An Implementation of Web-based Personal Platform for Programming Learning Assistant System with Instance File Update Function

Soe Thandar Aung, Lynn Htet Aung, Nobuo Funabiki, Shigo Yamaguchi,
Yan Watequlis Syaifudin, and Wen-Chung Kao

***Abstract*—Java, recognized for its reliability and portability in object-oriented programming, has found application in diverse systems such as enterprise servers, smartphones, and embedded systems. To facilitate self-study in Java programming, we have developed *Java Programming Learning Assistant System (JPLAS)* that provides a range of exercise problems tailored to support novice students at different skill levels. In this paper, we implement the web-based personal platform for JPLAS using *Node.js*, named *NPLAS*. *Docker* is adopted for its easy and solid deployments to novice students. The *instance file update function* is incorporated to help a teacher to distribute the files of the new/updated problem instances to the students, using *Angular* and *Laravel*. In addition, we extend *NPLAS* to *Python programming learning*. For evaluations, we asked teachers and students in three universities in Japan and Indonesia, to install *NPLAS*, solve instances on it, and update instance files. Then, all of them could successfully complete them, which confirms the efficiency and validity of the proposal.**

***Index Terms*—Java, Python, NPLAS, Node.js, Laravel, Angular, Docker, update.**

I. Introduction

JAVA has been extensively used in industries for decades in various application systems including mission critical systems in large enterprises and small-sized embedded systems, as a reliable and portable object-oriented programming language. Thus, the cultivation of *Java programming* engineers has been in high demands amongst industries. A great number of universities and professional schools are offering *Java programming courses* to meet these needs.

To enhance *Java programming* studies, we have developed *Java programming learning assistant system (JPLAS)* [1]. JPLAS offers various types of exercise problems to assist self-studies of Java programming at different levels by novice students, where the answer from a student is automatically marked at the system. They include the *grammar-concept understanding problem (GUP)* [2], the *value trace problem (VTP)* [3], the *mistake correction problem (MCP)* [4], the *element fill-in-blank problem (EFP)* [5], the *code completion problem (CCP)* [6], the *phrase fill-in-blank problem (PFP)*, and the *code writing problem (CWP)* [7]. Originally, the web-based JPLAS platform was implemented using *JSP/Java* on *Tomcat* for online use [8], and using *JavaScript* on a web browser for offline use [9]. It is noted that the offline JPLAS platform is indispensable for students in developing countries where the continuous Internet access may not be available.

We have observed that many students around the world have difficulty in programming study. The step-by-step programming study approach can be a solution for it. JPLAS offers several types of exercise problems with automatic marking functions at different levels to cover different stages of programming self-study.

To learn programming effectively, it has been suggested that students should first focus on *basic grammar* and *code reading* studies by solving a lot of simple problems for them. Then, they can move to *program coding* studies using *object-oriented programming concepts* in Java. Students are expected to solve the offered exercise problems along the above mentioned order of the problem types in JPLAS. It is designed that the difficulty level of the problem type will increase in this order.

In a lot of countries including Myanmar, Indonesia, and Bangladesh, the continuous Internet access is still not common for many students due to the high cost and the limited bandwidth. The electric supply to function the Internet is not stable. Actually, half of the world's population do not have access to the Internet. Even if available, they can be expensive, slow, and unreliable. The continuous Internet access remains luxury in developing countries. Even in developed countries such as Japan, many students are suffering from high costs of broadband connections.

To use the offline platform, students can install the necessary software in universities using USB memories if the Internet access cannot be used. Then, after solving exercise problems, students need to submit their answer files whose size is very small because of text files. Thus, they can easily submit the answer files even through the narrowband Internet. Therefore, the offline platform of JPLAS will be very effective for students who are learning programming languages at home without the Internet connections, and contribute to advance programming education.

However, the previous platform using *Tomcat* is not suitable for the offline use. Since it was implemented by Java

Manuscript received August 23, 2023; revised November, 2023.

S. T. Aung is a PhD candidate of Department of Information and Communication Systems, Okayama University, Okayama, Japan, (e-mail: soethandar@s.okayama-u.ac.jp).

L. H. Aung is a PhD candidate of Department of Information and Communication Systems, Okayama University, Okayama, Japan, (e-mail: lynnhtetaung@s.okayama-u.ac.jp).

N. Funabiki is a professor of Department of Information and Communication Systems, Okayama University, Okayama, Japan, (e-mail: funabiki@okayama-u.ac.jp).

S. Yamaguchi is a professor of Graduate School of Sciences and Technology for Innovation, Yamaguchi University, Ube, Japan, (e-mail: shingo@yamaguchi-u.ac.jp).

Y. W. Syaifudin is an associate professor of Information Technology Department, State Polytechnic of Malang, Indonesia, (e-mail: qulis@polinema.ac.id).

W.-C. Kao is a professor of Department of Electrical Engineering, National Taiwan Normal University, Taipei, Taiwan, (e-mail: jungkao@ntun.edu.tw).

and JSP, the system size is rather large. Besides, it has undergone continuous implementations and modifications by several batches of students in our group as a long-time laboratory project, where the source codes become large, complex, and difficult for students to be installed on their personal computers.

In this paper, we implement the personal learning platform for JPLAS using *Node.js* [10]. *Node.js* has gained popularity and been used in many popular web application systems [12]–[14]. The platform covers any problem type in JPLAS, while avoiding the redundancy in the implementation and improving the portability. This platform is named *NPLAS* for convenience. By adopting *Node.js* for the web application server, *JavaScript* is used on both the server and client sides. *Express.js* [11], utilized as a framework, collaborates to minimize implementation costs in connecting server programs with web pages. This combination provides uncomplicated and user-friendly implementations of functions for seamless message and file sharing between the client and the server. As a result, the code size of *NPLAS* is drastically reduced if compared with the size in the previous implementation using *Tomcat*.

The following features can be observed in this implementation that will satisfy the requirements for use in various universities around the world:

1) Open source or free software is used to reduce the costs of users.
2) No dedicated public server is required to newly install the system where the existing public servers for *Gmail* and *Google Drive* are used.
3) The system can be used without the Internet connection, and be installed using the USB memory. Thus, students can learn Java programming using this system at any place. It should be noted that in some countries, the continuous Internet connection is not realistic because of the poor infrastructure.
4) The system offers several types of exercise problems with different levels so that students can progress their Java programming studies step-by-step, from the grammar study level to the full coding level.
5) *Docker* is introduced together for deployments of *NPLAS*. *Docker* is a tool designed for executing applications and services within compact, lightweight containers, ensuring they operate independently of the installed software or configurations on the host computer [15].

The features are important as a programming learning tool for a lot of people all over the world. The cost for the software is zero by using open source. The cost for the server is zero by using the free public server. Thus, students and teachers around the world can easily install and manage *NPLAS*.

In addition, we implement the *instance file update function* for *NPLAS*, using *Angular* [16], and *Laravel* [17] as another web application system. A popular programming language often extends the grammar and introduces new libraries to enhance programming capabilities. Then, the problem files for the related exercise problems need to be updated or inserted in *NPLAS*. For this use, the *instance file update function* helps a teacher to distribute the files for the new or updated instances to the students. After receiving the files, a student can reconfigure *NPLAS* in his/her PC, using *Docker bind mount*. For the better security of the PCs of the teacher and the students, they are connected with private networks using private IP addresses. Thus, to connect them, we adopt *Google Gmail* as a reliable public cloud, to inform the file update request to the students with the email message from the teacher to the students.

Nowadays, *Python* programming becomes very popular because it is very efficient in application developments, task automations, and data analysis. Therefore, we extend *NPLAS* to *Python programming learning* where the unit testing framework *unittest* is adopted in *CWP*.

For evaluations, we asked teachers and 58 students in Okayama University, Yamaguchi University, Japan, and State Polytechnic of Malang, Indonesia, to install *NPLAS* and solve several instances on *NPLAS*. We also asked ten students to install the *instance file update function*, upload the *instance update files*, send/receive the update request message, and update *NPLAS* by downloading the *instance update files* to a folder and mounting it on the current container for *NPLAS*. All of them could successfully complete them, which confirms the efficiency and validity of the proposal.

The rest of this paper is organized as follows: Section II introduces adopted open source software. Section III presents the implementation of *NPLAS*. Section IV presents the implementation of the file distribution function. Section V presents the extension to *Python programming*. Section VI evaluates the proposal. Section VII introduces related works in literature. Section VIII concludes this paper with future works.

II. Open Source Software

In this section, we introduce the open source software that are adopted in this paper for the completeness and readability.

A. Node.js

Node.js, an open-source server environment, is compatible with multiple PC platforms such as *Windows*, *Linux*, and *Mac OS*. It serves as an interpreter and runtime environment for executing *JavaScript* source codes on the server. *Node.js* is versatile, supporting the implementation of both desktop and server applications. Consequently, developers can use the same programming language, *JavaScript*, to create both the front-end and back-end of their application systems.

B. Angular

Angular is a front-end framework for building *single-page application (SPA)* using *TypeScript* [18]. *TypeScript* is a superset of *JavaScript* and has been developed by Microsoft. It comprises a set of seamlessly integrated libraries encompassing a diverse range of functionalities, such as routing, form management, and client-server communications. *Angular* allows an intuitive user interface in a web application.

For the *instance file update function*, a simple interface is necessary to upload the new instance files to the *Laravel* back-end application. *Angular* is much simpler than the other *JavaScript* client-side frameworks. We can create more compatible and robust UI applications because it has a restricted modular-based code structure design, which arranges the code into different modules.

C. Laravel

Laravel, a server-side *PHP* framework, is open source and crafted to streamline and accelerate web application development. It boasts a wealth of built-in features, facilitating ease of use. Additionally, it supports a modular packaging system with efficient dependency management, enabling the seamless incorporation of functionalities into *Laravel* applications without the need to start from scratch. The framework is designed with the Model-View-Controller (MVC) architecture as a fundamental component. It also serves as an API back-end to the *JavaScript* single-page application.

Therefore, *Laravel* can be adopted as the API back-end for *Angular* and *Node.js* applications. Moreover, it can provide drivers for the *PHP* mail function *sendmail* using *Simple Mail Transfer Protocol (SMTP)*. *sendmail* is an internetwork email routing facility that supports many kinds of mail transfer and delivery methods for email transporting through a local or cloud-based service over the Internet.

Laravel is packed with many built-in object-oriented libraries that are full of useful features for developers. The built-in package *SwiftMailer* can be used to allow the configuration setting without specifying any specification. It is noted that for *Node.js*, an *SMTP* server needs to be set up for delivering email messages.

D. Docker Bind Mount

Docker bind mount is the process of sharing files between the *Docker image* in a Docker container and the host. It can be a local directory or a file on the host. When a file is changed on the host, the mounted file in the container is also changed automatically.

E. Express.js

Express.js, recognized as a minimalist and adaptable framework tailored for *Node.js*, demonstrates its capacity to furnish a comprehensive set of features conducive to robust web application development. In the realm of frameworks, the norm involves the provision of pre-existing components and solutions, encompassing supporting programs, compilers, code libraries, and APIs. These resources serve as invaluable assets for application development, offering the advantage of customization to expedite the development process.

Express.js can be delineated as an abstraction layer constructed atop *Node.js*, facilitating the streamlined management of servers and routes [19]. This framework encompasses a robust suite of features essential for the development of web and mobile applications, encompassing the following functionalities:

- It is applicable for the development of web applications, encompassing single-page, multi-page, and hybrid designs.
- It enables the configuration of middleware to handle responses to HTTP Requests.
- It articulates a routing table that directs varied actions in accordance with the HTTP method and URL.
- It enables the dynamic rendering of HTML pages by passing arguments to templates.

F. Embedded JavaScript (EJS)

The user interface is constructed utilizing Embedded *Embedded JavaScript (EJS)*, *Bootstrap* files, and *CSS* with management orchestrated by *Node.js* and *Express.js* programs. *EJS* serves as a template engine, facilitating the rendering of JavaScript codes on the client-side. Essentially, *EJS* is employed to embed *JavaScript* codes within *HTML* codes, a practice commonly applied when utilizing *Node.js* within the *Express.js* framework.

III. Implementation of *NPLAS*

In this section, we present the implementation of *NPLAS*.

A. Overview of NPLAS

NPLAS is a web application system designed to enable educators to furnish programming exercises to numerous students while overseeing their learning activities on the server. In the server architecture depicted in Figure 1, *Node.js* is employed as the web application server, complemented by the *Express.js* framework. In realizing the *MVC model*-based application, *Java* is utilized for the *model (M)*, executing *JUnit* [20] for testing answer source codes in code-writing problems [7]. The *view (V)* is constructed using *EJS, CSS, JavaScript*, while *JavaScript* is employed for the *controller (C)*. Notably, no database system is incorporated for data management.

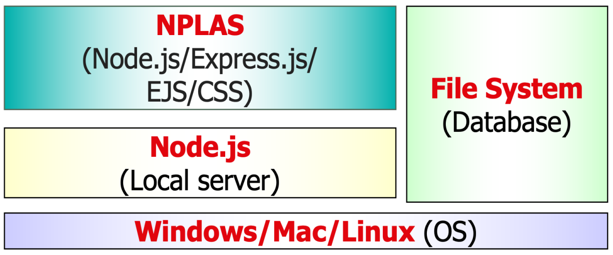


Fig. 1: *NPLAS* server platform.

B. Server-Side Implementation

Fundamentally, *Node.js* exhibits an intricate structure, posing challenges in its utilization. Hence, in our implementation, *Express.js* is employed in conjunction by installing the *node package manager (npm)* from the provided binary packages tailored for each operating system.

npm plays a pivotal role in managing *Node.js* applications, offering a repository of reusable *JavaScript* libraries. Additionally, it serves as a tool for executing tests and facilitating various development processes. The application environment incorporates essential dependencies, including frameworks and template engines, through the use of *npm*.

Within *Node.js/Express.js*, each web application initiates and operates its individual web server. *Express.js* furnishes methods to designate the function called for a specific *HTTP* verb (GET, POST, SET, etc.) and *URL* pattern, referred to as a *"route"*. It also allows the specification of the template engine for the *"view"*, determining the location of template files and the specific template used to render a response.

C. File for Connection between Server and Browser

Upon a client navigating to the server through the domain name *(URL)* and port, such as *localhost:4000*, via the browser, the web application executes the requested procedure in accordance with the roles outlined in Figure 2.

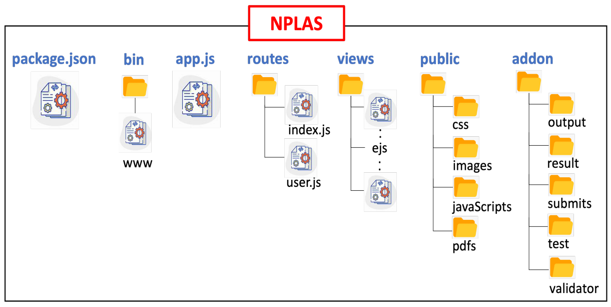


Fig. 2: *NPLAS* application directory structure using *Express.js*.

- *package.json*: The application invokes the *package.json* file, which meticulously enumerates all dependencies for the specific JavaScript "package". This comprehensive listing encompasses 1) package's name, 2) version, 3) description, 4) initial executable file, 5) production dependencies, 6) development dependencies, and 7) compatible version of *Node.js*. The *package.json* file contains all the necessary information for *npm* to fetch and execute the application. By identifying the *"start"* key and the value *"node ./bin/www"* in the script tag, it signifies that the *Node.js* project references the file named *"www"* within the bin folder. This file gathers data for *Express.js* to utilize within the application.
- *www file*: The *www* file defines the entry point of the application and accommodates various setup configurations. Within this application, three distinct scripts—namely, *app*, *debug*, and *http*—are configured in this file.
- *app.js*: The *app.js* will declare the package that is required by the application globally and will be the main root file of the whole application directory.
- *index.js*: The *index.js* file under the *routes* folder will route the application's requests to its appropriate controller and then render the associated view.
- *views*: The *views* folder includes all the *NPLAS* user interface files that will be displayed to the browser by using *EJS*.
- *public*: All the static files, such as the *CSS*, images, and *JavaScript* files, are set up under the *public* folder.
- *addon*: The customized *addon* folder serves as the database, encompassing the test code files, answer source code files, and validator files.

Here, the *user.js* route was a result of it being part of the default structure provided by *Express.js* during the project initialization. It is noted that the current backend does not require the user route, and the authentication and authorization features are not implemented. As this application focuses on aspects of programming learning through the personal use, we have not implemented the user management functionality at this stage. Details were not described on the *user.js* route.

D. MVC Model Software Architecture in NPLAS

The proposed software architecture for the *NPLAS* platform adheres to the *MVC model*, recognized as the standard architecture for a web application system, as illustrated in Figure 3.

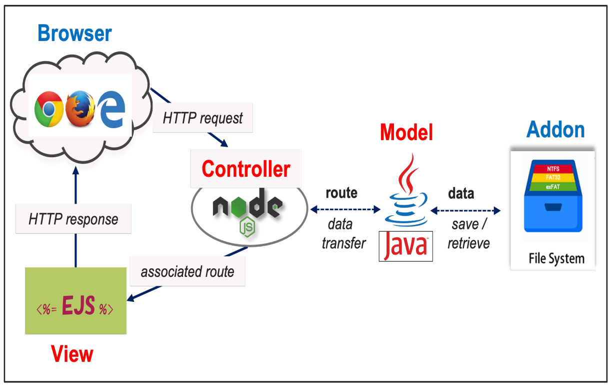


Fig. 3: MVC model in *NPLAS* for *Java*.

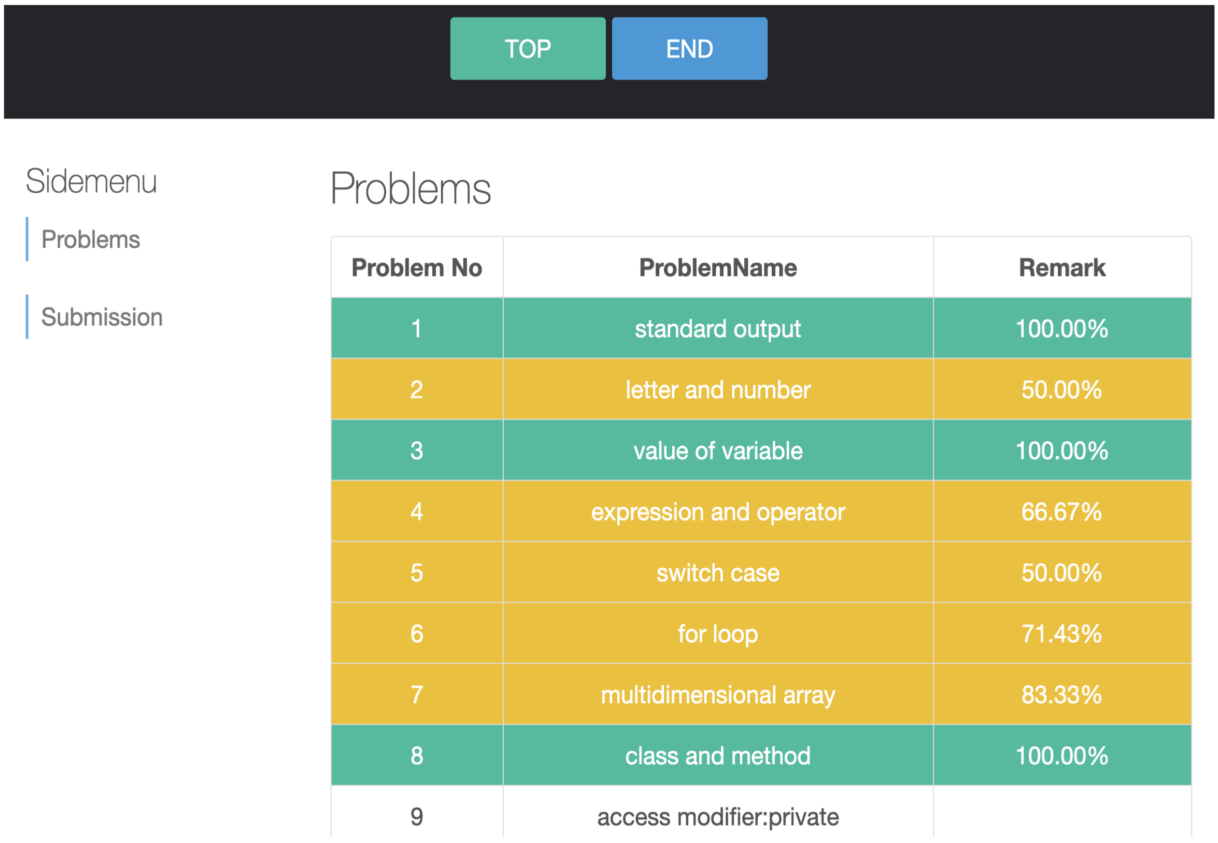
1) Model: The *model* component reads essential data files from the designated file system named *"addon"*. These files encompass the test code files, answer source files, and validator files.

In *NPLAS*, the marking function varies based on the problem type. For the *code writing problem*, the background execution of the answer code testing is facilitated by a function implemented in *Java* using *Eclipse*. This function is exported as a *JAR* file and imported into *Node.js*. It autonomously tests any submitted answer source code from the browser by executing the provided *test code* on *JUnit*. The outcomes, inclusive of *JUnit* logs, are documented in the file system and are accessible for student viewing through the browser.

For other problem types within *NPLAS*, the marking function is embedded in the *JavaScript* program that operates within the browser as students solve problems and submit their answers. This function undertakes a comparison between the submitted answer and the predefined correct answer. Temporarily, the answers and marking results are stored in the browser's local storage, identified by a *unique* key. When students download this information to a text file for submission to the teacher, the *JavaScript* program stores the data in the local storage within a text file in the output folder located under the *"addon"* directory of the application.

2) View: The *view* component in *NPLAS* actualizes the user interface through the utilization of the *EJS* template engine, enabling the rendering of dynamic content in the browser. *EJS* constructs *HTML* code incorporating *JavaScript* codes transmitted through the back-end of the application. In this architectural design, the consistent elements of the interface are generated using *EJS* and *SkyBlue CSS*, while the adaptable components are created through *JavaScript* functions stored in the public directory. Depending on the client's requested route, the main body is dynamically replaced with the pertinent *EJS* file, a strategy aimed at minimizing code size and streamlining the code architecture.

3) Controller: The *controller* in *NPLAS* is implemented using *JavaScript*. Upon receiving a request, the application determines the necessary action based on the *URL* pattern



Problems

Problem No	ProblemName	Remark
1	standard output	100.00%
2	letter and number	50.00%
3	value of variable	100.00%
4	expression and operator	66.67%
5	switch case	50.00%
6	for loop	71.43%
7	multidimensional array	83.33%
8	class and method	100.00%
9	access modifier:private	

Fig. 4: Problem list interface with correct answer rates.

and the pertinent information contained in *POST* or *GET* data. This process may encompass reading or writing information to the file system, or undertaking the requisite tasks to fulfill the request.

Here, *Express.js* facilitates the transfer of data through the corresponding route. In the process of data transfer, the appropriate name is assigned within the route. Subsequently, the application responds to the web browser, often by dynamically generating an *EJS* page. This page is rendered by the browser to display the view associated with the assigned name, incorporating the retrieved data into placeholders within the *EJS* template. *Node.js* and *Express.js* are well-equipped for running a website with dynamic data. Moreover, the structure incorporates hierarchical arrangements and grouping concepts, accessible to individuals proficient in *Node.js* and *Express.js*.

E. Introduction New Features in NPLAS

We implement five new features in *NPLAS* to help students to solve the assigned problems. The first feature is to show the correct answer rate. The generator calculates the number of questions or input forms in each instance, and writes it in the output HTML file. Then, it adds the *JavaScript* program that counts the number of correct answers for each instance by the student, and calculates and shows the correct answer rate by dividing the number of correct answers by the number of questions when the problem list page is selected. To distinguish the instance, the different key is used for storing the answer marking results for each instance in the web storage. The key is also stored in the local storage.

Figure 4 illustrates the list of the problem instances to be solved within this category. This page displays the correct answer rate for each problem instance, aiming to encourage students to solve all instances accurately.

The second one is the instance transition buttons. By clicking either button, the page for the next instance or for the previous instance can be shown.

The third one is to record the last answer in each blank in the web storage so that the student can restart solving the instance from his/her previous answers.

The fourth one is to automatically adjust the input form size for each blank size by the number of characters in the correct answer.

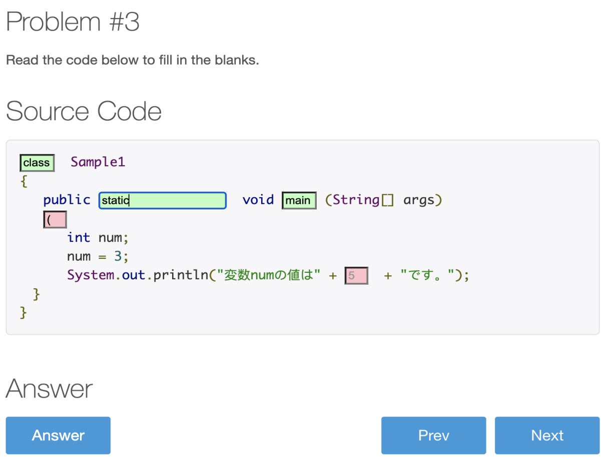
The final one is the background color alterations in the input form. It changes to green when the answer by the student is correct and turns red otherwise.

Figure 5 shows the problem answer page example.

F. Automated Sub-Folder Creation in NPLAS

In order to enhance file organization within the *NPLAS* system, we have implemented a sub-folder structure to improve the management and accessibility of answer files submitted by students. To achieve this, we developed a program that dynamically creates sub-folders based on the problem type. When a user submits an answer through the browser, the program automatically generates a sub-folder within the main directory, named according to the specific problem type associated with the answer. This approach ensures a logical and intuitive arrangement of files, providing a more organized and structured approach to saving and accessing text files in *NPLAS*.

The program we developed plays a crucial role in ensuring efficient file organization within *NPLAS*. When a student



Problem #3

Read the code below to fill in the blanks.

Source Code

```
class  Sample1
{
   public static        void main (String[] args)
   (
      int num;
      num = 3;
      System.out.println("変数numの値は" + 5  + "です。");
  }
}
```

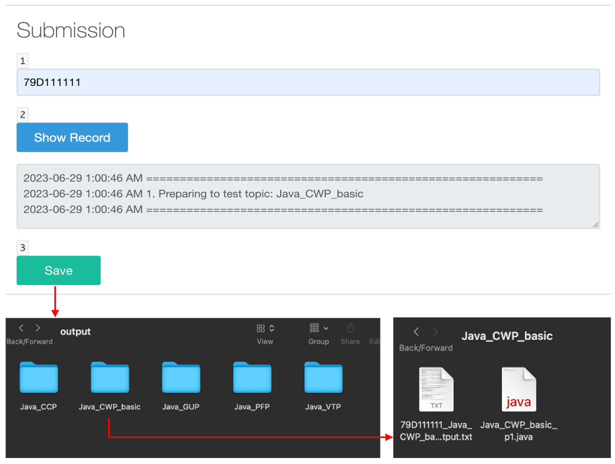
Fig. 5: *EFP* problem answer page.

submits an answer on the browser, the system automatically creates the necessary sub-folders based on the problem type, using the *"mkdirSync"* function to create the sub-folder if it does not exist. This automated process eliminates the need for manual folder creation and ensures a consistent and standardized file structure.

Subsequently, the program saves the output text files within the corresponding sub-folders. By utilizing the *"writeFileSync"* function and providing the appropriate path, which combines the sub-folder name and the related text file name, the output files are efficiently stored in their respective sub-folders. This clear and organized structure enables easy navigation and retrieval of specific files based on problem types.

The enhanced file organization provides users, including students and teachers, with a structured and systematic approach to managing their files. It eliminates the issue of having numerous files scattered in a single folder, making it easier to locate, access, and work with the specific files related to each problem type in *NPLAS*.

Figure 6 shows the output text file corresponding to the sub-folder example.



Submission

1
79D111111

2

```
2023-06-29 1:00:46 AM ============================================================
2023-06-29 1:00:46 AM 1. Preparing to test topic: Java_CWP_basic
2023-06-29 1:00:46 AM ============================================================
```

3

Fig. 6: Output text file in sub-folder.

G. Adoption of Docker for NPLAS

With its previously mentioned advantages, *Docker* is adopted to help students to install the *NPLAS* platform. Thus, there is no compatibility issue happening across users. Figure 7 depicts the process of employing Docker to install the *NPLAS* platform in a PC.

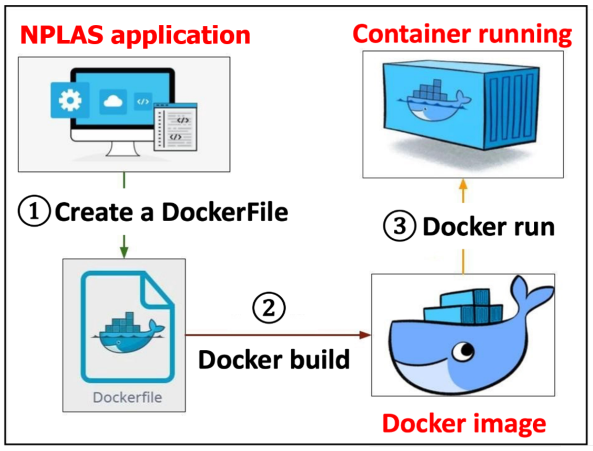


Fig. 7: Installation procedure of *NPLAS* platform using *Docker*.

A *Docker container* can be generated using a plain text file named *Dockerfile*, providing both human-readable and machine-readable instructions for crafting computing environments and interacting with data. The *Docker container* operates independently anywhere, provided *Docker* is installed, and is constructed from the *Docker image* to provide the necessary software environment for executing the target application. To facilitate easy distribution of the *NPLAS Docker image* to students, it is stored in the *Docker Hub* [21] account using the *Docker "push"* command, as illustrated in Figure 8.

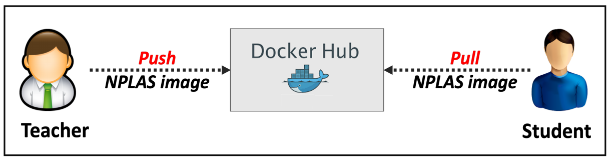


Fig. 8: Upload and download *NPLAS* image from *Docker Hub*.

H. Usage Procedure

To utilize the *NPLAS* platform, students are required to execute the following steps:

1) Download and install *Docker* corresponding to the student's PC operating system. For *Windows OS*, the installation of *Windows Subsystem for Linux (WSL)* is also necessary.
2) After initiating *Docker* on the PC, download the *NPLAS Docker image* from *Docker Hub* using the *Docker pull* command.
3) Execute the *Docker run* command to operate the image on the PC. The specifics of this command may vary based on the OS of the PC. Following the execution of this command, the student should examine the *"output"* folder to store the answer files.
4) Access the *NPLAS* platform by opening the browser and entering *localhost:4000* in the address bar.

Here, the *Docker image* size of *NPLAS* is $578MB$, which is considered large due to two main reasons. The **first reason** is that the size is influenced by the "openjdk:16-slim-buster" Docker image, which includes the Java Development Kit (JDK) and the slim version of the Debian Linux operating system. This image is larger because it contains the necessary components and libraries for Java execution. However, it is a smaller variant of the full JDK image, excluding non-essential components.

Since our *NPLAS* platform integrates with this JDK image, the resulting Docker image size is affected. It is note that Docker image sizes may change over time as new versions and updates are released. Currently, we chose "openjdk:16-slim-buster" for specific reasons. It offers compatibility with our software packages and libraries, has a larger package repository, and has benefits from robust security updates.

The **second reason** is that our Docker image incorporates the "openjdk:16-slim-buster" base image, along with the additional commands used to install Node.js and npm. The commands ensure that Node.js and npm are installed in the Docker image, allowing for the execution and management of Node.js applications within the container. Although these commands add some size, they are relatively small compared to the base image.

Therefore, the primary factor influencing the image size is the choice of the base image. Thus, the *NPLAS docker image* may take a few minutes to download it. After downloading the image one time, the student can solve the exercise problems in *NPLAS* at any time and any place without the Internet connection.

IV. Implementation of Instance File Update Function

In this section, we present the implementation of the *instance file update function* for *NPLAS* as another web application system.

A. Software Architecture

Figure 9 shows the overview of the *instance file update function*. *Linux* is adopted for the operating system in the server platform. For the client side, we adopted *Angular* to create the interface for uploading the required instance files on the browser. For the server-side, *PHP* is used as a web application server together with the *Laravel* framework. *RestAPI* is adopted to connect between the client-side and server-side. Then, the *Simple Mail Transfer Protocol (SMTP)* is used to send updated instance files to the student. Instead of using database for managing the data, we used the built-in *Laravel* storage.

1) RestAPI: RestAPI (Representational State Transfer Application Programming Interface) [22] is a widely adopted architectural style and API framework that enables seamless interactions with Restful web services. It provides developers with a high level of flexibility, allowing them to design and implement APIs that can handle various types of calls and respond with data in different formats. One of the key advantages of *RestAPI* is the lightweight nature, as it does not necessitate the installation of additional software or libraries. This simplicity allows easier adoptions and integrations into

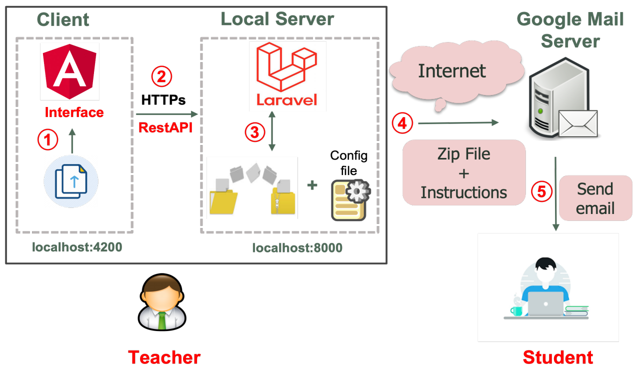


Fig. 9: Overview of instance file update function.

existing systems, reducing the overall complexity of development and deployment processes.

It can be utilized over almost any protocol, leveraging the capabilities of *HTTP*. In RestAPI, all the standard *HTTP* methods can be employed to perform operations on resources. The *GET* method is used to retrieve data or records from the server, providing a means to access information. The *POST* method is employed to create new data or records on the server, enabling the submission of data to be stored or processed. The *PUT* method is utilized to update existing data or records on the server, allowing for modifications or changes. Lastly, the *DELETE* method is used to remove data or records from the server, providing a mechanism for data deletion.

B. Client-Side Implementation

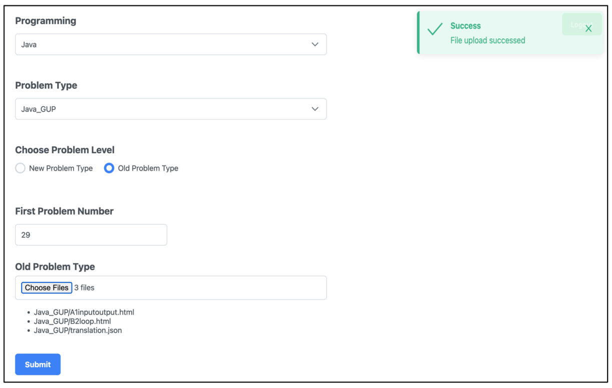
In the client-side implementation, *Angular13* and *PrimeNG* are adopted. *PrimeNG* is the *CSS framework* specific to *Angular*.

To use *Angular*, we have to install *Node.js* that allows developers to write JavaScript codes that run on the OS of the computer instead of a browser. *Angular CLI* is used as the *ng command-line interface* tool to initialize, develop, scaffold, and maintain *Angular* applications from a command shell.

1) Exercise Creation and Update Functionality in NPLAS for Teachers: The teacher can create a new exercise/instance by following the set of guidelines or templates provided by the *NPLAS*. The number of files required may vary depending on the complexity of the exercise, but typically it involves creating source code files and additional resources or instructions necessary for the exercise. In terms of the expected structure for these files, it is recommended to organize them in a logical manner that facilitates easy understanding and navigation. For example, using appropriate naming conventions, and providing clear instructions within the exercise files can enhance the overall user experience.

The function is designed to facilitate the update or modification of exercise problems by the teacher and the subsequent distributions of these updated problems to the students' PCs via a local server and Gmail. The intention behind implementing the function is to support managing exercise updates within the *NPLAS* system, particularly when using the tool as part of an assignment course in a university setting.

2) User Interface: Figure 10 shows the interface for the instance file update function. It includes the input type of the instance file, and the button to run the application using *ng* command and call the application on the browser with the domain name (*URL*) and the port such as *localhost:4200*. The input type in the form as the requirements of the instance file as the following procedures.



Programming

Java

Problem Type

Java_GUP

Choose Problem Level

New Problem Type [✓ Old Problem Type]

First Problem Number

29

Old Problem Type

Choose Files 3 files

- Java_GUP/A1inputoutput.html
- Java_GUP/B2loop.html
- Java_GUP/translation.json

Success

File upload successed

Fig. 10: Interface for instance file update function.

1) Choose the programming language type. Then, we will see the problem type list depending on the programming type we chose.
2) Choose the problem type.
3) Choose the problem level. The old ones will perform the instance file update or insert separately, and the new ones will both update and insert instance files concurrently.
4) Assign the problem number that we want to update or insert.
5) Upload the instance files. After uploading them, the system will create a problem folder for each instance file in order as the *NPLAS* application requires.
6) After getting the required input types in the form, we will submit the instance file data to the back-end application, *Laravel*, as a request through *RestAPI*.
7) If the response from the back-end is successful, it will show a successful alert. Otherwise, it will show with a failed alert.

C. Server-Side Implementation

In the server-side implementation, the *Laravel PHP* framework is adopted. *PHP* [23] is first installed. Then, *Composer* [24] is installed to manage *Laravel* dependencies. After that, *composer.json* is used as a collection of *PHP* libraries that are necessary to run the application using *artisan* command and call the application on the browser with the domain name (*URL*) using the port such as *localhost:8000*.

Then, the server application will perform the following steps:

1) Accept the instance file from the client as a request.
2) Filter the received data and store them into the *Laravel* storage.
3) Compress the files into the zip file when multiple files are received from the client.
4) Send this zip file into the mail server using *Simple Mail Transfer Protocol (SMTP)*, after preparing the mail configuration setting and the instruction for students.
5) Send the instance file together with the instruction messages to the students.

D. Connection between Server-Side and Client-Side

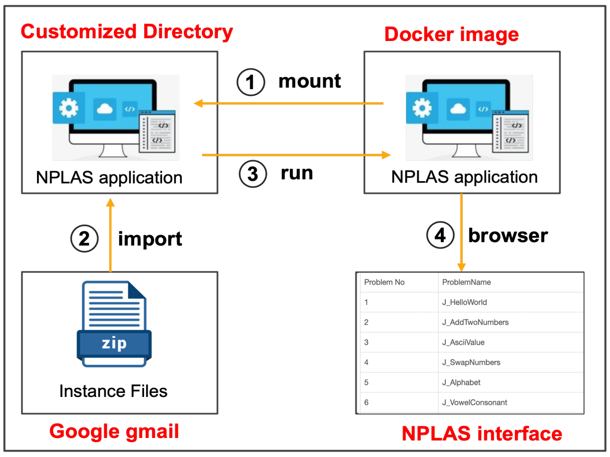
When a client navigates to the domain name (*URL*) using the port such as *localhost:4200*, the client sends the request data to the server that includes the instance file by uploading it through the *Submit* button on the browser. At the same time, to receive the request and respond to the data, the server navigates to the domain name (*URL*) using the port such as *localhost:8000*. Then, the following procedure is performed:

1) Generate the *RestAPI* for sending the data at *Angular*.
2) Store the data after receiving the data via the *RestAPI* from the client-side at *Laravel*.
3) Combine the *storing data* and the *SMTP configuration file* to connect the mail server.
4) Send them together to the students by email.

E. Docker Bind Mount

Using the *Docker bind mount* features, the students update *Docker image* for *NPLAS* by changing the folder containing the existing files by the folder for the received files in the file system.

Figure 11 illustrates the workflow of the *NPLAS* application using *Docker bind mount*. First, the teacher sends the updated or new instance files to the student via email. Second, the student downloads them and creates a new folder to copy the application from the Docker image in the container. Third, the created new folder mounts with the *NPLAS* application in the container and imports the downloaded instance files into the application as the instruction in the email. Finally, the student can access to the *NPLAS* application by calling *localhost:4000* and seeing the *NPLAS* platform with the new instance files.



Problem No	ProblemName
1	J_HelloWorld
2	J_AddTwoNumbers
3	J_AsciiValue
4	J_SwapNumbers
5	J_Alphabet
6	J_VowelConsonant

Fig. 11: *NPLAS* application workflow with *Docker* bind mount.

F. System Installation by Student

After receiving the email from the teacher, the student needs to perform the following procedure, described in the instruction in the email, to access the updated instance files into his/her previous *NPLAS* application.

1) Download the instance files from the received email.
2) Run the Docker image using *Docker run* command.
3) Get the Docker container ID of the running image using *Docker ps* command.
4) Mount the *NPLAS* application from the Docker image in the container into the customized directory path using *Docker cp* command. After that, the students can see the *NPLAS* application on the directory path that they copied. Here, there are two *NPLAS* applications on the student's PC. The first one is in the created folder in the host and another is in the Docker image in the container.
5) Import the downloaded instance file into the copied *NPLAS* application.
6) Kill the previous container ID to run the new image using *Docker kill* command.
7) Run the new image using *docker run* command.
8) Open the browser and call the *localhost:4000*. Then, the student will see the *NPLAS* platform with the updated instance files on the browser.

V. Extension to Python Programming

In this section, we present the extension of the *NPLAS* platform to *Python programming*.

A. Prerequisites for NPLAS platform to Python

In this paper, we present the implementation of the *NPLAS* platform for Java programming study, including the distributions using *Docker*, and the *instance file update function* for updating the *Docker image*. The extension to Python programming study is the straightforward where the modifications are small. Actually, in the platform, only the implementation for the *code writing problem (CWP)* needs to be changed, because the testing function with the test code and the student source code runs on Python, and the Python testing tool called *unittest* [25] is used instead of *JUnit* for Java in the marking function.

1) Problem Types: It offers a range of exercise problems designed to provide to students at different learning levels. These include the grammar-concept understanding problem (GUP), the value trace problem (VTP), the Code Modification Problem (CMP), the element fill-in-blank problem (EFP), and the code writing problem (CWP). To ensure comprehensive coverage of Python programming concepts, we conducted an analysis and selection of sample source codes from various programming websites on the internet for each problem type. These source codes were carefully chosen to exemplify different aspects of Python programming. Using our generators, we transformed these selected source codes into exercise problems within the *NPLAS* platform. These exercise problems were then installed and integrated into the system.

2) Unit Testing Framework: In the *NPLAS* platform, we employ the *unittest* framework [25], an integrated unit testing framework in Python, to assess provided source codes. This framework streamlines automated unit testing through the execution of test codes. Leveraging the *unittest* framework involves importing the *unittest* module and constructing a testing class, extending the *TestCase* class provided by the framework. Each test method within the code compares the execution result of the source code with the anticipated result. If they match, the test is considered passed; otherwise, it is marked as failed.

3) Test Code: For the code writing problem (CWP) instance in an assignment, the teacher needs to prepare the test code beforehand. This test code is used to validate whether the student's answer source code meets the required specifications. It should be written using the unittest library, allowing it to clearly present the assignment's specifications to the student. By providing clear explanations and well-formed sentences within the test code, the student can understand the expected requirements more effectively.

4) Multi-Language Support in NPLAS: Java and Python Integration: The *NPLAS* platform is designed to support multiple programming languages, including *Java* and *Python*. Students can choose their preferred language for solving programming problems within the platform. This support of multiple languages is achieved through the integration of language-specific plugins or modules, enabling users to work with their language of choice.

NPLAS allows the simultaneous use of both languages within the same platform. In this paper, we presented *Python* as a separate version of the tool to be provided to a specific programming language course in a university. This separation allows us to provide targeted support and customization for each language.

However, it is important to note that for learners who wish to study both programming languages using the tool as a self-study resource, we will provide access to both *Java* and *Python* within the same version of the platform. This means that learners can switch between languages and explore programming problems in both *Java* and *Python* without the need for separate versions or installations.

B. Software Architecture

Figure 12 shows the *MVC model* of this extension. Instead of *Java*, *Python* is used in the *model (M)* to run *unittest*. The marking function for CWP is also implemented by *Python*. Automatically evaluating any answer source code submitted from the browser, this function executes the corresponding *test code* using *unittest*. The outcomes, along with logs, are documented in the result file, accessible for student viewing through the browser.

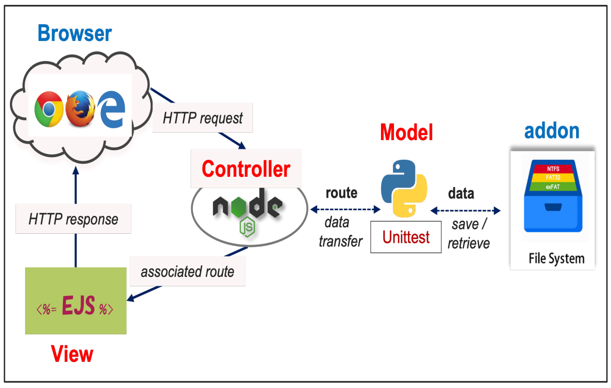
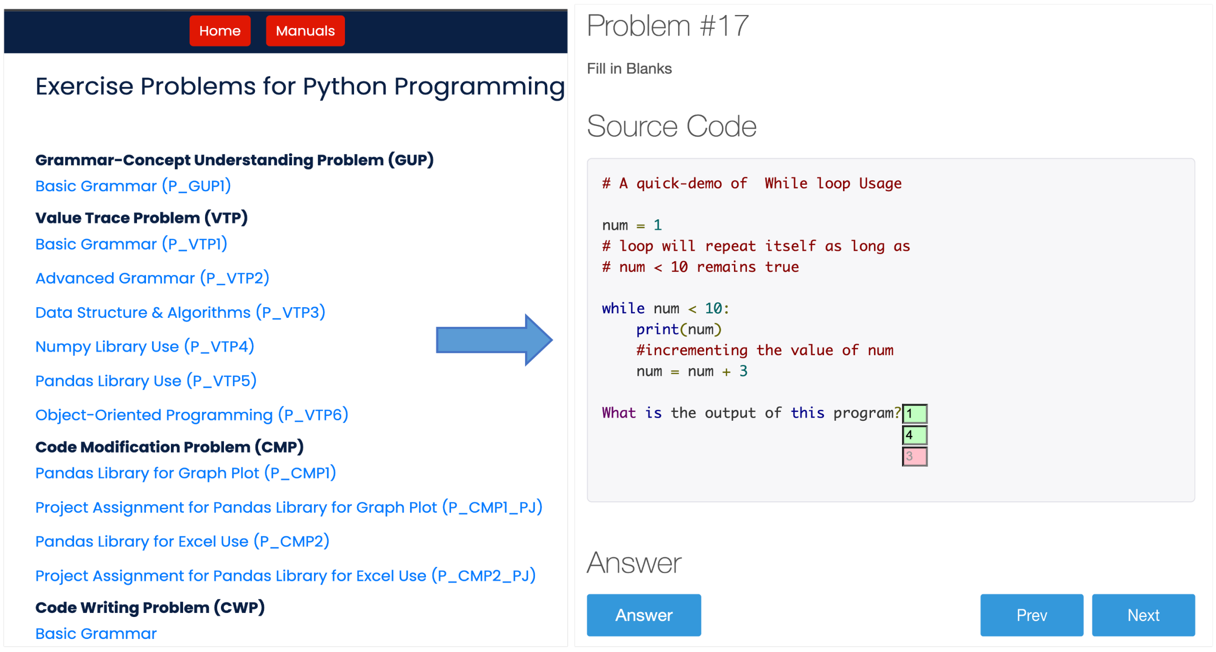


Fig. 12: MVC model in *NPLAS* for *Python*.

C. Problem Answer Interface

Figure 13 shows the answer interface.



Exercise Problems for Python Programming

Grammar-Concept Understanding Problem (GUP)
Basic Grammar (P_GUP1)
Value Trace Problem (VTP)
Basic Grammar (P_VTP1)
Advanced Grammar (P_VTP2)
Data Structure & Algoritms (P_VTP3)
Numpy Library Use (P_VTP4)
Pandas Library Use (P_VTP5)
Object-Oriented Programming (P_VTP6)
Code Modification Problem (CMP)
Pandas Library for Graph Plot (P_CMP1)
Project Assignment for Pandas Library for Graph Plot (P_CMP1_PJ)
Pandas Library for Excel Use (P_CMP2)
Project Assignment for Pandas Library for Excel Use (P_CMP2_PJ)
Code Writing Problem (CWP)
Basic Grammar

Problem #17
Fill in Blanks
Source Code

```
# A quick-demo of  While loop Usage

num = 1
# loop will repeat itself as long as
# num < 10 remains true

while num < 10:
    print(num)
    #incrementing the value of num
    num = num + 3

What is the output of this program?1
                                   4
                                   3
```

Answer

Fig. 13: Problem list and *VTP* answer interface.

TABLE I: Number of code files and total size in *NPLAS* platform implementations.

file extension	previous	proposal
js	301	57
html/ejs	38	24
css	21	8
total	**360 (17.2 MB)**	**89 (12.4 MB)**

VI. Evaluation

In this section, we evaluate the efficiency and validity of the implemented *NPLAS* platform and the *instance file update function*.

A. Comparison of Two Implementations

First, we compare the number of source code files and the total size between the previous implementation on *Tomcat* and this implementation on *Node.js*. Table I shows the number of code files and the total file size, where the problem instance files are excluded. It indicates a substantial reduction in both aspects within this platform. The utilization of *Node.js*, *Express.js*, and *EJS* facilitates straightforward and uncomplicated implementations of functions for sharing messages and files between the client and the server in the web application.

B. Evaluation of NPLAS Installation and Operation

Next, we evaluate the installation and operations of *NPLAS*.

1) Setup: For the installation of *NPLAS*, we prepared the manual that explains how to install and use it. It includes *Docker installation*, *Windows Subsystem for Linux (WSL) installation*, and *NPLAS system downloading from Docker Hub*. Here, they will download the *NPLAS* system from the *Docker Hub*. In the manual, we recommended to use Linux or Windows 11 to avoid errors in the Docker installation.

2) Results: Then, we asked the *Java* and *Python* programming teachers and the total of 58 students in Okayama and Yamaguchi Universities, Japan, as well as the State Polytechnic of Malang, Indonesia, to install *NPLAS* and solve several instances on it by following the manual. Among them, 24 are graduate and undergraduate students in Japan who have installed the *NPLAS* platform for *Java*. They have used the previous JPLAS before. The remaining 34 are undergraduate students who have took the Python programming course in Indonesia to install the *NPLAS* platform for *Python*, but have not used the previous one. After they successfully completed the installation, we asked them 12 questions as the questionnaire in Table II with the five grades (1: strongly disagree, 2: disagree, 3: neutral, 4: agree, 5: strongly agree) to collect their feedback on the proposal. We also provide a detailed explanation of each question, the hypotheses, and the measurements expected in Table III.

Figure 14 shows the answer distribution to each question from the students. As the summary, Table IV shows the percentage of each grade answer on *NPLAS* and on the *instance file update function*. Most of the students were satisfied with this platform. Thus, the validity and efficiency of *NPLAS* are confirmed. However, a few students may not be fully satisfied with the current implementation of *NPLAS*, since they replied 1 to Q6 and Q8. We will examine their opinions in details and continue to study improvements of *NPLAS* in future works. Regarding Q11, if 34 students have never used the previous tool, their responses may not be directly applicable to evaluating the improvements of NPLAS over the previous JPLAS system. It would be more appropriate to consider the responses from participants who have prior experiences with the previous JPLAS system for the comprehensive analysis of the comparison. This limitation should be acknowledged and discussed, emphasizing that the responses of the 34 students may not contribute directly to assessing the improvement.

Figure 15 shows the *SUS (System Usability Scale) score* on the usability level that is calculated from the questionnaire

TABLE II: Questions for questionnaire on *NPLAS*.

no.	question
Q1	Is it easy for you to solve programming using NPLAS?
Q2	Do you think that NPLAS is useful for people who are studying programming?
Q3	Are the instructions in the manual file of NPLAS clear?
Q4	Is the installation process you did for using NPLAS easy for you?
Q5	Do you think that the percentage calculation for each problem in NPLAS is accurate?
Q6	Do you feel that showing your score percentage for each problem in NPLAS motivated you to solve the problems again?
Q7	Do you think that your answers remaining in the blanks are better than removing them after solving the problem?
Q8	Do you feel that you want to solve the problem already corrected again?
Q9	Do you feel that your programming study is improved after solving the problems in NPLAS?
Q10	Are you satisfied with NPLAS?
Q11	If you have used the previous JPLAS system, do you feel that this NPLAS is better than the previous ones to use?
Q12	How many rate do you want to give the NPLAS system overall?

TABLE III: Hypothesis, and Measurement for questionnaire on *NPLAS*.

no.	hypothesis	measurement
Q1	The NPLAS system is designed to make programming problem-solving easier.	Assessing participants' perceived ease of solving programming problems using NPLAS.
Q2	NPLAS has value as a learning tool for individuals studying programming.	Evaluating participants' perceptions of the usefulness of N-PLAS in the context of programming education.
Q3	The instructional materials provided with NPLAS are clear and easy to understand.	Gauging participants' assessments of the clarity and comprehensibility of the instructions in the NPLAS manual.
Q4	The installation process for NPLAS is user-friendly and straightforward.	Assessing participants' perceptions of the ease of installing and setting up NPLAS on their systems.
Q5	The percentage calculation mechanism in NPLAS provides accurate results.	Evaluating participants' opinions regarding the accuracy of the percentage calculation feature in NPLAS.
Q6	The display of score percentages in NPLAS acts as a motivational factor for participants.	Assessing participants' perceptions of the motivational impact of score percentages in NPLAS.
Q7	Allowing participants to keep their answers in the blanks after problem-solving has benefits.	Evaluating participants' preferences regarding retaining the answers in the blanks after solving problems.
Q8	Participants may have the desire to reattempt problems they have already solved and received corrections for.	Assessing participants' inclinations to revisit and rework previously solved problems in NPLAS.
Q9	Engaging with programming problems in NPLAS contributes to improvements in participants' programming skills.	Evaluating participants' perceptions of the impact of solving problems in NPLAS on their programming study.
Q10	Overall, participants express satisfaction with the NPLAS system.	Assessing participants' level of satisfaction with NPLAS as a programming learning tool.
Q11	Participants who have used the previous JPLAS system may perceive NPLAS as an improved system.	Gathering feedback from participants who have experiences with both systems to evaluate their preferences.
Q12	Participants have overall impressions of the NPLAS system that can be quantified by assigning ratings.	Collecting numerical ratings from participants to gauge their overall assessments of the NPLAS system.

TABLE IV: Summary of questionnaire answers on *NPLAS* and *instance file update function*.

	1	2	3	4	5
NPLAS	1%	2%	16%	45%	36%
Update	0%	0%	16%	38%	46%

TABLE V: Interpretation of *System Usability Scale (SUS)* score.

SUS score	grade	adjective rating
>80.3	A	Excellent
68-80.3	B	Good
68	C	Okay
51-68	D	Poor
<51	F	Awful

results on *NPLAS* [26]. Table V shows the general guidelines for the interpretation of the *SUS score* in [27]. From Figure 15, the scores of 32 students among 58 (55.2%) are *Excellent (Grade A)*, the scores of 15 students (25.9%) are *Good (Grade B)*, the scores of three students (5.2%) are *Okay (Grade C)*, the scores of six students (10.3%) are *Poor (Grade D)*, and the scores of two students (3.4%) are *Awful (Grade F)*, respectively. It means that 81% students among 58 are satisfied with *NPLAS* in the usability by responding either *Excellent (Grade A)* or *Good (Grade B)*.

C. Evaluation of Instance File Update Function Installation and Operation

Next, we evaluate the installation and operations of the *instance file update function*.

1) Setup: For the installation of *instance file update function*, we prepared the manual that explains how to import/update the new instance files in *NPLAS* from the email using the *Docker bind command*. Also, we made the *script file* to run the instructions step by step automatically, except for downloading the problem file from the browser, which is easy and small work for students. Our experiment results show that all the students could easily complete updating the files in *NPLAS* on their PCs.

For the teacher side, he/she does not require to generate the *Docker* images repeatedly. The teacher just needs to make the files for the new instances and run the implemented *script file* to deliver them to the students. However, we will further improve the system architecture and the script file in our future works to automate all the procedures.

2) Results: Then, we asked one teacher and 10 students in the same university to install the *instance file update function* and solve some instances on it by following the manual. At this installation, two students had difficulties. Thus, we prepared the script file to run the instructions automatically. Then, all of them successfully completed the

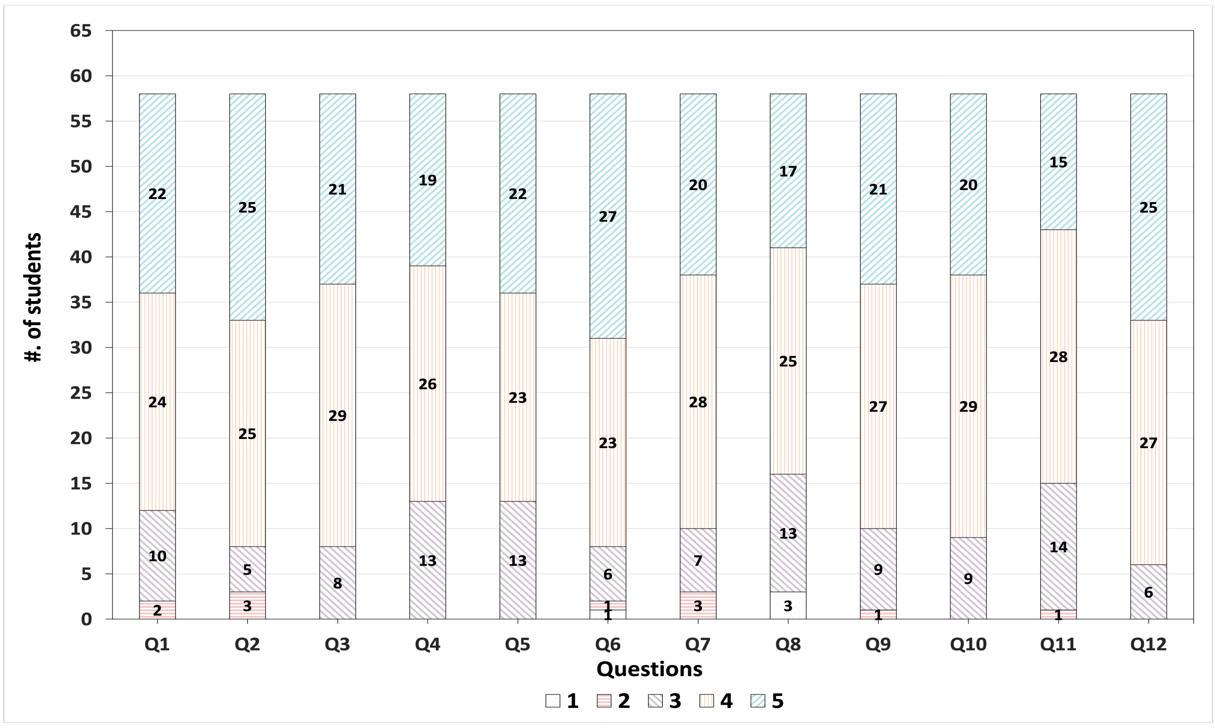


Fig. 14: Answers for questionnaire on *NPLAS*.

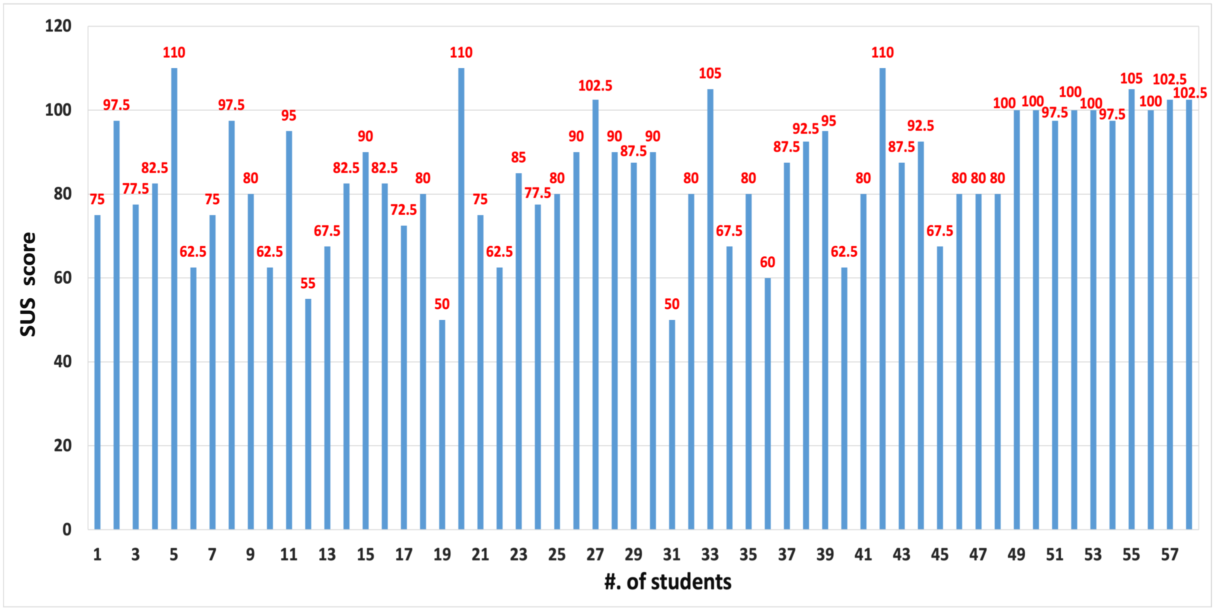


Fig. 15: SUS scale on *NPLAS* usability by each student.

installation. After that, we asked the students to answer the four questionnaire questions in Table VI with the five grades.

Figure 16 shows their answers. Like in the results in Figure 14, most of the students were satisfied with this function. As the summary, Table IV shows the percentage of each grade answer on the *instance file update function* and on *NPLAS*.

Figure 17 shows the *SUS scale* on the usability level that are calculated from the questionnaire results on the *instance file update function*. From Table V and Figure 17, the scores of three students among ten are *Good (Grade B)*, the scores of four students are *Okay (Grade C)*, the scores of two students are *Poor (Grade D)*, and the score of one student is *Awful (Grade F)*, respectively. Unfortunately, the *SUS scores* of three students among 10 are either *Poor (Grade D)* or *Awful (Grade F)*.

Here, we have to note that to calculate the *SUS score* correctly, the 10 standard questions in [28] should be used. The score in Table V can be changed by adopting the questions and increasing the number of questions. Besides, the number of subjects in our evaluation is very small. In future works, we will continue to evaluate the *instance file update function* by applying it to more teachers and students in various universities and to ask the usability using standard questions.

3) Evaluation of File Distribution: For the *instance file update function*, the teacher needs to distribute the files for the added/modified instances to the students. For this

TABLE VI: Questions for questionnaire on *instance file update function*.

no.	question
Q1	Are the instructions for the update function in the email clear?
Q2	Is it easy for you to update instance files using only instructions from email?
Q3	Is it easy for you to import the new instance files by yourself?
Q4	Do you think you can learn the web application structure by seeing our NPLAS project?

TABLE VII: Cronbach's alpha coefficient on *NPLAS* and *instance file update function*.

	total questions	sum of question variance	variance of total scores	Cronbach's alpha
NPLAS	12	7.401	37.888	0.878
Update	4	1.922	6.322	0.928

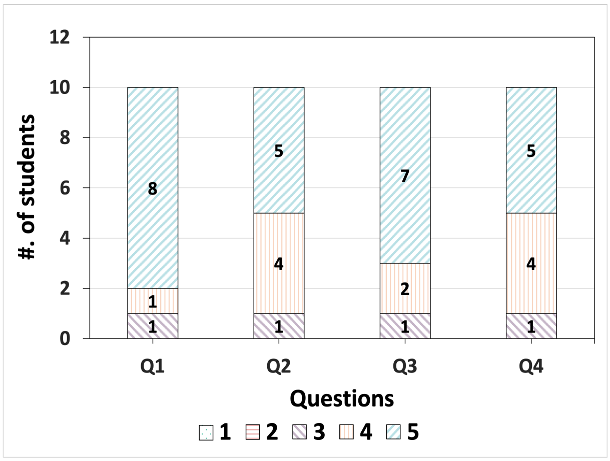


Fig. 16: Answers for questionnaire on *instance file update function*.

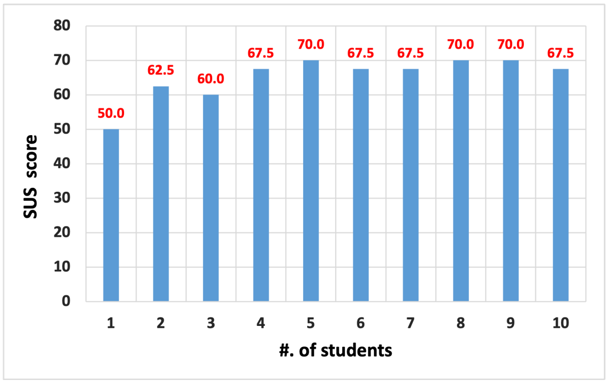


Fig. 17: SUS scale on *update function* by each student.

evaluation, we prepared the manual that explains how to run the application, upload the instance files, and send the message to the students, and generated two new *GUP* instance files. We asked the three students in our laboratory to conduct the procedure of this function by sending the instance files to the ten students. We confirmed that they successfully uploaded the files and sent the message.

4) Result Reliability Analysis by Cronbach's Alpha Coefficient: Furthermore, we evaluate the reliability of the results based on the *Cronbach's alpha coefficient*. Table VII shows 0.878 for *NPLAS*, which is minimally acceptable by most standards, and 0.928 for the *update function*, which is very close to the highest reliability 1.0.

TABLE VIII: Numbers of exercise problems in *NPLAS*.

GUP	VTP	MCP	EFP	CCP	PFP	CWP
28	15	12	26	13	15	43

D. Application to Java Course

For the application of the proposed system to the *Java programming course* in Okayama University, we generated exercise problems from sample source codes in the textbook, *"Yasashi Java"* [29]. This Japanese textbook is very popular among Japanese universities that use it in their Java programming courses [30]–[32]. It offers a lot of sample source codes that should be read and understood by students so that they can learn the Java programming using this textbook efficiently and comprehensively. However, for *Grammar-concept Understanding Problem (GUP)*, we used existing problems since it should cover the basic Java programming keywords that are common in any textbook. For *Code Writing Problem (CWP)*, we also used existing problems,which are different from the sample source codes, to avoid copying them.

To cover the Java programming concepts in the textbook, we selected several sample source codes at each chapter of the textbook for each problem type. Then, we generated the exercise problems in *NPLAS* from them using our generators, and installed them in the system. Table VIII shows the number of the problem instances in *NPLAS* for this Java programming course.

Then, we requested 57 third-year undergraduate students in Okayama University who were enrolled in a Java programming course in 2022, to install the *NPLAS* platform for Java and solve the exercise problems by themselves as the course assignments. Most of the students took the one-semester *C programming* course two years ago, and have never used the previous *JPLAS*. To help them install the platform and use the system, we prepared the installation and usage manuals. Besides, to help students understand hard concepts of *Java programming* such as object-oriented programming (OOP) concepts, we gave them short lectures on them using slides. After that, the students solved the exercise problems by themselves in home, where several graduate students assisted them as teaching assistants. It is noted that this course was the full online one due to the COVID-19 pandemic. All the students were able to successfully install the system and continue to study with their assignments.

Here, some students met compilation warnings at solving CWP instances on the system. After we analyzed them, we found that the *SuppressWarnings* annotation is necessary in the test code to disable the less important compilation warnings on the type, field, method, parameter, constructor, and local variable of the source code made by a student. By ignoring inappropriate compiler warnings for novices, they can concentrate on writing source codes using programming concepts to be studied. Thus, we modified the test codes and distributed them using the proposed *instance file update function*.

Furthermore, we also utilized the *NPLAS* system in the *Java programming* course in the 2023 class, taking into account the difficulties and feedback from the 2022 class. In the 2022 class, there were instances of errors during the installation process due to variations in students' PC operating system specifications. At that time, our teaching assistants analyzed and resolved the errors during online classes and informed them through emails. To address this issue, we updated the system installation manual to provide the solutions for those errors and other potential issues at installing *Docker* and *Windows Subsystem for Linux (WSL)*.

Additionally, we revised the CWP instance test codes and reduced the number of exercise problems from 43 to 33 to make the assessment more manageable and less overwhelming for the students, allowing them to focus on a slightly smaller set of problems and potentially improve their performance. By refining the implementation and the installation manual, it is expected that *NPLAS* system will be an effective and user-friendly platform for teaching and learning introductory Java programming.

E. Evaluation Summary

Here, we provide the summary of all the results in the various experiments and evaluations conducted in the paper.

1) Evaluation of NPLAS: The installation and operation of *NPLAS* were evaluated positively. The students' feedback on *NPLAS* showed the high level of satisfaction where 81% of them rated the system as either "Excellent" or "Good".

2) Evaluation of Instance File Update Function: Although the installation of the *instance file update function* was initially challenging for some students, this was addressed by providing the script to automate the process to them. The students' feedback on this function also showed the high level of satisfaction where 84% of them rated the function as either "Excellent" or "Good".

3) Application to Java Course: The *NPLAS* system was successfully applied to the *Java programming* course with a relatively large number of students. The feedback from the students and the teaching assistants led to the improvements of the system for the following year's course, including the improved installation process and manual, and the improved *test codes* for properly handling compilation warnings to novice students.

F. Discussion

In this section, we summarize the findings of our experiments and present the limitations of this study. Besides, we will compare our approach with existing solutions such as Codio and JetBrains, highlighting their specific strengths and differentiating features.

1) Selection of Server Platforms: Node.js and Laravel: In this paper, we presented two different systems. One is for the student and another is for the teacher. The first system, *NPLAS*, is presented as a personal learning platform for the student to solve programming problems. The second system, the *update function*, is presented for the teacher to modify or add exercises and send them to students' PCs via a local server and *Gmail*. The second system is presented in the paper because the first system is offline and the files cannot be changed by the teacher directly, although the teacher sometimes needs to do so.

For the first system, we used *Node.js* as the server platform, because it is a JavaScript-based framework that is familiar to us, and our other web application systems have been implemented using it. We were able to reuse some existing codes and simplify the development process.

For the second system, we used *PHP-Laravel* as the server platform to send data or updated files through the *Gmail* server. Laravel has a reputation for its strong focus on security, and it provides a secure default configuration. Laravel comes with a built-in CSRF protection that helps to protect against cross-site request forgery attacks. The authentication system is known to be robust and easy to use. Thus, Laravel becomes an excellent choice for the second system.

In conclusion, we used *Node.js* for *NPLAS* for the student side, and *Laravel* for the *update function* for the teacher side, based on their requirements and considerations, including the code reuse and security.

2) Findings: The experiments conducted in this study aimed to evaluate the effectiveness and usability of our proposal in programming education. Our approach focuses on providing a stable environment for learning by distributing the system through *Docker*, enabling both offline and online use. This offline functionality sets this solution apart from others, as it allows the student to access and work with the platform without an Internet connection. Furthermore, our platform offers a range of exercises at different levels, covering various aspects of programming mastery. We prioritize guiding students through the essential steps such as grammar study, code reading, code debugging, and code writing from scratch. The emphasis on code reading as a foundational skill is a distinguishing feature of our approach.

The results of the experiments indicated that the proposed approach effectively facilitated learning and skill development among participants. Students reported positive experiences with the stability and accessibility of the platform, particularly appreciating the ability to work offline. The structured progression of exercises, with a focus on code reading and gradual skill developments, was found to be beneficial for students' learning process. The inclusion of exercises like GUP and VTP for code reading proved valuable in strengthening students' understanding of programming concepts.

3) Comparing NPLAS with Codio, and JetBrains: While discussing the strengths of our proposed approach, it is important to acknowledge the specific strengths of other solutions such as *Codio* and *JetBrains* in the context of programming education. *Codio* offers cloud-based accessibility, allowing students to access their coding projects from anywhere with an Internet connection. It also provides educational features such as automated assessments and project-based learning, enhancing the learning experience. Additionally, *Codio* boasts a user-friendly interface, making it particularly suitable for beginners.

JetBrains, on the other hand, offers a powerful Integrated Development Environment (IDE) with advanced code editing, debugging, and testing capabilities. The IDE provides a comprehensive development environment for students to learn and practice programming. *JetBrains* is widely used in professional software development, exposing students to industry-standard practices and workflows. Moreover, *JetBrains* IDE supports a wide range of programming languages, enabling students to explore different languages and gain proficiency across multiple domains.

In comparison, the *NPLAS* system offers offline capability, customizability, and a strong emphasis on code reading and levels. The offline functionality allows students to use the system without relying on the Internet connection, providing flexibility and accessibility. The customizability of our system allows teachers to tailor it to their specific teaching requirements, enhancing the adaptability of the platform. Additionally, the focus on code reading and the structured progression of exercises provide comprehensive learning experiences, guiding students through different stages of programming mastery.

4) Limitations of the Study: Despite the positive findings and strengths of our proposed approach, it is important to consider the limitations of this study.

Firstly, the experiments were conducted with a limited number of participants, which may affect the generalizability of the results. Future studies with larger and more diverse participant samples would provide a more comprehensive understanding of the effectiveness of our approach.

Secondly, the study was conducted within a specific learning context, and the results may be influenced by the characteristics of the participants, the learning environment, and other contextual factors. It is crucial to consider the transferability of the findings to different educational settings and student populations.

Furthermore, the study primarily focused on the perceptions and experiences of the students using the proposed approach. While this provides valuable insights into usability and user satisfaction, future studies could incorporate objective performance measures to assess the impacts of the approach on students' programming skills and outcomes.

In summary, the experiments conducted in this paper demonstrated the effectiveness and usability of our proposed approach to programming learning. The emphasis on stability, offline functionality, code reading, and structured progression of exercises sets this approach apart from other existing solutions. However, it is important to acknowledge the limitations of the study, including the sample size, contextual factors, and the focus on subjective measures. Future research should address these limitations and further explore the effectiveness and impact of this approach in different settings.

VII. Related Works

In this section, we discuss related works in literature.

In [33], Ala-Mutka highlighted common challenges faced by novice programmers and provided an overview of existing efforts and discussions on contemporary methods employed in teaching programming. In [37], Konecki also pointed them out. By referencing these works, we can establish that our approach in *NPLAS* addresses these challenges and contributes to the ongoing discussions on effective programming education.

In [34], Carbone et al. investigated various factors that may contribute to student attrition in an introductory programming course, encompassing motivations and problem-solving skills. By referring to this study, we can emphasize the importance of addressing these factors in our *NPLAS* platform to improve students' learning experience and reduce dropout rates.

In [35], Queiros et al. discovered that *ToolPetcha* exemplifies a tool functioning as an automated assistant for programming-related issues, despite the existence of several proposed tools aimed at aiding students in overcoming programming learning challenges. It is relevant to our proposed research as it demonstrates existing efforts to develop tools that support students in overcoming programming learning difficulties. By referencing this work, we can show that our *NPLAS* platform aligns with the goal of providing automated assistance and guidance to students.

In [36], Kalemi et al. made tests and measurements for the time spent updating information using *Node.js* versus other technologies such as *PHP* or *Apache*, to highlight the advantages of using *Node.js* to update web contents in real-time. The results indicated that the *Node.js* application was twice as fast as the *PHP* or *Apache* application while the execution time appears more stable throughout the tests. Although the focus is technical, it is relevant to our proposed research as we implement *NPLAS* using Node.js. By referring to this study, we can highlight the advantages of using Node.js, such as its speed and stability, which contribute to the efficiency and reliability of our platform.

In [38], Frees proposed the use of *Node.js* for developing server-side web logic in JavaScript to overcome challenges in teaching web developments. The use of JavaScript on both the server and client allows students to acquire more knowledge depth, such as the internals of web developments including sockets, HTTP, and templates, and to study important core concepts such as event-driven programming and functional programming. It is relevant to our proposed research as we utilize Node.js in *NPLAS*. By referencing this work, we can emphasize the advantages of using JavaScript for web development, enabling students to gain a comprehensive understanding of web technologies.

In [39], Heredia et al. built two similar web applications using the *MEAN* and *Java EE* stacks, and compared the features in performances. The *MEAN* stack comprises *MongoDB* for the database, *Express* for the back-end framework, *Angular* for the front-end framework, and *Node.js* for the web server. On the other hand, the *Java EE* stack includes *MongoDB*, *Spring Boot* for the back-end framework, *JSP/HTML/CSS* for the front-end, and *Apache/Tomcat* for the web server. They found that the *MEAN* stack is an effective solution for web applications based on REST services. While the emphasis is different, it is relevant to our proposed research as we implement *NPLAS* using the MEAN stack. By referencing this study, we can support our decision to choose the MEAN stack, highlighting its effectiveness in developing web applications based on REST services.

In [40], Lala et al. provided methods to build a secure web application using *Node.js* according to *Open Web Application Security Project (OWASP)* guidelines. They used *HTML* with *EJS* for the front-end, *Node.js* for the back-end, and *MySQL* for the database. They also focused on the various vulnerabilities and the methods to secure the web application from these vulnerabilities. It was found that the web application considering them showed no negative results for the attacks that were performed on it earlier after their implementation. It is relevant to our proposed research as we implement *NPLAS* using Node.js and need to address security concerns. By considering the vulnerabilities mentioned in this work and implementing security measures accordingly, we ensure the safety and integrity of our *NPLAS* platform.

In [41], Figueiredo et al. introduced the HTProgramming application, crafted to aid in the instruction and learning of introductory programming. Students have access to exercises and receive immediate feedback based on the content introduced by the teacher. This tool empowers teachers to efficiently monitor the entire teaching and learning process, identifying students who are at a higher risk of failure at an early stage, enabling the allocation of additional attention to those individuals. It is relevant to our proposed research as our *NPLAS* platform also offers personalized guidance and feedback to students. By referencing this work, we can emphasize the alignment of our goal to enhance comprehension through *NPLAS* and provide effective support to students.

In [42], Crow et al. presented a systematic review of existing systems and the prevalence of various supplementary resources within them. In general, some systems were developed with the aim of teaching introductory programming concepts, and other systems were for guiding more specific aspects of programming. There are many tutoring systems to learn how to program at various levels, contents, and quality, and to support teaching of programming for introductory programming courses. They can be categorized by features including programming languages, manually or automatically generated learning problems, static or dynamic code analysis, and program visualizations. It is relevant to our proposed research as it provides an overview of existing systems in programming education. By referencing this work, we can position *NPLAS* within the broader context of tutoring systems and highlight its unique contributions and features.

In [43], Parihar et al. introduced *GradeIT*, a system designed to achieve both automated grading and program repairs in introductory programming courses. It evaluates submitted source code based on the number of passed test cases, the reciprocal of the time taken to solve them, and the percentage of successful compilations. The compilation score is computed as 1 - (number of compilation errors in the code) / (maximum number of compilation errors among students). For repairs, GradeIT employs straightforward rewriting rules to rectify common but simple compile-time errors. It is relevant to our proposed research as it emphasizes the importance of automated grading and feedback. By referring to this work, we can highlight the similar objectives of *NPLAS*, which provides automated assessment and support to students.

In [44], Fu et al. introduced a plug-in system for *Moodle* called *LAPLE (Learning Analytics in Programming Language Education)*. LAPLE is designed to offer a learning dashboard within Moodle, capturing students' behaviors in the classroom and identifying individual students' challenges with various concepts. *LAPLE* prompts students to write complete source codes, collecting and analyzing their compiling logs every five minutes to provide real-time visualizations for in-class feedback. The authors conducted an analysis of the distribution of 36 classified error types and encouraged students to focus on addressing frequent error types. *LAPLE* is tailored for use in online programming classes, while *NPLAS* is designed for self-study at home, even after class.

In [45], Hasany introduced an e-learning system named *c-Learn*, accessible to students as a web application anytime and anywhere. This system aims to address the issue where weaker students tend to struggle more as additional programming lessons are covered. Through *c-Learn*, students can incrementally grasp programming concepts, and in the event of errors, the system guides them back to the relevant topics they need to understand to resolve the problems. It is relevant to our proposed research as it addresses the problem of weaker students struggling with programming. By referencing this work, we can demonstrate that *NPLAS* offers similar guidance and support to students, helping them gradually improve their programming skills.

In [46], Souza et al. conducted systematic literature reviews on assessment tools for programming assignments, aiming to assist instructors in making informed selections for programming courses. They identified three categories of assessment tools: contests, quizzes, and software testing. In contests, a tool compiles a student's source code, executes it with a predefined set of test cases, and provides feedback on whether the code is accepted or not. Quizzes involve presenting a set of questions, with each requiring students to input a code fragment in the tool's interface. In software testing, the tool assesses the correctness of a student's source code by comparing its output with the model code's output or by running the test code against the source code. It is relevant to our proposed research as *NPLAS* incorporates testing functionalities using JUnit. It also supports a hint function to assist students in reaching the correct answer by providing partial guidance. By referencing this work, we can highlight the assessment capabilities of *NPLAS* and its support for students in improving their code quality and correctness.

In [47], Krusche et al. introduced an automated assessment management system named *ArTEMiS* for interactive learning. This system automatically evaluates solutions to programming exercises, delivering immediate feedback. Additionally, it includes an online code editor with interactive exercise instructions. *ArTEMiS* prompts students to submit complete source codes, a distinction from our proposed approach. It is relevant to our proposed research as both *ArTEMiS* and *NPLAS* focus on assessing and providing feedback on programming solutions. By referring to this work, we can emphasize the importance of automated assessment and feedback in programming education, which *NPLAS* offers.

In [48], Tung et al. introduced *Programming Learning Web (PLWeb)*, a programming exercise management system crafted to support teachers in generating exercises and aid students in programming studies. *PLWeb* offers an integrated development environment, serving as both an authoring tool for teachers to create exercises and a user-friendly editor for students to study programs and submit solutions. Visualized learning status assists teachers in aiding students facing challenges in reaching solutions, and a plagiarism detection tool is incorporated to discourage plagiarism among students. It is relevant to our proposed research as both *PLWeb* and *NPLAS* aims to support teachers in designing exercises and provide a friendly environment for students to study and submit solutions. By referencing this work, we can highlight the similar functionalities and goals of *NPLAS*.

In [49], Busjahn et al. determined, through literature reviews and interviews, that *code reading* is linked to understanding programs, algorithms, and algorithmic ideas, along with details, and is essential in various aspects of learning programming. Despite this, there is limited knowledge about the reading and comprehension process of learners. The authors suggested that a potential approach to enhance programming learning is to directly teach code reading, incorporating specific reading strategies. It is relevant to our proposed research as *NPLAS* introduces the value trace problem (VTP) as a means to foster programming learning. By referring to this work, we can highlight the significance of code reading and comprehension, which *NPLAS* aims to enhance through its unique features.

VIII. Conclusion

This paper presented the implementation of the *Java programming learning assistant system* using *Node.js* (NPLAS) and the *instance file update function* using *Angular* and *Laravel*. *Docker* is adopted for their easy and solid deployments to novice students. Besides, *NPLAS* was extended to *Python programming* learning. In evaluations, all of the requested teachers and students in three universities in Japan and Indonesia successfully installed *NPLAS*, solved instances on it, and updated the instance files. The results validated the effectiveness and accuracy of the implementations. In future works, we will adopt a database to manage instance files and extend the *NPLAS* platform to encompass other prominent programming languages, including *C*, *C++*, and *JavaScript*. Additionally, we will persist in enhancing *NPLAS* by incorporating new features and functions for programming learning.

References

[1] S. I. Ao, A. H. S. Chan, and H. Katagiri, “IAENG Transactions on Engineering Sciences - Special Issue for the International Association of Engineers Conferences,” World Sci. Pub., vol. 2, pp. 517-530, 2018.

[2] S. T. Aung, N. Funabiki, Y. W. Syaifudin, H. H. S. Kyaw, S. L. Aung, N. K. Dim, and W.-C. Kao, “A proposal of grammar-concept understanding problem in Java programming learning assistant system,” Journal of Advances in Information Technology (JAIT), vol. 12, no. 4, pp. 342-350, 2021. Available at: https://doi.org/10.12720/jait.12.4.342-350.

[3] K. K. Zaw, N. Funabiki, Y. W. Syaifudin, H. H. S. Kyaw, S. L. Aung, N. K. Dim, and W.-C. Kao, “A proposal of value trace problem for algorithm code reading in Java programming learning assistant system,” Information Engineering Express, vol. 1, no. 3, pp. 9-18, 2015. Available at: https://doi.org/10.52731/iee.v1.i3.3.

[4] Y. Jing, N. Funabiki, S. T. Aung, X. Lu, A. A. Puspitasari, H. H. S. Kyaw, and W.-C. Kao, “A proposal of mistake correction problem for debugging study in C programming learning assistant system,” International Journal of Information and Education Technology (IJIET), vol. 12, pp. 1158-1163. 2022. https://doi.org/10.18178/ijiet.2022.12.11.1733.

[5] N. Funabiki, Tana, K. K. Zaw, N. Ishihara, and W.-C. Kao, “A graph-based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system,” IAENG International Journal of Computer Science, vol. 44, no. 2, pp. 247-260, 2017.

[6] H. H. S. Kyaw, S. S. Wint, N. Funabiki, and W.-C. Kao, “A code completion problem in Java programming learning assistant system,” IAENG International Journal of Computer Science, vol. 47, no. 3, pp. 350-359, 2020.

[7] N. Funabiki, Y. Matsushima, T. Nakanishi, N. Amano, “A Java programming learning assistant system using test-driven development method,” IAENG International Journal of Computer Science, vol. 40, no. 1, pp. 38-46, 2013.

[8] N. Ishihara, N. Funabiki, M. Kuribayashi, W.-C. Kao, “A software architecture for Java programming learning assistant system,” International Journal of Computer and Software Engineering, vol. 2, no. 1, pp. 1-7. 2017. https://doi.org/10.15344/2456-4451/2017/116.

[9] N. Funabiki, H. Masaoka, N. Ishihara, I-W. Lai, and W.-C. Kao, “Offline answering function for fill-in-blank problems in Java programming learning assistant system,” Proceedings of the IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), pp. 324-325, 2016. https://doi.org/10.1109/ICCE-TW.2016.7521045.

[10] D. Herron: Node.js web development, 3rd ed.; Packt Publishing Ltd: Livery Place 35 Livery Street Birmingham B3 2PB, UK, 2016.

[11] Express (online), https://expressjs.com/.

[12] J. Berm´udez-Ortega, E. Besada-Portas, J. A. L´opez-Orozco, J. A. Bonache-Seco, and J. M. de la Cruz, “Remote web-based control laboratory for mobile devices based on EJsS, Raspberry Pi and Node.js,” IFAC-Papers OnLine, vol. 48, pp. 158–163, 2015. Available at: https://doi.org/10.1016/j.ifacol.2015.11.230.

[13] J. I. Sihotang, Y. R. Simanjuntak, and A. F. Pakpahan, “Membership information system using Node JS,” International Scholars Conference (ISC), vol. 7, no. 1, pp. 1729-1740. 2019. Available at: https://doi.org/10.35974/isc.v7i1.1372.

[14] A. A. Pushkarev, and O. E. Yakubailik, “A web application for visualization, analysis, and processing of agricultural monitoring spatial-temporal data,” Proceedings of the CEUR Works, pp. 231-237, 2021.

[15] R. McKendrick. *Monitoring Docker*, Packt Publishing, United Kingdom, 2015.

[16] Angular (online), https://angular.io/.

[17] Laravel (online), https://laravel.com/.

[18] TypeScript (online), https://www.typescriptlang.org/.

[19] Express.js Tutorial (online), https://www.javatpoint.com/expressjs-tutorial.

[20] JUnit (online), https://junit.org/junit5/.

[21] Docker Hub (online), https://hub.docker.com/signup.

[22] RestAPI (online), https://www.ibm.com/cloud/learn/rest-apis.

[23] PHP (online), https://www.php.net/.

[24] Composer (online), https://getcomposer.org/.

[25] Unittest (online), https://docs.python.org/3/library/unittest.html.

[26] J. Brooke, “SUS: A Retrospective,” Journal of Usability Studies, vol. 8, no. 2, pp. 29-40, 2013.

[27] Measuring and Interpreting System Usability Scale - SUS (online), https://uiuxtrend.com/measuring-system-usability-scale-sus/.

[28] System Usability Scale - SUS (online), https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

[29] Mana Takahashi. *Yasashi Java*, 7th ed.; SB Creative, Japan, 2019.

[30] Java Programming Course (online), https://java.cse.ce.nihon-u.ac.jp/enshu5/java/index.html.

[31] Java Programming Course (online), https://syllabus.hosei.ac.jp/web/preview.php?nendo=2020&t_lmode=sp&template=t1&no_id=2015538&gakubu_id=%E7%B5%8C%E5%96%B6%E5%AD%A6%E9%83%A8&gakubueng=AF

[32] Java Programming Course (online), https://navi.hus.ac.jp/upload/files/12.2021%E5%B9%B4%E5%BA%A6%E5%BE%8C%E6%9C%9F%E6%95%99%E7%A7%91%E6%9B%B8%E8%B2%A9%E5%A3%B2%E3%83%AA%E3%82%B9%E3%83%88%E3%83%A1%E3%83%87.pdf

[33] K. Ala-Mutka, “Problems in learning and teaching programming - a literature study for developing visualizations in the Codewitz-Minerva project,” Codewitz needs analysis, vol. 20, 2004.

[34] A. Carbone, I. Mitchell, J. Hurst, and D. Gunstone, “An exploration of internal factors influencing student learning of programming,” Proceedings of the Eleventh Australasian Computing Education Conference (ACE2009), pp. 25-34, 2009.

[35] R. Queiros, and J. P. Leal, “PETCHA - a programming exercises teaching assistant,” Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE ’12), pp. 192–197, 2012. Available at: https://doi.org/10.1145/2325296.2325344.

[36] E. Kalemi, and K. Tola, “Updating web content in real time using Node.js,”Journal Soft. Eng. Simul, vol. 1, pp. 01–06, 2013.

[37] M. Konecki, "Problems in programming education and means of their improvement," DAAAM Int. Sci. Book, Chapter 37, pp. 459-470, 2014.
[38] S. Frees, "A place for Node.js in the computer science curriculum," Journal of Computing Sciences in Colleges, vol. 30, pp. 84-91, 2015.
[39] J. S. Heredia, G. C. Sailema, "Comparative analysis for web applications based on REST services: MEAN stack and Java EE stack," KnE Engineering, vol. 30, pp. 82–100, 2018. Available at: https://doi.org/10.18502/keg.v3i9.3647.
[40] S. K. Lala, A. Kumar, S. T, "Secure web development using OWASP guidelines," Proceedings of the 5th International Conference on Intelligent Computing and Control Systems (ICICCS), pp. 323-332, 2021. Available at: https://doi.org/10.1109/ICICCS51141.2021.9432179.
[41] J. Figueiredo, and F. Gracia-Penalvo, "A tool help for introductory programming courses", Proceedings of the Ninth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'21), pp. 18-24, 2021. Available at: https://doi.org/10.1145/3486011.3486413.
[42] T. Crow, A. Luxton-Reilly, and W. Burkhard, "Intelligent tutoring systems for programming education: a systematic review," Proceedings of the 20th Australasian Computing Education Conference, pp. 53-62, 2018. Available at: https://doi.org/10.1145/3160489.3160492.
[43] S. Parihar, R. Das, Z. Dadachanji, A. Karkare, P. K. Singh, and A. Bhattacharya, "Automatic grading and feedback using program repair for introductory programming courses," Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, Bologna, pp. 92-97, 2017. Available at: https://doi.org/10.1145/3059009.3059026.
[44] X. Fu, A. Shimada, H. Ogata, Y. Taniguchi, and D. Suehiro, "Real-time learning analytics for C programming language courses," Proceedings of the Seventh International Learning Analytics and Knowledge Conference, pp. 280–288, 2017. Available at: https://doi.org/10.1145/3027385.3027407.
[45] N. Hasany, "E-learning student assistance model for the first computer programming course," International Journal on Integrating Technology in Education (IJITE), vol. 6, no. 1, pp. 1-7, 2017. Available at: https://doi.org/10.5121/ijite.2017.6101.
[46] D. M. Souza, K. R. Felizardo, and E. F. Barbosa, "A systematic literature review of assessment tools for programming assignments," Proceedings of the IEEE 29th International Conference on Software Engineering Education and Training (CSEET), pp. 147-156, 2016. Available at: https://doi.org/10.1109/CSEET.2016.48.
[47] S. Krusche, and A. Seitz, "ArTEMiS: an automatic assessment management system for interactive learning," Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18), pp. 284–289, 2018. Available at: https://doi.org/10.1145/3159450.3159602.
[48] S. H. Tung, T. T. Lin, and Y. H. Lin, "An exercise management system for teaching programming," Journal of Software, vol. 8, no. 7, pp. 1718-1725, 2013. Available at: https://doi.org/10.4304/jsw.8.7.1718-1725.
[49] T. Busjahn, and C. Schulte, "The use of code reading in teaching programming," Proceedings of the 13th Koli Calling International Conference on Computing Education Research (Koli Calling '13), pp. 3–11, 2013. Available at: https://doi.org/10.1145/2526968.2526969.

S. T. Aung received the B.E. degree in Information Technology from the University of Technology (Thanlyin), Myanmar, in 2017. She received the M.E. degree in Electronic and Information System Engineering at Okayama University, Japan, in 2023. She is currently a Ph.D student in Department of Information and Communication System Engineering at Okayama University, Japan, and is the recipient of an OU fellowship. Her research interests include educational technology.

L. H. Aung received the B.C.Sc. degree in Information Technology from the University of Computer Studies, Yangon, Myanmar, in 2019. From 2019 to 2022, he was with FRONTIIR Co., Ltd (ISP Company of Myanmar) as an associate engineer/senior associate engineer. He is currently a Ph.D student in Department of Information and Communication System Engineering at Okayama University, Japan. His research interests include educational technology.

N. Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. In 2001, he moved to the Department of Communication Network Engineering (currently, Department of Electrical and Communication Engineering) at Okayama University as a professor. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.

S. Yamaguchi received the B.E., M.E. and D.E. degrees from Yamaguchi University, Japan, in 1992, 1994 and 2002, respectively. He was a Visiting Scholar in the Department of Computer Science at University of Illinois at Chicago, United States, in 2007. He is currently a Professor in the Graduate School of Sciences and Technology for Innovation, Yamaguchi University, Japan. His research interests are in the area of net theory and its applications including IoT, Cyber Security, and Service Science. He is a Board of Govertnor, IEEE Comsumer Electronics Society, and the Chair of Young Professionals Committee of the society. He is a senior member of IEEE and IEICE.

Y. W. Syaifudin is an associate professor at State Polytechnic of Malang, Indonesia. He received a bachelor's degree in informatics from Bandung Institute of Technology, Indonesia, in 2003 and a master's degree in information technology from Sepuluh Nopember Institute of Technology, Indonesia, in 2011. Finally, in 2021, the Ph.D. degree in information and communication systems was received from Okayama University, Japan, respectively. He is also a reviewer for some reputable journals and conferences, and is active as Director of Intelligence System and Digital Economy Innovation Research Institute (ISDEI) and Chairman of Academic Association of Creative Economy (AACE). His research interests include technology-enhanced learning, intelligent systems, and data analytics.

W-C. Kao received the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, New Taipei, in 1992 and 1996, respectively. He is a chair professor with the Department of Electrical Engineering, National Taiwan Normal University, New Taipei, Taiwan. Before he joined academia in 2004, he was a department manager with SoC Technology Center, ITRI, Taiwan, an AVP with NuCam Corporation, Foxlink Group, and the co-founder of SiPix Technology Inc. He is the president of the IEEE Consumer Electronics Society. He is a fellow of IEEE. His research interests include system-on-a-chip (SoC), embedded software design, flexible electronic paper, machine vision systems, and digital camera systems.