], and
- The compiler should be quick so that the result can be advantageous [13].

Therefore, an analysis is carried out, as shown in Table II, to compare available Python in-browser compilers. To develop the Python IDE upon the one that suits the scope of this project. Brython [18], PyPy.js [19], Skulpt [20], Transcrypt [21], and Pyodide [22] are the available open-source Python in-browser compilers to choose from [17].

TABLE II. IN-BROWSER COMPILER COMPARISON

Compiler	Description	Advantages	Disadvantages
Brython	Brython is a client-side Python interpreter, it replaces JavaScript with Python.	– Only 500kB in size, with a build-in DOM, and – Provides bindings customization to browser API.	– Poor documentation , and – Lacks development tools, compared to the other compilers.
PyPy.js	Consists of a PyPy Python compiler, compiled in JavaScript.	– Provides a rapid and compliant environment for Python development, and – Features a Just-In-Time compiler.	– 12MB in size, and – Limited built-in Python modules.
Skulpt	A JavaScript client-side Python interpreter.	– Supports asynchronous programming , and – It is 950kB in size.	– Specifically for small Python programs, and – Short documentatio n.
Transcrypt	A source-to-source Python	– Supports access to any	– Does not support pure Python

	translator into JavaScript.	JavaScript library, and – Better in programming front-end Python applications.	libraries like; “Matplotlib”, and – Not the best in compiling complex Python programs since it is just a “transpiler”.
Pyodide	Is a Python-to-WebAssembly compiler, browser-based Python compiler via WebAssembly	– Includes most of Python’s popular scientific libraries, and – Just-in-Time compiler, which provides fast and reliable compilation.	– Compiler size increases when using many libraries, and – Development stopped in 2019.

Pyodide is chosen over the rest of the Python in-browser compilers; because of its Just-in-Time compiling feature and transparent object conversion, which ensures quick output, input, and errors. Additionally, Pyodide includes many of the important Python’s scientific libraries.

F. Debugger

There are hardly any open-source Python debuggers. Therefore, the development and integration of a Python debugger from the compiler are required. Debugging is a mechanism for discovering software bugs within a program. Kohn and Manaris [23] added that debuggers are not entirely for discovering bugs, but also to provide the user with an observation of the internal state of a compiler when executing programs. The execution of a debugger is performed by aligning breakpoints to lines of code, which provide users with the ability to interrupt their program using specific conditions [24]. According to Naert, Azhari, and Dagenais [24], condition checking should be executed in the program context, which lead to faster condition checking and more efficient debugging interaction with conditional breakpoints.

The debugger should be visualised to show the program’s state; variables, lists, functions, and values [23]. Moreover, Kohn and Manaris [23] have stated that it is crucial to highlight any relationships in the program’s state; a relationship between any functions, variables, and objects. Additionally, according to them, any trace function invocated should directly update the debugger’s visuals.

IV. Similar Systems

The Python IDE is a combination of a novice programmer’s IDE and an expert programmer’s IDE. It is to develop one platform that is efficient for all kinds of programmers. Therefore, in terms of similar systems, we will be looking at IDEs that are developed for experienced Python programmers, and IDEs that are developed for Python-learning programmers.

Codesters is an introductory web-based Python programming environment, released in 2014 and used for creating Python programs [25]. The strengths of Codesters according to Šiaulys [25], is that it supports Python conditionals, loops, variables, functions, and objects. On the other hand, Codesters weakness is that it was developed primarily for education and is not suitable for expert programmers [26]. WebTigerJython is the web version of TigerJython, released in 2018 and used for teaching programming in Python [27]. The strengths of WebTigerJython according to Schneider [27], is that it supports Python 3 with two different debuggers; a step-by-step debugger and a breakpoint debugger. Similarly, WebTigerJython’s weakness is that it was mainly developed for teaching programming and is not a platform for expert programmers [28].

PythonAnywhere is also a web-based IDE, released in 2012 which executes Python and have a command-line console [29]. According to Sutton and Swickard (2020), PythonAnywhere provides running Python scripts with data progressing rapidly and easily. Repl.IT is an online IDE, released in 2016 which supports Python and other programming languages [30]. According to Kusumaningtyas, Nugroho, and Priadana [30], Repl.IT provides the most popular Python libraries using a friendly user interface with minimum internet data usage.

V. Proposed System Architecture

The Python IDE’s architecture is demonstrated in this section along with Docker’s architecture, for a more comprehensive approach on how the Python IDE operates using Docker.

A. Docker

Docker operates using client-server architecture, Docker client and Docker daemon, both run on the same system. The Docker client interacts with the Docker daemon to run, build, and allocate Docker containers. The interactions between the Docker client and the Docker daemon are fulfilled using REST API through UNIX socket for macOS and Linux, and through network interface for Windows operating system [7]. Fig. 1 manifests Docker’s architecture.

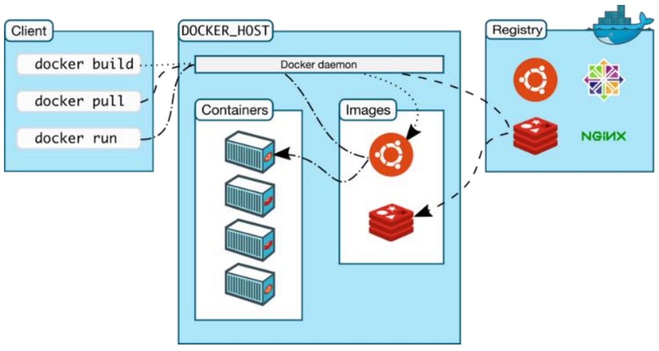

Fig. 1. Docker’s architecture [7]

The Docker client is what Docker users usually communicate with, to send API requests to one or more Docker daemon. The Docker daemon waits until it receives an API request to and controls Docker’s networks, images, and containers. The Docker registry is a public hub, where images are stored. Docker is fully developed to deliver its functionalities using the GO programming language, along with various features of the Linux kernel [7]. For Docker to run a container, it uses namespace technology to isolate containers [7].

B. *System Design*

In this section, the appropriate design methodology is chosen for a detailed system demonstration. Structured System Analysis and Design Methodology (SSADM) is chosen to demonstrate the Python IDE's system. Therefore, the context diagram, DFD level 0 and DFD level 1 are provided below.

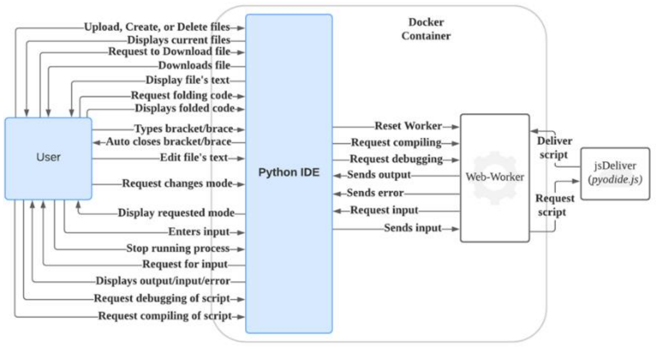

Fig. 2. Python IDE's context diagram

As demonstrated in the Fig 2., Docker will containerise the Python IDE with the web-worker related. A web-worker is JavaScript running independently in the background. It is used to prevent scripts from affecting the performance of the Python IDE. Once the web worker is created, it will request the compiler's script (pyodide.js) from jsDeliver. The jsDeliver is a Content Delivery Network (CDN), used to deliver the compiler's JavaScript to run in the web-worker independently.

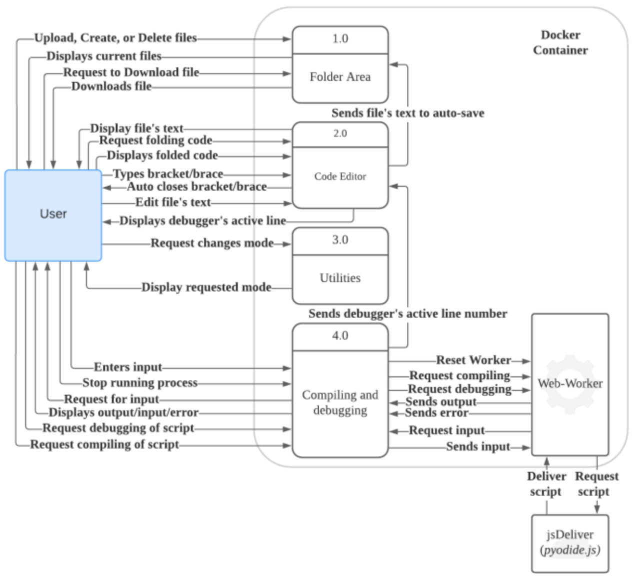

Fig. 3. Python IDE's DFD Level 0

As demonstrated in Fig 3., the Python IDE is integrated with four different processes. Code editor, Compiling and debugging, Folder area, and Utilities. Each process provides its name function, such as the code editor process provides all code editor functionalities and features. On the other hand, the utility process provides different necessary functions and features, such as changing Python IDE's mode/theme.

In Fig 4., the Compiling and debugging process is broken down into its subprocesses. To demonstrate how the process works internally.

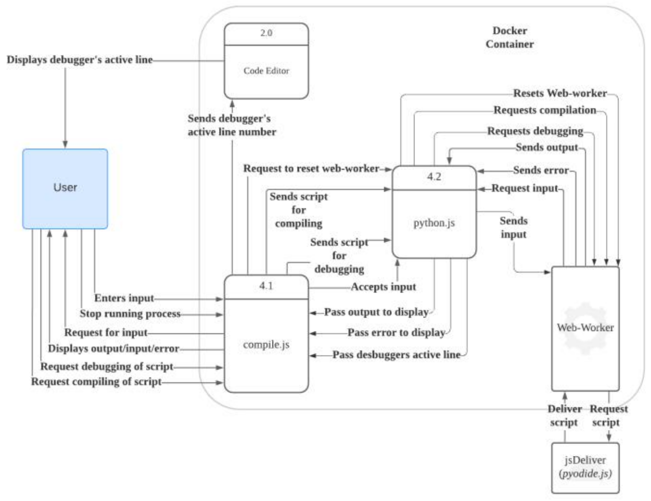

Fig. 4. Python IDE's compiling and debugging Level 1

The folder area process is decomposed into its subprocesses in Fig 5. To illustrate how the internal process operates.

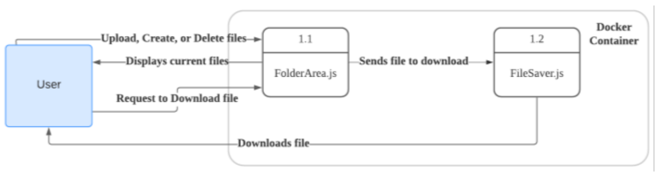

Fig. 5. Python IDE's folder area Level 1

The code editor process is divided down into its subprocesses, as shown in the Fig. 6. To demonstrate how the internal system operates.

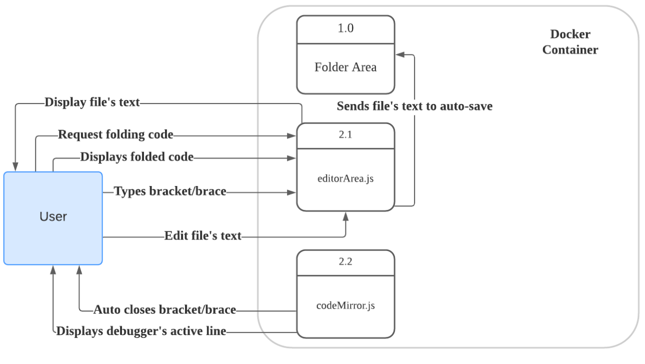

Fig. 6. Python IDE's code editor Level 1

VI. CONCLUSION

Conclusively, in this project, research was conducted to understand more, find solutions, and discover development methods to be able to fully develop the Python IDE to achieve its requirements and objectives. A questionnaire was carried out to support the designing stage of the Python IDE. To get feedback from the users about the user interfaces created and what can be improved, to ensure user-friendly interfaces. Additionally, to understand what programmers prioritise when choosing an IDE. The development stage started shortly after the designing stage.

The code editor and the compiler were chosen to be open-sourced and integrated with the Python IDE. The user interface and the files view are to be developed from scratch, and the debugger is to be developed from the compiler, to ensure its suitability with the scope of the Python IDE. By using the Docker container the Python IDE was able to fully operate using a reduced amount of memory on the disk storage.

The Python IDE's system architecture was provided, as well as the implementation and the integration of its component for a more comprehended overview. Three types of testing were conducted in the testing stage to ensure the quality of the Python IDE and to validate the system. All the test scenarios in the unit and integration testing have successfully passed. The User Acceptance Test was very helpful, as it has discovered a bug regarding the debugger component which will be fixed in the next minor release, version 2.1. Additionally, introduced very amazing features to have in the Python IDE, all of them were noted and scheduled to be added to the system in the next versions of the Python IDE. Eventually, in the documentation stage, all the above stages were documented in an academic writing manner.

REFERENCES

[1] A. Walker, "What is an IDE (Integrated Development Environment)? - G2," 2018. [Online]. Available: https://www.g2.com/articles/ide.

[2] S. Travarca, "Reasons for an Integrated Development Environment," VantageOne Software, 2020. [Online]. Available: https://vantageonesoftware.com/reasons-integrated-development-environment/.

[3] Singh, "What is Pycharm Ide? what is PyCharm used for?," TechGeekBuzz, 2020. [Online]. Available: https://www.techgeekbuzz.com/what-is-pycharm/.

[4] "Visual Studio 2019 system requirements," Microsoft, 2019. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements.

[5] "Install PyCharm," PyCharm, 2021. [Online]. Available: https://www.jetbrains.com/help/pycharm/installation-guide.html.

[6] M. Özateş, "Things you need to know about docker to get started," Medium, 2020. [Online]. Available: https://medium.com/carbon-consulting/things-you-need-to-know-about-docker-to-get-started-565979482a86.

[7] "Why Docker?," Docker, 2021. [Online]. Available: https://www.docker.com/why-docker.

[8] F. Stroud, "What are Hyper-V containers?," Webopedia, 24-May-2021. [Online]. Available: https://www.webopedia.com/definitions/hyper-v-containers/.

[9] "Container and virtualization tools," Linux Containers, 2021. [Online]. Available: https://linuxcontainers.org/.

[10] "Rkt topic page," Red hat, 2021. [Online]. Available: https://cloud.redhat.com/learn/topics/rkt.

[11] N. Trachsler, "WebTigerJython - A Browser-based Programming IDE for Education," thesis, ETH Zurich, Zurich, 2018.

[12] J. Staub, "xLogo online - a web-based programming IDE for Logo. Master Thesis," thesis, ETH Zurich, Zurich, 2016.

[13] N. Mitchell, M. Kiefer, P. Iborra, L. Lau, Z. Duggal, H. Siebenhandl, M. Pickering, and A. Zimmerman, "Building an Integrated Development Environment (IDE) on top of a Build System," IFL, vol. 2, no. 4, pp. 1–5, Sep. 2020.

[14] "The High Performance Code Editor for the web," Ace. [Online]. Available: https://ace.c9.io/.

[15] "Codemirror," CodeMirror, 2021. [Online]. Available: https://codemirror.net/.

[16] A. Henley, J. Ball, B. Klein, A. Rutter, and D. Lee, "An Inquisitive Code Editor for Addressing Novice Programmers' Misconceptions of Program Behavior," thesis, Cornell University, New York, 2021.

[17] Y. Khalid, "Running Python in the Browser," Yasoob, 2019. [Online]. Available: https://yasoob.me/2019/05/22/running-python-in-the-browser/.

[18] P. Quentel, Brython, 2021. [Online]. Available: https://brython.info/.

[19] PyPy.js, 2021. [Online]. Available: https://pypyjs.org/.

[20] Skulpt, 2021. [Online]. Available: https://skulpt.org/.

[21] "Python in the browser," Transcrypt, 2018. [Online]. Available: https://www.transcrypt.org/.

[22] Pyodide, 2019. [Online]. Available: https://pyodide.org/.

[23] T. Kohn and B. Manaris, "Tell me what's wrong," Proceedings of the 51st ACM Technical Symposium on Computer Science Education, 2020.

[24] P. Naert, S. V. Azhari, and M. Dagenais, "Interactive and targeted runtime verification using a debugger-based architecture," Journal of Systems Architecture, vol. 115, 2021.

[25] T. Šiaulys, "Modelling the System For Interactive Tasks Development: Engagement Taxonomy For Introductory Programming Tools," thesis, Vilnius University, Vilnius, 2020.

[26] Codesters, 2020. [Online]. Available: https://www.codesters.com/.

[27] J. Schneider, "Design and Implementation of a Graphics Window and Debugger for WebTigerJython," thesis, ETH Zurich, Zurich, 2020.

[28] WebTigerJython, 2018. [Online]. Available: https://webtigerjython.ethz.ch/.

[29] S. Sutton and K. Swickard, "Text mining 101," The Serials Librarian, vol. 78, no. 1-4, pp. 3–8, 2020.

[30] K. Kusumaningtyas, E. D. Nugroho, and A. Priadana, "Online integrated development environment (IDE) in supporting computer programming learning process during COVID-19 pandemic: A comparative analysis," IJID (International Journal on Informatics for Development), vol. 9, no. 2, pp. 66–71, 2020.