Software Algorithms Evaluation Platform Using a Docker-Based Framework

By

Acil Ramadan Ibrahim Abdel Naby

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

M.Eng. in Computer Engineering

Memorial University of Newfoundland

May 2021

Under the supervision of

Theodore S. Norvell, Ph.D., P.Eng.

Associate Professor

Electrical and Computer Engineering

Faculty of Engineering & Applied Science

Memorial University of Newfoundland

St. John's, NL A1B 3X5, Canada

Mohamed Shehata, Ph.D., P.Eng.

Associate Professor

Electrical and Computer Engineering

Faculty of Engineering & Applied Science

Memorial University of Newfoundland

St. John's, NL A1B 3X5, Canada

Abstract

Automating the evaluation of algorithms is a complex task that involves many dependencies, e.g., the inputs, outputs, the compilers, interpreters, execution environments, libraries, operating systems, and hardware dependencies. Each algorithm requires specific performance evaluation criteria to evaluate it during and after execution. This work presents a ADESA (Automatic Deployment and Evaluation of Software Algorithms) framework based on a Docker framework for automating the execution of the algorithms provided by the user and automating the evaluation of these algorithms.

The framework is applied to the automatic evaluation of image processing algorithms (AEIPA). AEIPA automates the processes of searching for a matching data set or uploading new data sets, uploading a new image processing algorithm (IPA) that has been developed using any programming language, searching for matching IPAs, executing all those IPAs with one or more data set, and comparing the results of the executed IPAs using each data set. As a case study, two face detection algorithms that implement the Haar Cascade classifier using C with OpenCV and Python with OpenCV, have been used to evaluate AEIPA. Results from this work confirm that AEIPA supports the execution of different programming languages using Docker images.

Acknowledgments

First and foremost, I would like to thank my supervisors sincerely, Dr. Theodore Norvell and Dr. Mohamed Shehata, who guided me through this research experience, taught me how research is done, provided insightful advice, and developed my attention to details. I would also like to extend my gratitude to the Department of Electrical and Computer Engineering, Memorial University of Newfoundland. Without their generous support, this thesis would have been impossible. NSERC (Natural Sciences and Engineering Research Council of Canada) Strategic Projects Grant and scholarships from the University's School of Graduate Studies have financially supported this research.

I want to take this opportunity also to express my heartfelt gratitude to my family for their unwavering encouragement. They are my backbone to hold me up and my always supporters. Also, I would like to dedicate all my work to my beloved father Ramadan I. Abdel Naby, who has passed away and to my beloved daughter Nada who has fruitfully enlightened my life since her birth during my master's journey.

Table of Contents

Publications

[1] A. A. Naby, M. S. Shehata, and T. S. Norvell, "AEIPA: Docker-based system for Automated Evaluation of Image Processing Algorithms," 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), Windsor, ON, 2017, pp. 1-4.

[2] A. A. Naby, M. S. Shehata, and T. S. Norvell, "A Survey of Distributed Architectures for Computer Vision Systems" 2015 IEEE 24th Annual Newfoundland Electrical and Computer Engineering Conference (NECEC). St. John's, NL, 2015.

Abbreviations

ADESA	Automatic Deployment and Evaluation of Software Algorithms framework
AEIPA	Automated Evaluation of Image Processing Algorithms system
DAO	Data Access Object
DTO	Data Transfer Object
IPA	Image Processing Algorithm
MVC	Model-View-Controller framework

Table of Figures

Chapter 1

Introduction

This first chapter introduces the proposed framework, ADESA, the motivation behind this work, research questions, problem overview, and the organization of the rest of the chapters.

1.1 Motivation

In recent decades, the complexity of software systems design has increased [71-72]. These software systems go through the software development life cycle (SDLC): requirements definition, system design and analysis, implementation, testing and quality control, verification and validation, maintenance, deployment, reengineering, and reuse when needed as detailed in [12]. The increasing complexity of software systems leads to the need to use automated approaches in order to develop and evolve the systems in an economical and timely manner. Applying automation to software engineering activities significantly increases the productivity and quality of the software being developed [73]. Automating software engineering processes is accomplished by modeling, adapting, and representing knowledge in software activities, as explained in the next chapters.

In this work, research focuses on automating two main software engineering processes. The first one is automating the processes of deploying and executing software algorithms. The second focus is automating the evaluation of the software, i.e., automating the software quality control process. The process of developing software algorithms involves many different factors such as the programming language used, operating system, hardware specifications, multiple inputs to

the algorithm, and multiple outputs. The combination of all these factors increases the complexity of automatically deploying the software.

Evaluating software in this work refers to the process of assessing the quality of the software. Software evaluation represents the testing, quality control, verification, and validation phases of SDLC. Algorithm evaluation is the key in detecting malfunctioning software and errors that would lead to additional cost. Evaluating an algorithm takes over 50% of the cost of the SDLC as in [1]. Software customers reported that they had spent 21 billion US dollars on software maintenance as a result of inadequate software evaluation and errors reported [2]. The increasing complexity of software products is measured in thousands of lines of code as reported in U.S. software industry which leads to lower quality and increased cost to manage [5]. Time limitations and human resources costs are noteworthy constraints on the quality evaluation process, so, investing in software quality evaluation infrastructure saves these previously mentioned expenses [4]. That is why automating software evaluation helps to improve the quality and effectiveness of the software algorithm and reduces the software costs in the long run [5].

In this thesis, the evaluation of algorithms is automated in order to improve the effectiveness of the evaluation process. Data is collected during and after the execution of a software algorithm and the results are presented in a final report. The report contains some predefined evaluation matrices, in addition to a comparison between all of the algorithms being executed. Using the ADESA framework, software developers can focus on developing the algorithm and the test cases, leaving the repetitive evaluation tasks to be executed automatically. By this, human resources can be used more efficiently, which consequently results in savings in the testing, evaluation processes, and the overall development budget [3]. Also, investing in automating the

quality evaluation of software can cover a large range of test cases and can save all the headache of delivering a malfunctioning software [5].

The U.S. software industry, recognizing the importance of automating the evaluation of software, invested 2.6 billion dollars in automating software evaluation tools in 2004 [6]. Evaluation automation is more suitable for running repetitive tasks using different inputs [7,8]. Also, automating the evaluation process depends predominantly on the testability of the software algorithm [9]. In this work, the software algorithm to be evaluated needs to be developed in an independent form, which means that the software algorithm needs to be an encapsulated set of procedures that require a predefined type of input(s) and produce an expected type of output(s) to become eligible for automating the evaluation. The output(s) of the software algorithm are used as an input to the evaluation algorithm in order to evaluate the quality and efficiency of the software algorithm. Typical evaluation automation process includes the development of one or more evaluation algorithms, executing that evaluation algorithm(s) comparing the output of the software algorithm versus the expected output, and verifying the final results.

Reduction of software costs, in addition to the improvement of the quality of software products are the key objectives in all software development processes [10, 11]. Thus, a successful introduction of automating the software evaluation infrastructure combines these two objectives.

The outcome of this study is an implemented framework that serves two purposes. First, automating the software algorithm(s) deployment and execution. Using Docker containers in the proposed system has provided deployment portability by separating the development and

deployment environments. Second, automating the evaluation of the algorithm's results and generating an evaluation report. Further, a discussion is provided about the usability, applicability, and maintainability of the implemented design during the conducted research. Results show that the software engineering life cycle is sped up by automating the deployment and execution of software and automating the evaluation of the software's execution. Using the proposed framework, software dependencies become no more a concern to the software's developers when designing code and that is because of using fully independent Docker images that require only the source code in order to be executed.

1.2 Research Questions

The primary aim of this research is to study and develop a web-based software automation framework based on Docker for automating the execution and evaluation processes of software algorithms. This research leads to some fundamental questions as follows:

What is the importance of developing a shared system for automating the execution and evaluation of software algorithms?

Developing a shared automation system simplifies the processes of executing software algorithms that have been developed using different programming languages, libraries, operating systems, and other dependencies. The shared system contains Docker containers for every different working environment, ready to execute the algorithm. Also, the user can easily find other algorithms solving a specific problem in order to compare and evaluate different algorithms.

Is it possible to automate the creation and deployment of the software working environment?

Yes. Using Docker, a whole working environment is created from scratch using a sequence of Unix commands to be executed using Docker Daemon server. In this work, a separate Docker image is built for each unique working environment. For example, a Docker image deploys Ubuntu V.16 and Python V.3.7.1 with some additional libraries. Another Docker image deploys Ubuntu V.16 and Python V.2.7.15 with some additional libraries. Each Docker image is instantiated into Docker containers by just selecting the needed working environment type.

How does the proposed framework enhance the portability of executing multiple software applications through a distributed system?

As mentioned in the previous answer, the implemented framework generates independent Docker containers that contain a separate working environment. Each algorithm executes in a separate Docker container, and multiple containers can be executed at the same time in parallel. Each of the Docker containers is created in a separate thread, and each thread processes only one data input at a time. So, executing an algorithm using ten, hundreds, or even thousands of inputs can be distributed all over the CPUs, independently. That is how containerizing the algorithms enhances the mobility of executing one or more algorithms over a distributed system, especially with large data sets.

Why is automating the evaluation of software algorithms useful?

As mentioned before in [73], automating software engineering activities significantly increases the quality and productivity of the software being implemented. Manual evaluation of

software algorithms is an extremely time-consuming process. Algorithms require evaluating both the execution performance and execution results. The more time the execution of a software algorithm takes the more time evaluating that software takes. Also, some algorithms' output would be only in a visual representation, like the output of Image Processing Algorithms (IPAs) as we will see through this study. Such outputs need extra processing in order to evaluate its end-results quality. Each output is analyzed and compared to a benchmark output in order to evaluate the accuracy of the algorithm. Quantitative results are required for an accurate comparison of the IPAs. That is why the process of manually evaluating the software algorithm is time-consuming. In the proposed system, the Evaluation module allows the visual results of the algorithms to be examined according to a set of standard evaluation matrices. Typically, the Evaluation module automatically generates evaluation matrices [13]: true positive, false positive, true-negative, false-negative, true-positive rate, false-positive rate, precision, and recall, for the visual results of IPAs.

1.3 Problem Overview

The primary objectives of this work are to develop a framework for automating software engineering activities: execution and evaluation. The new framework addresses the difficulty of automating the execution of software algorithms and automatically evaluating and comparing the execution results. The complexity of the unlimited dependencies combination for each software algorithm makes it hard to generalize a single approach to automate the execution and evaluation of the software algorithm. Dependencies vary from the used programming language such as such as C/C++ (GCC), Java, PHP, Python, Hy (Hylang), Go (Golang), Node, Perl, Rails,

Clojure, Ruby, OpenCV and Matlab, operating systems, hardware specifications, and even the slight differences between each version of the programming language.

On the other side, software developers spend time searching for the appropriate dataset for executing the software algorithm. Also, there is a hectic effort in searching for, installing, executing, and evaluating all the algorithms that match the same type of the algorithm being developed for the purpose of a fair algorithms comparison. As explained in the case study of the Automated Execution of Image Processing Algorithms (AEIPA) in chapter 4, each IPA requires a specific type of images dataset. Preparing the required dataset of images, executing them against each IPA, and evaluating the final results have always been time-consuming problems to the developers.

1.4 Research Goals

The goals of the conducted research are summarized in the below points. Each goal is achieved and discussed in detail in the next chapters.

- Constructing a working environment for executing software algorithms with simple steps and within seconds (depending on the Dockerfile instructions).
- Easy automation of executing software algorithms.
- Scalable distribution of executing the software algorithms.
- Evaluating the quality of the end results of executing the software algorithms.
- Comparing the performance of many executed software algorithms solving the same problem.

1.5 Organization of the Thesis

This thesis is structured as follows. First, research questions about the problem that is being addressed through this work is explained in the first chapter. Chapter 2 presents a background on the research problem and software engineering automation. This chapter defines the software algorithms used and their structure, how to automate software algorithms execution and evaluation, and Docker containerization in depth. A general-purpose framework, ADESA, is explained thoroughly in Chapter 3 with its architecture and implementation. In the same chapter, the four modules composing ADESA are explained in detail: ADESA Input Supplier, ADESA Software Factory, Docker Engine, and ADESA Data Repository. A more detailed digging into the twofold (software deployment automation system design and performance evaluation automation) are explained intensively as a case study of image processing algorithms AEIPA (Automatic Evaluation of Image Processing Algorithms) in Chapter 4. The thesis concludes with a summary of the contributions, limitations, and outline of the future work in Chapter 5. All the related references are listed in the last chapter, Chapter 6.

Chapter 2

Background

This chapter reviews the work related to software engineering automation for both deployment and evaluation activities. For a better understanding of this domain and the different aspects of the problem, this chapter begins with an overview of software algorithms since they are the basic components to be automated. Then, analysing inputs, outputs, software execution automation, software evaluation automation, and Docker containerization are explained in detail. After this, software engineering automation will be reviewed in terms of how each activity in the SDLC is automated.

2.1 Software Algorithms

The development of the computers through the recent years has led to the rapid transition to digital implementation of the systems in many fields. In this work, the ADESA framework is implemented to automate the execution and evaluation of the software algorithms. So, let's take a deeper look at software algorithms. A software algorithm is the specification of a well-defined set of computational instructions that accomplish a certain computer task. That's the simplest way to describe the software algorithm in order to explain the proposed ADESA framework. In other words, an algorithm is the procedures that are implemented in order to solve an algorithmic problem. A computer program is designed based on some algorithms that solve an algorithmic problem [14]. An algorithmic problem can be defined by specifying some values as input instances and produces some other values as outputs after executing the software

algorithm [15]. Thus, the software algorithm is the sequence of instructions, with a precise description, for transforming inputs into outputs as in Figure 1.

Figure 2.1: Software Algorithm Components

As a simplified example, consider the search problem using a key. The problem is to search in a list of values to find a certain key. The output should be the index number of the matching value or a zero in case of not finding the key in the list. This search problem can be defined as follows:

Inputs : A list of n numbers (x_1, x_2, ..., x_n) and a key (k).

Output : i where $0 \leq i \leq n$ and x_i = k.

As an example, given the instance input list (W, Y, N, I, D) and a searching key (Y), a correct algorithm returns two as the output. Note how general this problem is. It can apply to lists of characters, lists of numbers, or lists of any type for which equality is decidable.

There are three types of a software algorithm's evaluation addressed in this work: evaluation of correctness, evaluation of error rate, and evaluation of system performance. A software algorithm is considered working correctly when every input instance results in a correct expected output. When there is a frequency of errors occurrence, then the algorithm's outputs would be evaluated by the error rate [12 - 14]. Image processing and number-theoretic algorithms are examples of the algorithms that use the error rate for evaluation. These algorithms collect the errors across all the executed inputs to evaluate the execution proportionally to the total number of the executed inputs. In chapter 4, a more detailed explanation is presented about how AEIPA

evaluates image processing algorithms using true-positive rate (TPR), false-positive rate (FPR), true- negative rate (TNR), and false-negative rate (FNR) in the case study of the implemented AEIPA system.

Also, evaluating an algorithm's performance relies on the efficiency of the algorithms used as much as the efficiency of the used hardware and operating system. Software's performance determines the stability, reliability, scalability, responsiveness, and resource usage. Performance evaluation results determine whether the software meets the user's requirements under expected workloads or not.

The three types of a software algorithm's evaluation are not alternatives. Suppose, for example, the goal of a face recognition algorithm is to count the number of faces. The input to this algorithm is an image, and the output would be a number indicating the number of faces found. If something interrupts the execution of the algorithm or the algorithm never terminates, the output was something different than the predefined number, then the algorithm is considered as incorrect. However, even if the number is correct, there is a need to evaluate the ability of the algorithm to count faces accurately, it may have counted some oval shapes in the image. So, comparing the end results only is not always enough to evaluate an algorithm. Also, the performance measurements, such as the speed and resources usages, are mandatory when comparing multiple algorithms.

2.2 Analyzing Software Algorithms

Analyzing the execution of a software algorithm measures the resources that the algorithm requires. Resources can be computer hardware, memory, computational time, and communication bandwidth. The quality of a software algorithm is identified using four desirable

properties: correctness, efficiency, accuracy, and the ability to be reused [12 - 14]. As mentioned in the previous section, a software algorithm is correct as long as every input instance yields a correct answer every time the algorithm executes. Precise reasoning is needed to prove that an algorithm is correct. That is why implementing, deploying, and evaluating any software algorithm are long processes and liable to errors.

An algorithm's efficiency must not be disregarded, especially for real-time programming. Identifying the efficiency of an algorithm can be done by analyzing and comparing it to many other software algorithms in the same specific problem category. The comparison process between many software algorithms is difficult because the behavior of each algorithm is different for each possible input instance, different implementation, and different platform. The time taken by the search problem, as an example, is determined by the input instance, i.e., searching for a value in a large list of inputs takes much more time than searching for a value in two values list. Moreover, the running time for finding value at the beginning of a large input list, what is called best-case scenario, is less than the running time in case of finding value at the end of the same input list, called worst-case scenario.

On the other hand, the software algorithm's reusability is essential to adapt the algorithm to various software applications [16, 17]. Algorithms should be designed in a generic way that leads to easy and flexible implementation, which will help in the system's scalability afterwards. In [74], the study shows that software reuse is identified as a major advancement in software quality and productivity. In order to design a reusable software, organizational and technical perspectives need to be taken into consideration. Organizational perspective covers the creating reusable

components while technical perspective enables the use of existing components in a new software [74 - 76].

2.3 Automating Software Engineering Processes

This work automates two software engineering processes: the deployment and execution of software algorithms and the software's quality assessment. Automating the deployment of software algorithms requires preparing the working environment with all the libraries, programming language installation, operating system, and any other dependencies [12]. A working environment should be established and ready to deploy and execute the software algorithm, and that is the main problem that was faced during this work. Each software algorithm has different requirements in order to be executed. If the solution is to prepare each possible environment, then the combination of all the programming languages, programming languages version number, libraries, operating systems, hardware specifications, and other dependencies will be tremendous. In addition, the solution system should have any new emerging technologies into consideration. The solution system should be open to include new technologies and upgrades with the least possible effort.

Moreover, automating the software's quality evaluation is a wide topic that involves evaluating the correctness, error rate, as well as the performance of the software being evaluated as explained in a previous section in this chapter. An algorithm is said to be correct when for each input it produces a correct output. A program is considered correct when: there is a finite set of inputs, every input is used and result in a correct output, and the program is well defined. The simplest technique to verify the program correctness is by feeding various input instances to the evaluated program and verifying the correctness of each output.

Error rate evaluation measures the accuracy of the algorithm's results. It can be determined in many ways. One way is by calculating the rate of errors found in the outputs and comparing them to the correct outputs (Groundtruth data, for example [82- 86]). Load testing is required in order to evaluate how the algorithm acts under a certainly expected load. In other words, it is putting heavy demand on the software program and measuring its response. Performance evaluation measures in a quantitative form of how efficient the software program is consuming resources. These measurements give the developer a practical insight into how the algorithm is acting while being executed [77 – 81].

2.4 Docker

Docker is an open-source technology that performs virtualization in the operating system level [61]. It speeds up the process to develop, ship, and run software applications as it separates the application from the infrastructure. Docker is a software platform that is designed for both Linux and Windows. Docker avails the OS kernel of resource isolation features using control groups (cgroups) in Linux. A cgroup shares available resources (hardware, memory, ...etc.) to Docker containers. That is how Docker can create multiple Docker containers independently using Docker images. Docker consists of Docker Engine and Docker Hub. Docker engine is the core technology that builds and runs Docker containers, which are instances of Docker images. Docker Hub is a Cloud-based service that shares Docker images. It allows building, testing, and storing Docker images. Docker engine creates Docker images using a set of well-defined Unix instructions for preparing an environment to execute the software algorithm. A Docker image consists of tools, system libraries, and dependencies for the executable application code.

A Docker image is a template file that includes instructions that are executed to create Docker container(s). As shown in Figure 2.4.1, Docker containerization has gained great importance recently because it encapsulates a whole operating system with an executable application above an abstraction layer from the physical hardware [18]. Containerization can ease the adoption of new software as the business evolves. Also, containerization helps applications receive greater protection while maintaining performance and enabling policy-driven automation [19, 20]. Docker containers have been adopted in DevOps and microservices. Docker containers support the agility of deployment and evaluation of software applications as in [21].

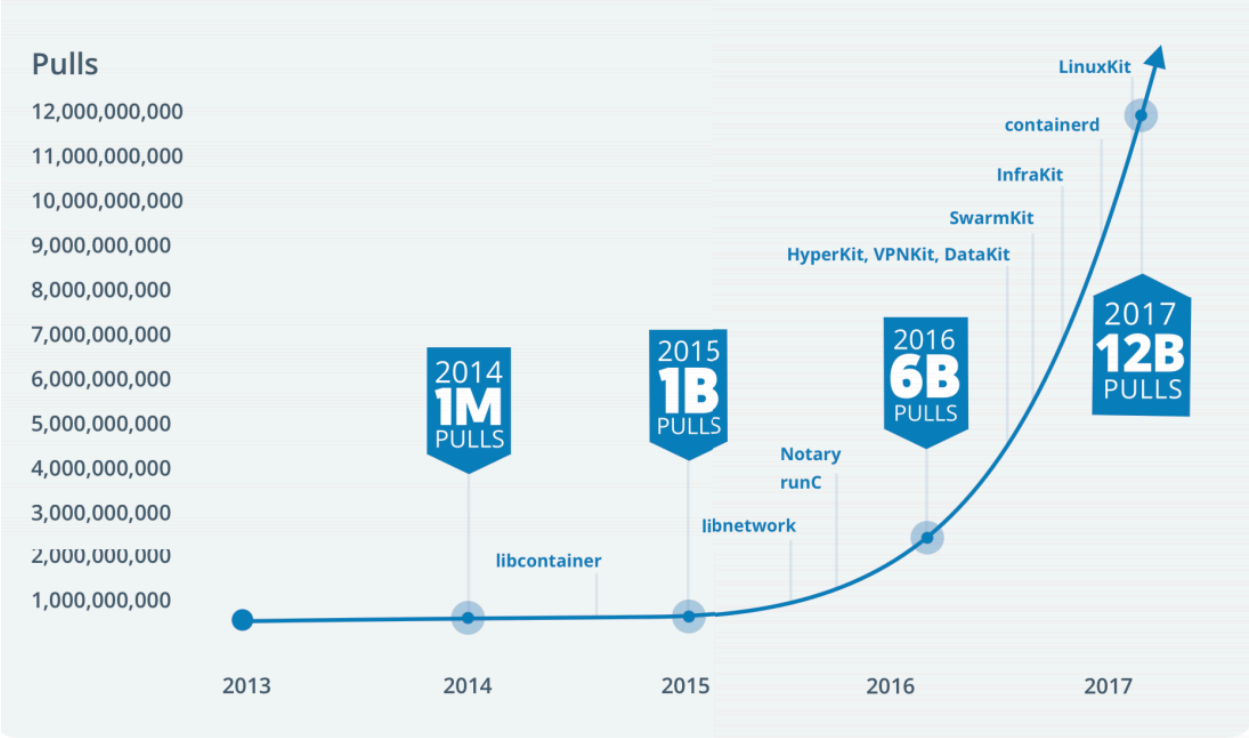


Figure 2.4.1: The growth of popularity of Docker [92]

Nevertheless, the growing need for software portability, high performance, and reproducibility from the development environment to the production environment made it critical to evaluate the software application while being executed using the Docker container. Docker containers are lightweight as they can be launched when needed without a guest OS as they share the host OS kernel, unlike the virtual machine (VMs) as depicted in the below Figures 2.4.2 and 2.4.3. Docker is the leading container solution to isolate the runtime environment from the underlying host and networking resources [22]. Docker containers provide a wrapping solution for deploying and shipping software applications.

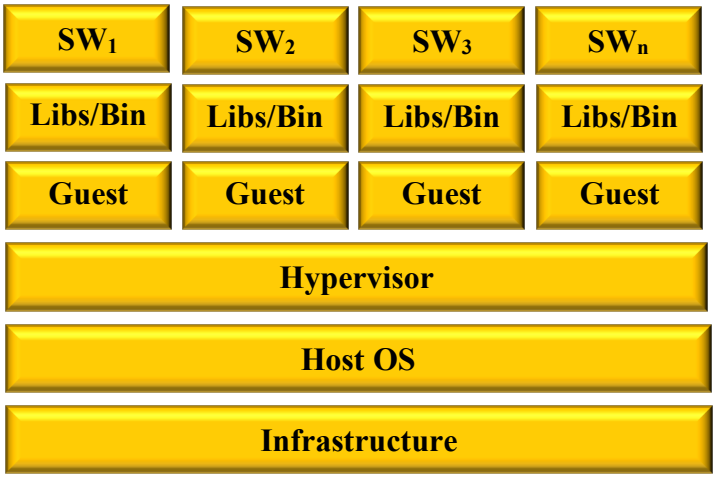


Figure 1.4.2: Virtual Machines

Containers and VMs have some common functionalities of resource allocation and encapsulation, but they use different architectural approaches. Containers are more efficient, scalable, portable, and easy to deploy compared to VMs [31, 32] as Docker uses a client-server architecture. Docker daemon does the heavy work of creating and managing the Docker containers. The Docker client and Daemon Docker engine manages resource utilization more

efficiently as it uses centralized resource management within the created containers. On the other side, VM's hypervisor allocates the maximum limit of resources to each virtual machine created while Docker containers dynamically allocate the resources, e.g., memory, processors, and page cache, at runtime [33].

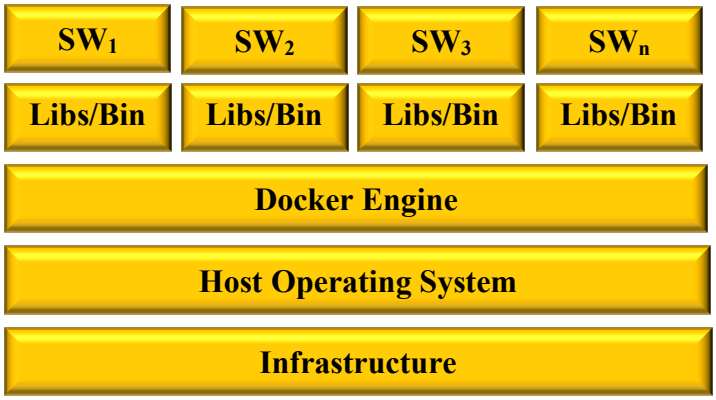


Figure 2.4.3: Docker Architecture

Docker makes installation much easier and unlimitedly replicable as proved in AEIPA system in [23]. Docker packages deployments into Docker images with all the required libraries and dependencies. These Docker images can be reused as a base Docker image to other Docker images without the need to start a new one from scratch. The contents of a Docker image are defined online in Docker Hub. Hence, Docker images can be pushed to and pulled from Docker Hub without cluttering the hosting machine with lots of files.

2.5 Related Work

Here is some of related work that the proposed system can be compared to. In this work we will be comparing the proposed system with one of the famous automation testing software (Selenium) and the continuous delivery system (Jenkins).

Selenium is an open source project that automates web application testing using the Selenium WebDriver [93 -96]. It is a browser automation client library. Users write the testing scripts using various languages like C#, Python, Java, etc. Selenium does not provide any facility to deploy the application being tested but it is used widely for automating the execution of test cases.

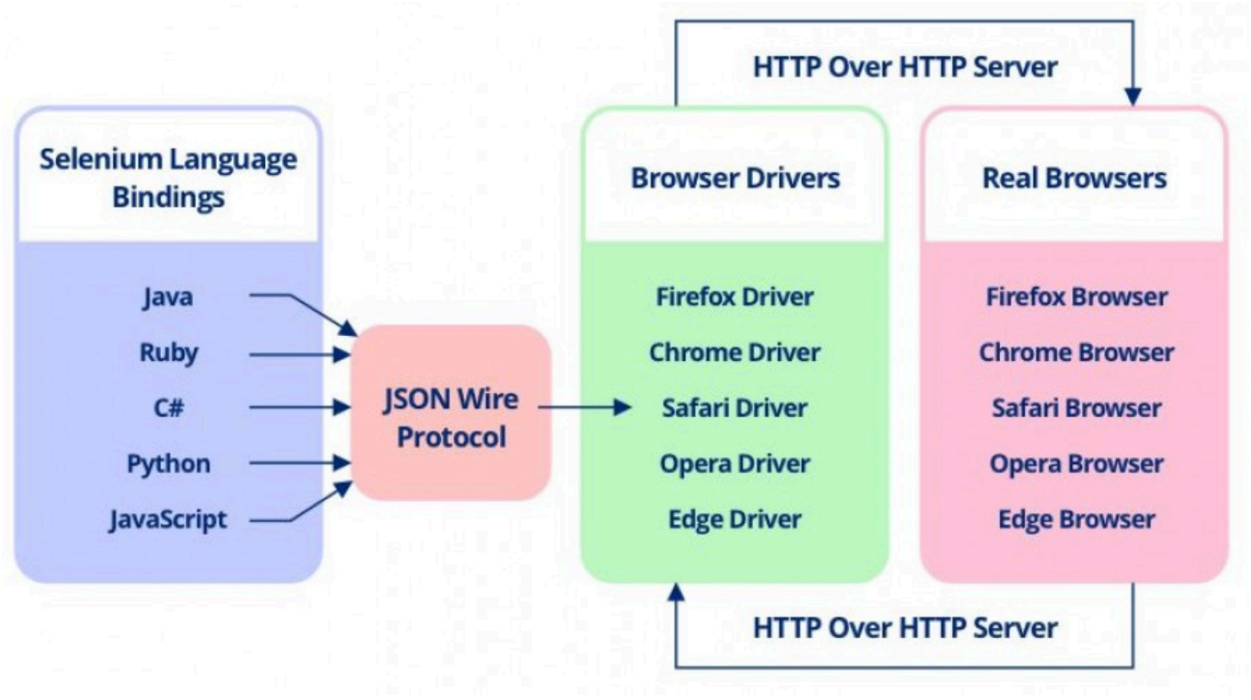


Figure 2.5.1: Selenium WebDriver Architecture [101]

A major limitation in Selenium is that it is used for automating the interaction with web browsers. Selenium cannot be used for testing desktop applications. Testing scripts in Selenium are limited to only Python, C#, Ruby, .Net, PHP, Java, and Perl. Users are limited by the programing languages that Selenium supports. Another major limitation is that it can't perform testing on images [97 - 100]. Another important point is that ADESA framework cover comparing the results of many software algorithms which cannot be done using Selenium framework.

Jenkins is an open source server that automates parts of the software life cycle like ADESA. Jenkins allows the developers to build their software, run automate tests and deploy their software [102]. Jenkins is simply a task executer [103]. It is capable of artifact packaging, code compilation, and executing automated test cases as shown in figure 2.5.2. A major problem is that Jenkins doesn't actually fully automate the deployment of the software applications. The users have to write and build their own software to deploy their software [104]. They have to write custom deployment script to take care of the deployment part while ADESA system already covers the deployment task from the beginning.

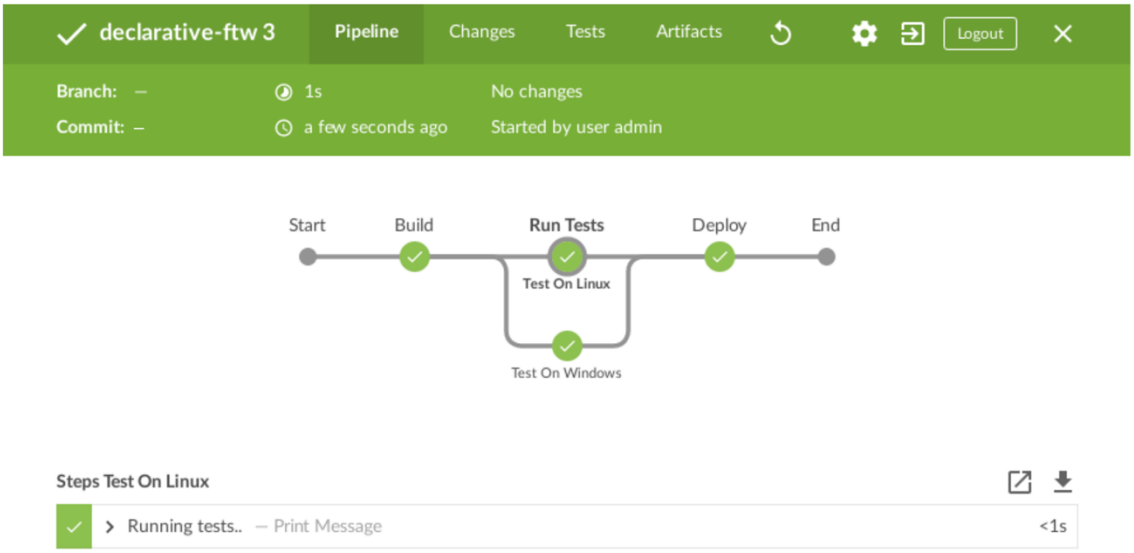


Figure 2.5.2: Jenkins Pipeline stages [105]

ADESA framework on the other hand covers all the processes related to releasing a software application from building the working environment, installing the needed programing languages and dependencies, executing, evaluating the algorithm by automating tests, and finally

comparing the results against other software. This full coverage focuses on a rapid continuous integration and continuous deployment (CI/CD) pipeline with a high-quality result after comparing the results with the other software's results. In addition, the way ADESA executes each input independently makes it easy to host the algorithm execution on many servers. A Docker container is constructed per each input which gives the ADESA a huge scalability advantage compared to the other systems.

Chapter 3

ADESA

This chapter explains in detail the proposed ADESA framework, the proposed system's architecture, and the use of the Model-View-Controller design pattern.

3.1 Proposed System

This research focuses on developing a framework for automating the deployment and evaluation processes of software applications, ADESA. The goal of this research can be divided into two parts: first, automating the creation of the working environment, deployment and execution of software applications using input data instances, second, to automating the evaluation of the executed software application and compare it with other software applications in the same category. Thus, five elements are the key components of the proposed system: input data instance, software application's execution, the output of the software's execution, evaluating the performance of the software, comparing the software with other software and representing the results visually. The remainder of this chapter will present each of these elements, within the two research goals, in addition to the design decisions that fit them.

3.2 ADESA Framework Architecture

This section presents an overview of ADESA's framework design. Design diagrams are provided focusing on the interactions between the modules. This framework is used to automate the deployment and evaluation processes in software engineering. The ADESA framework is divided into four modules: Input Supplier, Software Factory, Data Repository, and Docker engine as in figure 3.1.

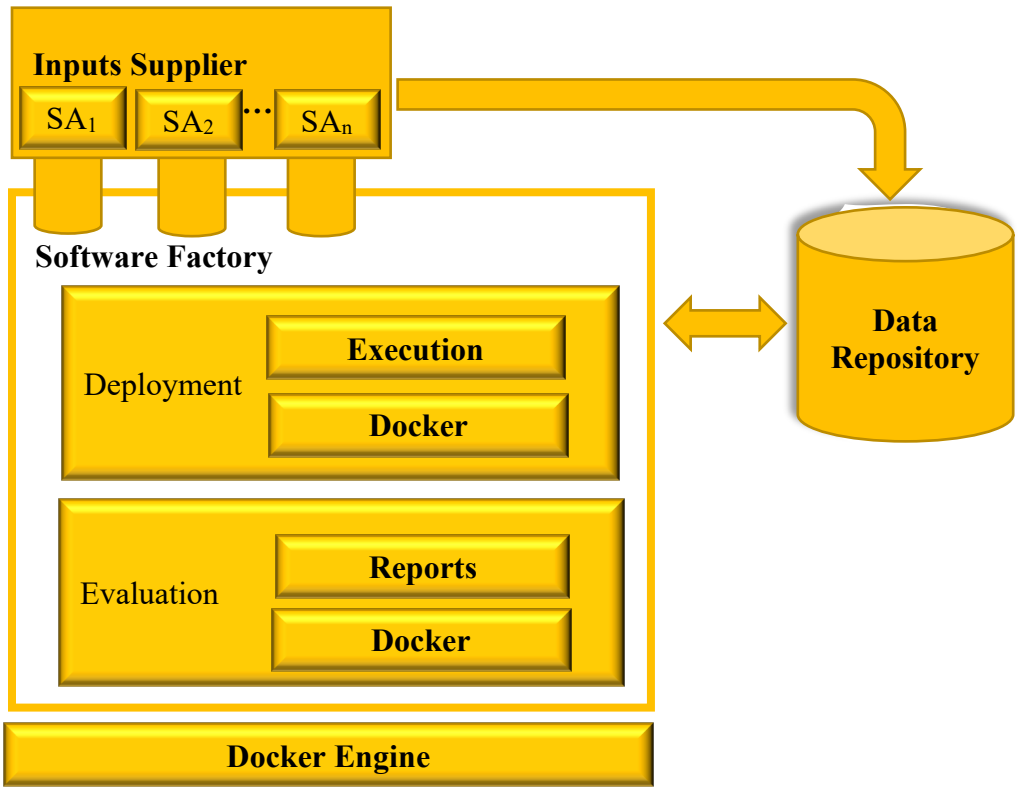


Figure 3.1: ADESA Framework Architecture

The Input Supplier is responsible for converting the inputs from the user to ADESA in a readable format and stores the inputs into the Data Repository. The execution of the software under evaluation is being controlled by the Software Factory module. The Docker engine is used in order to execute each of the software instances in a separate environment according to the programming language that the software is written with, the operating system, and libraries. A report is generated with the evaluation metrics plus a comparison with other software applications while and after the intended software application is being executed. The Data Repository is where all the input instances, software source code, execution information, and execution results are stored. In the upcoming sections, a detailed description of each module is presented.

3.3 Model-View-Controller Implementation

Software applications have to interact with one or more objects in order to be useful. Software applications can interact with users, machines, or other software applications. That is why it is important to invest effort on the software's interfaces that interact with the software's outer world. It is likely to change the software's interfaces while keeping the underlying application constant. It is also reasonable to keep the essence of the software application separate from the interfaces. To cope with that, the Model-View-Controller (MVC) architecture has been used in the ADESA framework [48]. This design pattern is very useful for designing interactive software applications as it is based on partitioning the application into independent layers [49].

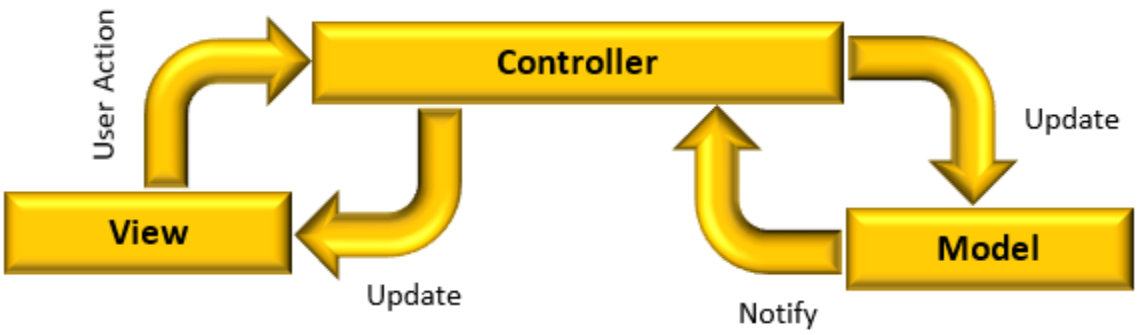


Figure 2.2: MVC Pattern

MVC architectural pattern divides the software application into three interconnected layers: Model, View, and Controller layers as in Figure 3.2. This design pattern is commonly used for implementing applications with user interfaces. It separates the business logic representation of the application from how this business logic is represented to and accepted from the user. MVC focuses mainly on the purpose of each section of the code. Some of the code is responsible for holding the data of the application's business logic. Other sections of the code are responsible for the interactions with the user. Some sections of the code control how the application should function [45 - 47]. The big idea behind using MVC is to organize the ADESA's framework architecture into neatly organized boxes as shown in Figure 3.3. MVC helps in scaling and reusing ADESA easily and cleanly.

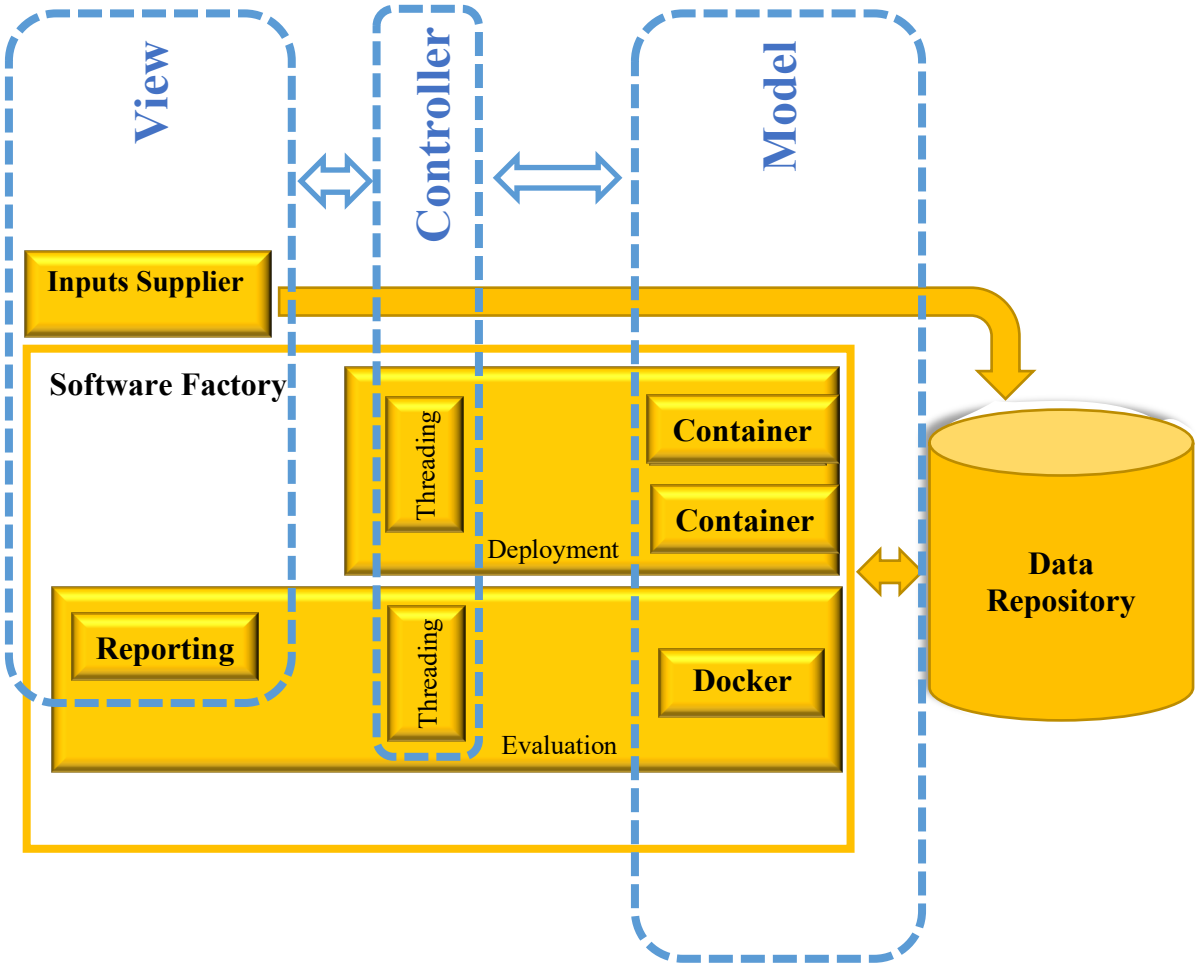


Figure 3.3: ADESA Framework Implementation Design

Model Layer

The Model layer is the unchanging essence of the software application. It is the set of classes, in object-oriented terms, representing the core problem. The Model layer should not interact with the outer world, i.e., the Model layer should not interact with the user, or any other applications [50]. All the interactions should go through the Controller Layer and Database storage [51]. This layer encapsulated the data model of the system. It maps the business logic of the system into representable data entities that are called Data Transfer Objects (DTOs).

The Model layer in ADESA is divided into two layers: Data Access Objects (DAOs) depicted in Figure 3.5, and Services depicted in Figure 3.6. DTOs are represented in Figure 3.4 encapsulate data in a value object that can be transferred over classes and modules in ADESA. Figures 3.7 and 3.8 shows the interconnection relationships between DTOs and DAOs, and DAOs and Service layers, respectively. The DTOs are the objects representing the software's data, and they only contain private fields for the data, getters, and setters. There is no business logic in the DTOs. DTOs are responsible for transferring the data between separate layers [52, 53]. DTOs are responsible for transferring data between the DAOs layer and Data Repository storage, the DAOs and Service layer, and Service and the Controller layers.

The DAOs layer maps application's calls from and to the database without exposing the details of the database, that is why DAOs are considered the interfaces to the database [54]. This layer encapsulates the logic for retrieving, saving, and updating data in the Data Repository. When the business logic is complicated and expected to grow, it is preferred to create a Service layer. It abstracts data access and business logic [55 – 57]. It is important to note that the Service layer is driven by the software's use cases.

For more detailed descriptive systems, two different systems that apply the mentioned layers above are shown in [58] and [59]: Hadoop MapReduce and PolarHub, respectively. Hadoop MapReduce is a data distribution architecture for processing input splits while PolarHub is a large-scale search engine that is implemented to discover the distributed geospatial services and data. PolarHub is developed using Service Oriented Architecture (SOA) and Data Access Objects (DAOs). The two systems show sustainable, high performance, scalable, and interactive platform. As shown in those two systems, it is a good idea to have the DAOs and Service layers when the

business logic of the system is more complex than the data logic. The service layer implements business logic by implementing each use case. This layer handles all the validations before calling a method from a DAO object. The DAOs layer defines the operations to be performed upon DTOs. In the mentioned systems, there is DAO for each entity (which is a DTO). The DAOs Layer implements additional business operations above the create, read, update, delete (CRUD). In ADESA, GenericDAO interface is defined (see Figure 3.5) with sets of type-safe operations for each DAO. The Services layer depends only on this GenericDAO, and that is how the DAOs and Services layers are separated as depicted in figures 3.8 moreover, 3.9

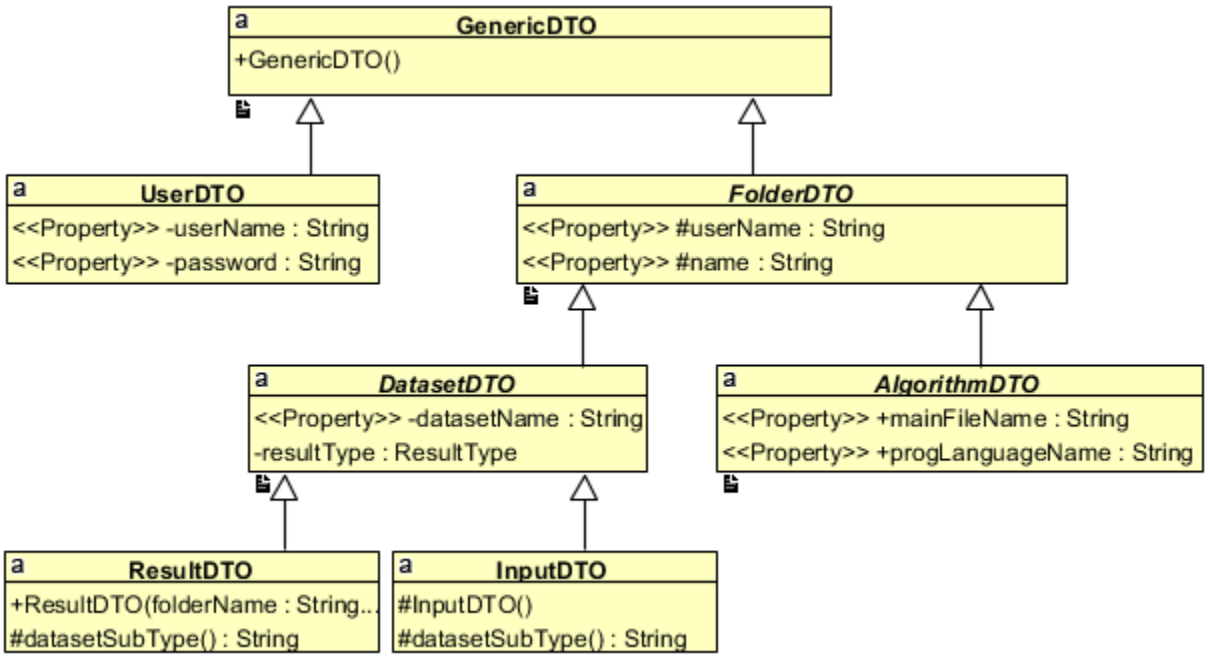


Figure 3.4: ADESA DTOs in Model Layer

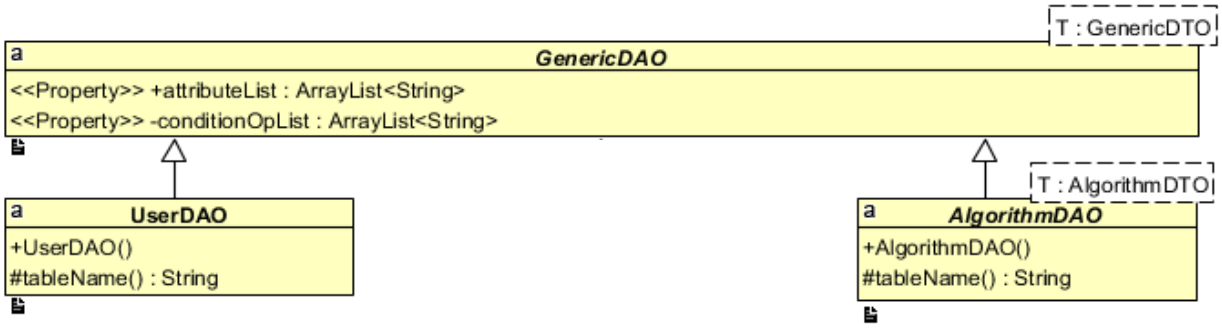


Figure 3.5 ADESA DAOs in Model Layer

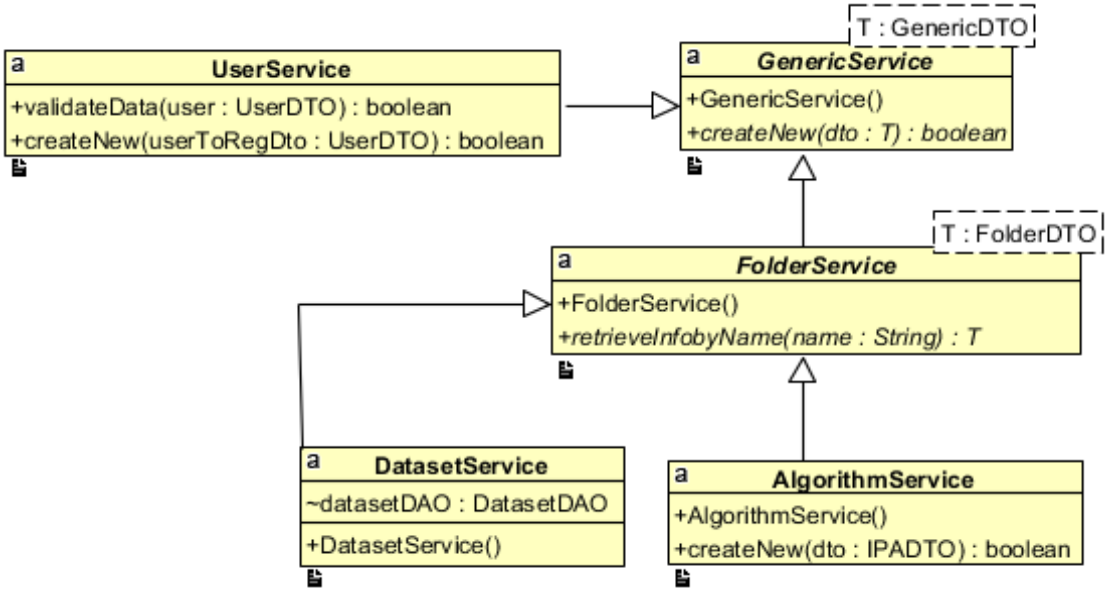


Figure 3.6: ADESA Service in Model Layer

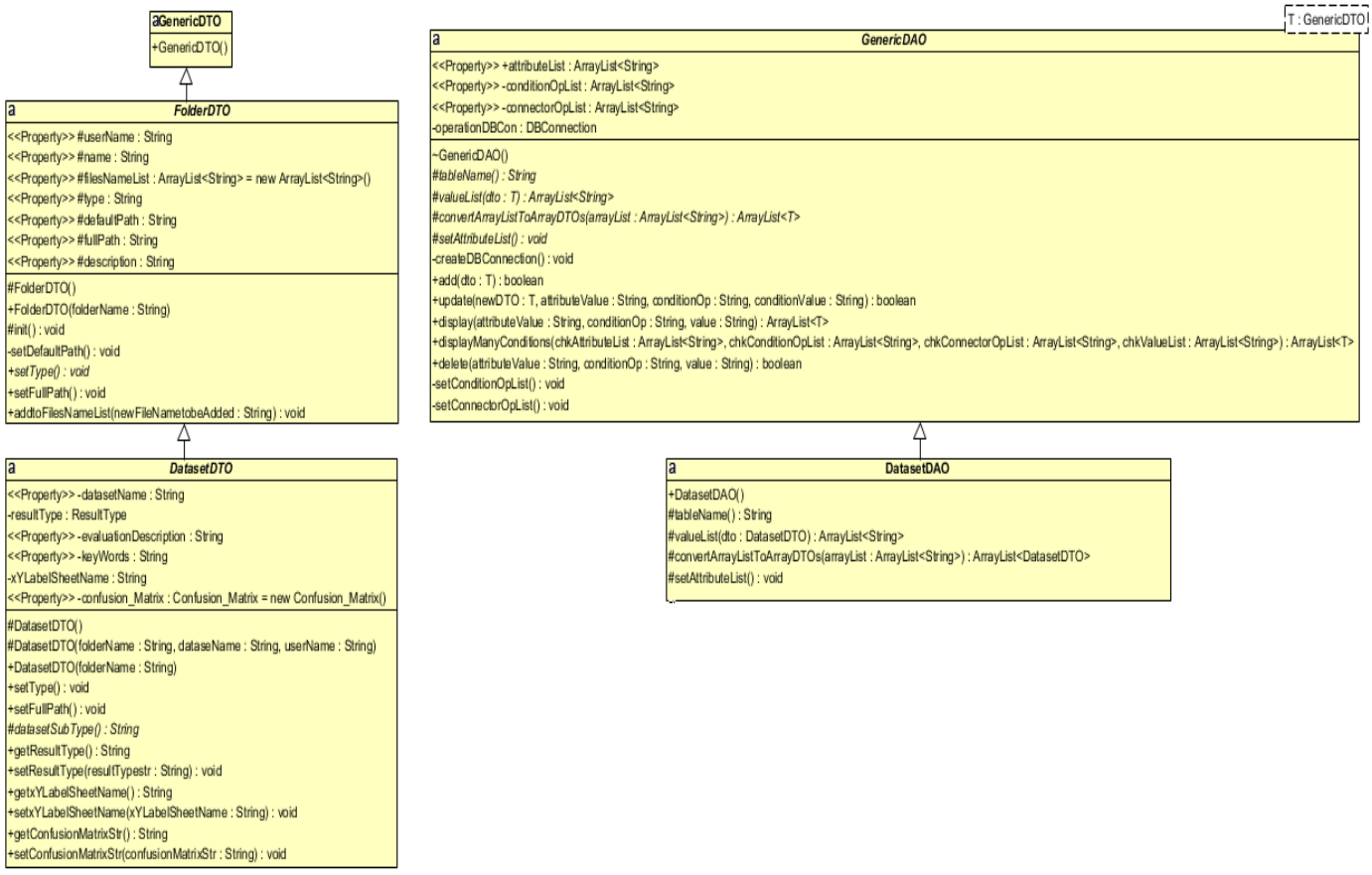


Figure 3.7: ADESA Dataset DTOs-DAOs relationship

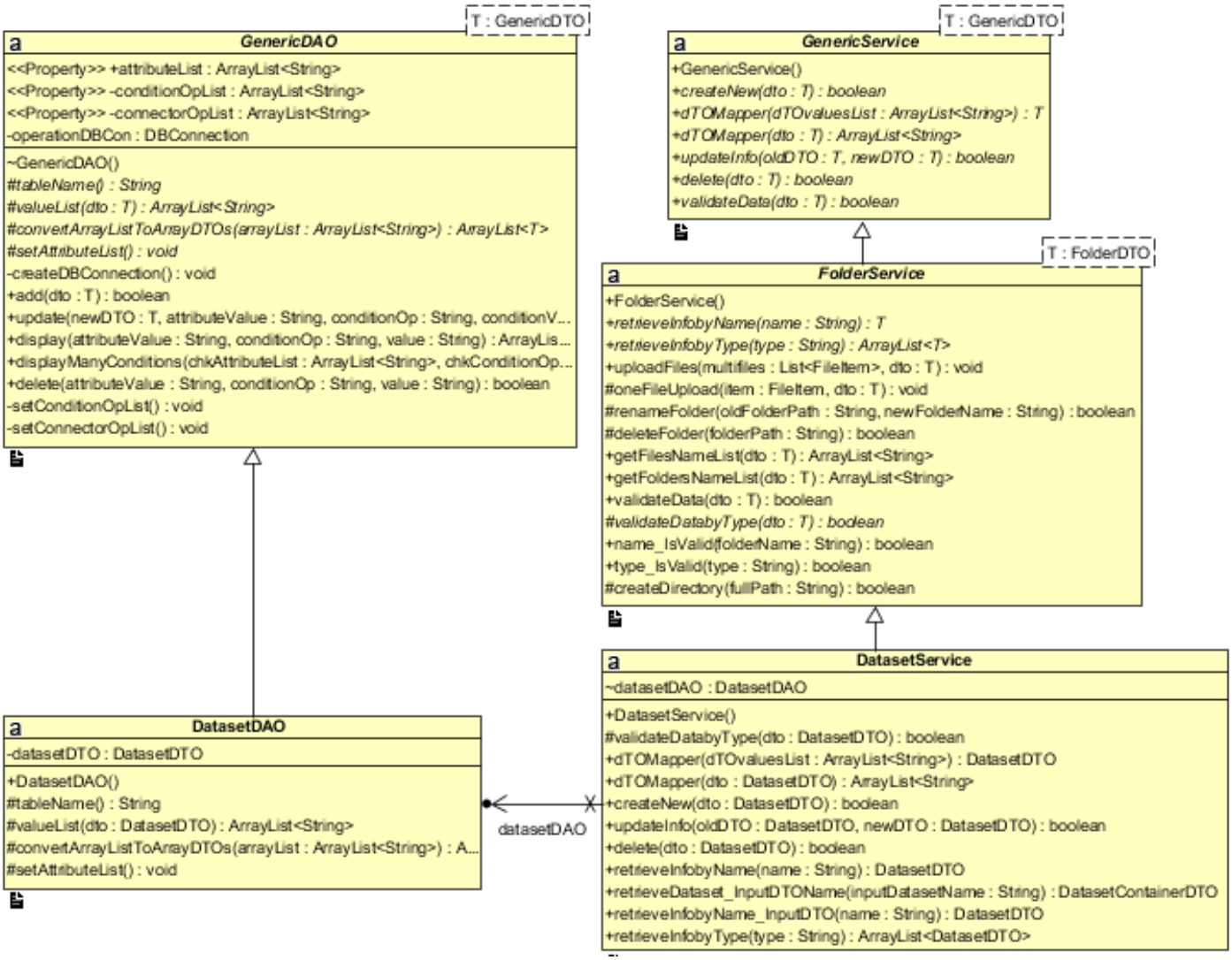


Figure 3.8: ADESA DAO-Service Layers relationship

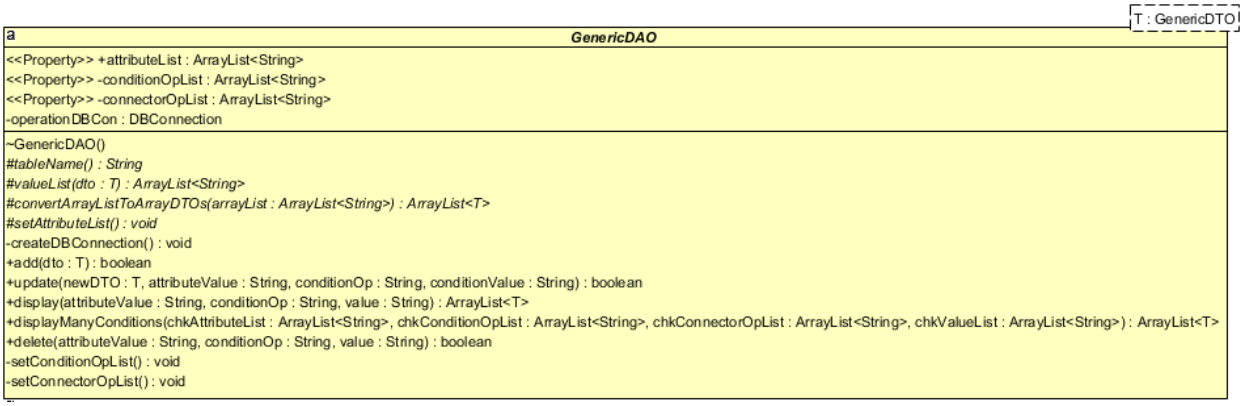


Figure 3.9: Data Access Objects

View Layer

The View layer is the visual representation of the problem to be solved using a software system [62]. It can be in the form of an application program interface (API), a graphical user interface (GUI), or a command-line interface (CLI). The View layer consists of the classes that are key to presentation and interaction with users [63 - 65]. A View layer can be as simple as an input form, a report sheet, a comparison table, or a graphical view [66].

ADESA is a shared system for software developers, researchers, and students. That is why ADESA has been implemented as a web-based system, and the View layer is responsible for translating information in HTML to be displayed by a client-side web browser. The View layer is responsible for collecting all the required inputs from the user and displaying the results back to the user after processing. It collects the software application's source code, the used programming language name, and the version number in order to create a Docker container with the correct Docker image, the inputs to the software application, and the evaluation type in the form of HTML pages and JSON responses. The output is the execution results, files, graphs, and tables back in the form of HTML pages and JSON responses again.

Controller Layer

The Controller layer ties the interactions between the View and Model layers. A controller is an instance that enables processing of the View's input and delivers the results to the View's output. Also, the Controller layer has information about the operating system and used platforms of the web application being developed [67 - 70]. There is a separate Controller for each user scenario in order to be handled using Services in the Model layer. Moreover, the Controller manages the execution of Threading when needed, which is the case in ADESA.

ADESA and MVC

As was shown in Figure 3.3, the MVC design pattern is used in ADESA. The four modules that will be explained in the upcoming sections are divided by the MVC layers. The Input Supplier and the Reporting modules interact with the users and are implemented totally in the View layer. Though, all the servlets controlling the interactions between the View layer and the business logic (Model layer) have been implemented in the Controller layer. The Controller layer maps the processing from user inputs to the deployment module, then to the evaluation module, and finally back to the user's output results (Reporting). The Model layer contains all the business logic of ADESA. It is divided into the three layers explained above: DTOs, DAOs, and Service layers.

3.3.1 Input Supplier

ADESA framework provides the user inputs through the Input Supplier. Any software program takes a set of input instances and produces a set of output values. In ADESA, the input can be the input data set, software algorithm's source code, evaluation criteria, groundtruth, and any other software algorithms to be executed and compared to. Input data can be anything the software application would need in order to start its processing. All the inputs are uploaded and stored to

the Data Repository. As an example, consider a problem of counting the number of occurrences of a word in a document. Pseudocode to solve this problem would be something like the one below:

```
word_count(String content, String key):
// content: the contents of the document
// key: word to count in the document
int count = 0;
for each word in the content
   if the word is equal to key
        count += 1;
   endif
    print count
```

The word_count function counts the number of occurrences of the second input parameter key in the first input parameter content. The inputs to such a problem would be a string of the word to count, the document, and the source code to the program that solves this problem, the programming language's name and version, other programs that would solve this problem, and the evaluation program. The inputs to this software algorithm can be easily determined in discussions with the users of the software at an early stage of the development cycle [25]. The needed information to be used by a software application is set from the user's external view, mostly in the requirements gathering activity in the SDLC [12].

ADESA applies the open-closed principle that it can be extended to support any input format [28]. ADESA users supply the needed inputs, and ADESA delivers the provided input with its interrelationships and representations to the Software Factory Module, i.e., the deployment

automation. ADESA user uploads the file that contains the input and ADESA's Input Supplier stores all the provided data into ADESA's Repository. ADESA inputs and results can be in all native and extended data types: boolean, byte, unsigned byte, short, unsigned short, integer, long, float, double, RGB, complex, and any other type in a file format. The provided software algorithm can be written using any language supported by either Docker Hub or directly by AEIPA, such as C/C++ (GCC), Java, PHP, Python, Hy (Hylang), Go (Golang), Node, Perl, Rails, Clojure, and Ruby.

3.3.2 Software Algorithms Factory

In this section, an overview of the core module of ADESA is presented in detail. The Software Algorithms Factory consists of two major submodules: Deployment Automation and Performance Evaluation Automation. Deployment Automation is responsible for automating the algorithm's installation and execution in ADESA. Performance Evaluation Automation is concerned about executing the performance measurements associated with the algorithm and comparing the evaluation results with other algorithms in the same algorithm's software category.

3.3.2.1 Deployment and Execution Automation

Deployment Automation submodule is responsible for creating a working environment inside a Docker container. There is a list of already prepared unique Docker images that creates Docker container(s) with specific dependencies, e.g., OS, programming languages installed, and specific libraries installed. The Docker images can be prepared and customized manually or can be downloaded from Docker Hub that contains a very wide range of Docker images supporting almost all the existing programming languages.

When the Input Supplier module provides the software algorithm and specifies parameters like the name and version number of the programming language it was written by, Docker containers are created using the Docker image that matches the user's specified parameters. Programming languages can be Java, C/C++, Python, PHP, Hy (Hylang), Go (Golang), Node, Perl, Rails, Clojure, Ruby, Matlab and OpenCV [27]. Programming languages version number can be Java 7, Java 8, Python 2.7, or Python 3.6.5., ...etc. Unique Docker images are prepared to support different programming languages with different versions.

The Input Supplier module supplies the plain source code file(s) of the software algorithm, the related parameters as mentioned previously, the input data, the and the evaluation criteria. Then in the Deployment Automation submodule, a Docker container is created for each execution of an algorithm with one input data. Since the program execution capacity is determined by the data that is processed by the program [26, 29], separating the execution of each of the input instances downsizes the execution bulk, and most importantly it helps in the mobility of spreading the execution of a single algorithm across a distributed system. The software algorithm gets executed inside the matched Docker container, and performance evaluation is processed during and after execution. The algorithm is executed along with the benchmark algorithms in separate Docker containers and in parallel. The execution results are stored in the ADESA Data Repository for evaluation.

3.3.2.2 Evaluation Automation

Software algorithms generate different output types based on the category of the algorithm. Some algorithms may produce numeric values, while others may produce visual results such as digital images and videos. In order to evaluate the efficiency of the algorithm and

compare it accurately to other algorithms, quantitative results are required. The evaluation of a software algorithm is a productive and formal procedure to measure the work and results based on the algorithm's execution. The evaluation submodule in ADESA examines automatically the effectiveness of the software algorithms being executed. The evaluation is measured according to a set of standard evaluation criteria depending on the category of the software algorithm. In order to evaluate the performance of a software algorithm, certain things need to be identified like: what to be accomplished by executing the algorithm, how it can be effectively measured, and what data instances can be used. All these points are defined to get the most benefit of executing the software algorithm.

There are many techniques for evaluating software algorithms. The main three techniques addressed, as mentioned before in this work are correctness, accuracy, and performance. Every input to the software algorithm should generate the correct expected output in order to evaluate the correctness of that algorithm. In order to save time, more sophisticated techniques try to select the inputs so that every, or at least the majority of the possible execution paths are tested. Though, these empirical methods do not indicate that the software is correct. The accuracy of the algorithm is measured through the error rate, specifically, errors found in outputs are collected and calculated proportionally to the total number of the executed inputs. Performance evaluation determines the stability, reliability, responsiveness, resource usage, and scalability of the software algorithm. In other words, performance evaluation determines whether the algorithm meets the expectations under workloads or not. ADESA implements the template for the users to add the evaluation algorithms as it is explained thoroughly in AEIPA's implementation (for example confusion matrix). Depending on the algorithm category and the

user's selected way of evaluation, the evaluation algorithm executes. Also, depending on the type of the evaluation algorithm, it gets executed during, or after the algorithm's execution. For example, the performance evaluation is executed during the algorithm's execution while the accuracy evaluation executed after the algorithm is done execution.

After the evaluation is completed, evaluation values are stored in the Data Repository and passed to the Reporting phase. The reporting submodule is concerned with representing the output results graphically to the user through the View layer, as explained previously. Data representation can be in any form such as comparison tables, statistical plots, resources consumption charts, performance graphs, pie charts, ... etc. ADESA framework contains the foundation to add new reporting. Users of ADESA can create new reporting types and use the evaluation results saved into the Data Repository.

3.3.3 Data Repository

The Data Repository is the storage database for ADESA framework. This module has been implemented as a generic abstraction, as in Figure 3.10 connecting to any database, retrieving, adding, deleting, and updating the data.

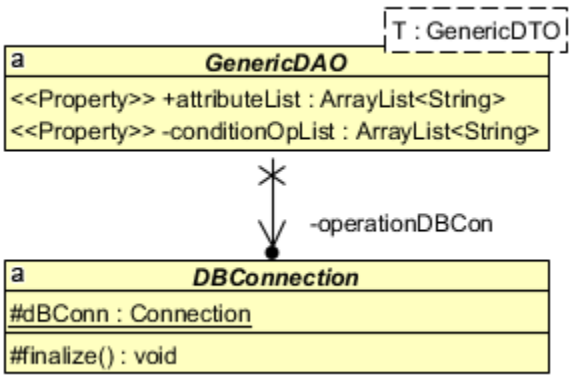


Figure 3.10: ADESA Database Repository

The DAO layer is the interface to all the database operations, though database details are hidden from the DAO layer. DAOs provide the operation type to be performed on data in the form of DTOs. This helps in the maintainability of the database separately from the Model layer. So, any changes to the database will not affect the business logic of ADESA and vice versa. The Data Repository stores all the data of the users, algorithms' source codes, inputs, and outputs from each execution, evaluation types, and evaluation results.

3.3.4 Docker engine

Software application life cycle is subject to frequent changes and updates, thus raising many deployment and maintainability issues [34 - 36]. Research applications, in particular, rely on incremental software development because of their experimental nature. These software applications are implemented with many implicit dependencies on libraries, programs, operating system, hardware specifications, and other components. As a result, these applications are difficult to install, configure, and deploy. Moving a software application from one environment to another environment used to have a little chance of running correctly without a significant effort. VM technologies were used as a solution to this problem, but they had some disadvantages due to their need to create a complete copy of the OS, as explained in the previous chapter. Besides, it is impossible to reuse piece(s) of data or software inside a VM because VM's contents are not accessible with a standard protocol [37, 38].

Meanwhile, Docker has recently gained an increasing level of attention because it allows software applications to be executed in an encapsulated and portable package that can be distributed across computing platforms, as shown in figure 3.11. Docker is used in the ADESA

framework to execute each software algorithm in an isolated container that is created from an immutable Docker image. Docker images in ADESA are predefined for constructing a wide range of ready-to-work environments. Each Docker image in ADESA is a unique set of instructions to build a prespecified working environment. For example, there is a Docker image for building an environment with Ubuntu 18.04, SSH server to easily access the container, runit that replaces Ubuntu's Upstart, install_clean for installing apt packages, and Java 8.181 for executing an 8.181 Java software algorithm. Another Docker image would be Ubuntu 16.04, SSH, runit, install_clean, and Java 7.67.

The Input Supplier in ADESA framework provides the software algorithm in a source code format in addition to the environment's specifications. The Docker image that matches the provided environment's specifications gets deployed and generates a Docker container. The Docker container establishes a ready environment as specified by ADESA Input Supplier.

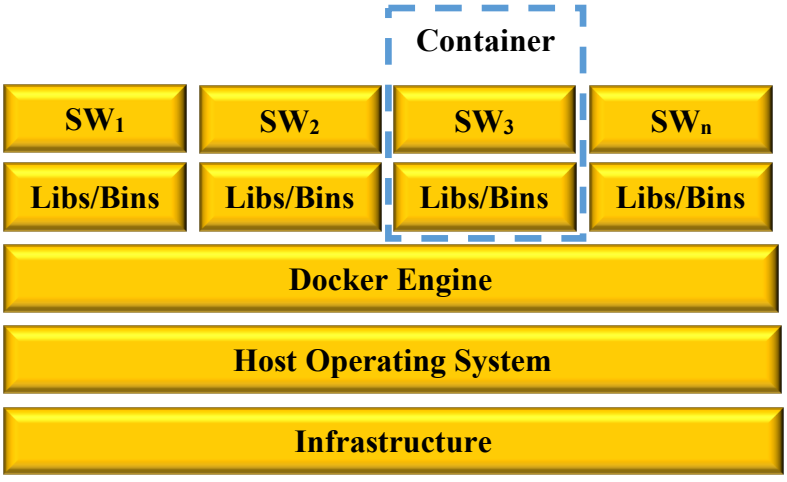


Figure 3.11: Docker containers and Software Applications

Docker containers created by ADESA's Docker images are prepared to execute the source code of a software program. There are many reasons to use Docker containers for executing software algorithms:

1. Using Docker containers guarantees that each software application runs in a predictable system configuration as all the configurations, dependencies, and libraries are configured internally.
2. Docker containers are isolated, providing an additional layer of security and preventing conflicts with other installed programs in the hosting environment.
3. Many instances of Docker containers can execute in the same hosting environment since they are lightweight processes, sharing the kernel with other programs.
4. Simplicity and maintainability since less time is required to add, update, or delete an environment using Docker.
5. Rapid deployment as Docker succeeds in reducing the deployment to seconds due to creating a container for every process without booting an OS.
6. The portability of the Docker paved the way for cloud computing providers to support Docker containers
7. Docker grants complete control over traffic flow, security, and management because applications running on containers are completely segregated and isolated.

Chapter 4

Image Processing Algorithms Case Study

Image processing is used by a broad range of applications that need computers, digital cameras, videos, and images for converting and processing data into meaningful information. The human eye can easily distinguish significant characteristics even in poor-quality images. Image processing applications aim to interpret images in a similar, or even faster, way compared to a human being. Image processing applications can powerfully manipulate images, quantify intensity, apply mathematics, detect edges, and enhance the contrast of images. Image processing is a dynamic research area that is being continuously investigated. It is involved in many real-life applications, such as traffic monitoring [39], quality inspection [40], automobile parking [41], and human identification [42], to name a few. Such applications add many challenges which require a rapid evolution of image processing algorithms (IPAs).

When image processing researchers, who are the main users of AEIPA, develop a new IPA to solve a specific problem, they find many approaches published. In order to reach better results in the new IPA being developed, they need to compare its results among other IPAs that are solving the same problem (which are called benchmark IPAs) [60]. Such an objective evaluation can be reached using ground truth through publicly available datasets (which are called benchmark datasets) like in [43]. Although there are online repositories that contain the implementation of benchmark IPAs and benchmark datasets, researchers get overwhelmed with configuring these benchmark IPAs and datasets to make them ready for use and facing all the troubles of customizing them according to the working environment. Researchers put too much

effort into setting up the environment for evaluating the new IPA, and they have to go through all the following steps:

- Installing all the related benchmark IPAs, and sometimes it would require implementing the benchmark IPAs
- Setting up benchmark datasets to be ready for use
- Creating new datasets to test different challenges
- Evaluating each of the related benchmark IPAs individually
- Reporting the results which usually needs writing scripts/code manually

In this work, a web-application implementation of ADESA framework is presented that aids researchers in automating the deployment and execution of IPAs and provides an evaluation of the outputs of the algorithms objectively. For additional quantitative evaluation, a comparison between the evaluation results of all the IPAs, with the same functionalities is presented in a report to the user.

4.1 AEIPA Prototype

AEIPA (Automated Evaluation of Image Processing Algorithms) design is presented in this chapter to solve the overhead that was mentioned above. AEIPA is an implementation of the ADESA framework. It allows automatic execution of IPAs, evaluation of the results, and comparing the evaluation results to already existing benchmark IPAs. AEIPA is structured into three architectural modules: AEIPA Supplier, AEIPA Factory, and Docker engine. AEIPA uses Docker containers to allow agility and efficiency of the continuous integration and deployment activities. To the best of our knowledge, AEIPA is the first system to solve these problems using Docker.

This chapter is arranged as follows: Section 4.2 gives the motivation of implementing the AEIPA system. Section 4.3 illustrates the datasets used for executing and evaluating the IPAs. Section 4.4 explains AEIPA's requirements' document. Section 4.5 explains AEIPA design. Finally, section 4.6 explains AEIPA's user manual in detail.

4.2 Motivation

Evaluating a new IPA is a time-consuming task. The user must develop a new IPA to solve a specific problem, then search for the matching datasets and ground truths, then set up the environment to execute and evaluate the IPA, then search for all the related IPAs with the same functionality, implement them, set up the environments to execute and evaluate each of them individually, finally, compare all the results together. Since image processing has been involved in most of the daily life applications, the need to automate all these steps has emerged.

The major concerns that were faced during thinking about the solution were the extreme variety of image processing categories, the IPAs' different programming languages, operating systems, and libraries. The first problem was setting up a working environment using a combination of all of these factors, which complicated the automation of deployment and execution processes. The other problem was the complicated ways for evaluating software algorithms. Each IPA's category requires a specific way of evaluation. For example, the image filtration algorithms generate filtered images while detection algorithms generate labeled bounding boxes (either on the image or in a separate file). Both of these two image processing categories require different types of evaluation. The image filtration results need to be analyzed and evaluated differently than calculating the accuracy of the detection algorithm according to

the ground truth. Also, the evaluation algorithms are implemented using any programming language according to the user's knowledge.

Consequently, there was need of a unified system that would accept all these varieties and make it easy for the user to deploy and execute a new IPA, find related IPAs, execute them all against some specified datasets, and compare the results altogether. The system should be shared with all the users: software developers, researchers, and students in order to make the most benefit of analyzing the algorithm and compare it against other algorithms solving the same problem. All the datasets, IPAs, evaluation algorithms, and evaluation results are stored in AEIPA.

4.3 Datasets

A Dataset (or data set) is a group of related data that corresponds to a particular event or experiment. Datasets mentioned throughout this thesis refer to image datasets that are collections of images representing separate images or a sequence of frames. Datasets are huge due to the number of images\frames per each dataset as in images of face detection image processing applications. The analysis and processing of such huge datasets need complicated automation tools. Image processing tasks used to be expensive and require specialized UNIX workstations [44]. Today image processing tasks are performed on inexpensive working environments equipped with the appropriate libraries and dependencies in order to be able to process the datasets. Dataset's type vary according to the functionality that it is being used for and the type of device that it was captured from. As examples, there are datasets for human faces that are used for facial regions detection algorithms, datasets for detecting moving objects, datasets for silhouette detection for human and animals, and videos datasets for video understanding research to name a few. Each dataset has ground truth data that is the

information collected from industry to identify how the dataset achieve the best results. Every Input Dataset in AEIPA must have ground truth data.

4.4 System Requirements

AEIPA is a common web-based application used for automating the execution and evaluation of the IPAs. Using the proposed system, users can find all IPAs that are relevant to the problem field, execute them, evaluate them, and compare the evaluation results against other IPAs. Users do not have to worry anymore about the algorithm's dependencies (execution environment, or the programming language that was used in developing the IPA, libraries, dependencies … etc.). Users only need to focus on developing their own IPA and upload it to AEIPA web-application in order to be executed and evaluated.

The expected users of AEIPA are researchers, software developers, and research and development (R&D) developers. The system can be easily used by any users as there is no need for any prior knowledge about IPAs' used technologies, working environment, programming languages, implementation details, or Docker technology. Users do not need to worry about how to execute and evaluate the IPA and compare it to other IPAs as detailed in the next section of the user stories.

4.4.1 ADESA's Use Cases

User stories are informal description in natural language of how ADESA's features will act.

ID	Use Case Name	System Scenarios
000	**User registration**	New users should be able to register to AEIPA with the below data, as shown in Figure 4.1: ▪ Username • A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. • This field should be unique across all the users in the system. ▪ Password • A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. • Typed letters should be hidden to the user. Letters can be replaced by asterisks or dots for security proposes. ▪ First Name • A mandatory field. A maximum of 45 letters is only accepted. Numbers and special characters should not be accepted. ▪ Last Name

		• A mandatory field. A maximum of 45 letters is only accepted. Numbers and special characters should not be accepted. ▪ Job • An optional field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. ▪ Organization • An optional field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. ▪ Date of Birth • An optional field. A valid date can be only accepted.
001	**User Login**	Logged in users should be able to (as in Figure 4.2): - Create a new IPA - Update an IPA - Delete an IPA - Create a new Dataset - Update a Dataset - Delete a Dataset - Execute IPA(s) - Logout

		- The user should be able to log in immediately after registration using (Username and Password) with no need for any additional activation\approval steps. o A valid Username and\or Password should only be accepted to log in. o Username and\or Password cannot be empty. o SQL injections should be handled.
002	**Create a new IPA**	The user should able to create and upload a new IPA, as in Figure 4.3, using the below fields. - After filling the IPA's information, the user is directed to upload the IPA's files. Multiple files should be allowed to be uploaded with no limitation on the number of the files. - The user has to upload all the files using one selection. - The IPA's main file should be uploaded within the uploaded file with the correct name that the user has entered in the Main File Name field. ▪ Name o IPA name. o A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters is accepted. o This field should be unique across all the IPA in the system.

		▪ Description o A brief description about the IPA, what it is about, the used algorithm, a comment regarding this specific uploaded IPA (for example what is new in this IPA version). o It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. ▪ Type o A mandatory drop-down list. The user can select the IPA type out of some specific options in a drop-down list. For now, the list contains image matching or image segmentation. o This list should enable adding and removing IPAs types in future work. ▪ Main File Name o The name of the main file that the execution should start with. There is no need to enter the extension of the file since the user will provide the used programming language name and the version number in other fields. o A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.

		○ The user is responsible for filling the correct name of the main file. If it is not the correct name, all the executions of that IPA will fail. ▪ Programming Language Name ○ The programming language that the user has developed the new IPA with. ○ A mandatory drop-down list. A list of the supported programming languages should be displayed to the user. For now, the list contains Java, Matlab, C/C++. ○ AEIPA should support IPAs written in different programming languages. ○ The drop-down list items should accept a maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters. ○ This list should enable adding and removing programming languages in future work. ▪ Programming Language Version Number ○ The version of the programming Language that the user has developed the new IPA with. ○ A mandatory drop-down list. A list of the supported versions of the selected programming language should be displayed to the user. For example, if the user has selected

		"Java" in the programming language, then the programming language version list should be filled with the supported versions of Java in AEIPA, the list contains Java SE 9, Java SE 8, and Java SE7. o The drop-down list items should accept a maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters. o This list should enable adding and removing programming languages versions in future work.
003	**Update an IPA**	- User should be able to modify only own IPAs - User should be able to update the information of the IPA without altering the IPA's code - If the user needs to make changes to the IPA's code, then the user has to create a new IPA - The user should not be able to modify IPA's name nor the uploaded files' content. - The user should be able to modify only these IPA's information: ▪ Description ▪ Type ▪ Main File Name ▪ Programming Language Name ▪ Programming Language Version Number

004	**Delete an IPA**	- User should be able to delete own created IPAs
005	**Create a new Dataset**	- Dataset is a collection of frames that represents a specific scene. Dataset can vary from small objects to large ones. - Logged in user can create a new dataset using the below fields then uploads the file(s) of the dataset as in Figure 4.5. ▪ Dataset Name o Dataset name. o A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. o This field should be unique across all the datasets in the system. ▪ Dataset Sequence Name o Dataset sequence name. o Every dataset can contain one or more dataset sequences o A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. o This field should be unique across all the sequences within the same dataset. ▪ Dataset Description

		○ A brief description about the dataset to be uploaded, what it is about, what does the frames contain mainly, what are the main uses of this dataset, ... etc. ○ It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. ▪ Dataset Evaluation Description ○ A brief description of how this dataset sequence is being evaluated. ○ It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. ▪ Keywords ○ The most important words that would describe the dataset. ○ This field is used mainly for search purposes in the existing datasets. ○ It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. ▪ Result Type ○ Dataset evaluation result type.

		○ A mandatory drop-down list. A list of the supported types of the dataset should be displayed to the user. Supported evaluation types are: Bounding Box and Labelled Bounding Box. ○ If the Labelled Bounding Box is selected, another field called "XY Label Sheet name" should be enabled so that the user must fill in the sheet file name. ○ The drop-down list items should accept a maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters. ○ This list should enable adding\removing result types in future work. ▪ XY Label Sheet Name ○ An optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted. ○ The user is responsible for filling in the correct values, so the execution and evaluation of this dataset does not fail. ○ For each dataset sequence, there is an XY Label sheet. ▪ After filling all this information, the user gets directed to upload all the files in the dataset sequence. Multiple files upload is allowed. In case the dataset type is labeled bounding box, then the user has to

		fill in the correct value and upload the file with the dataset frames in each dataset sequence. ▪ Finally, the user is directed to upload the ground truth dataset. Ground truth dataset is the ideal result of that dataset. Every dataset sequence has its ground-truth dataset. The user is responsible for matching the names of the files in the ground-truth dataset with the dataset sequence.
006	**Update a Dataset**	- The user should be able to modify only their own datasets. - The user should not be able to modify dataset's name, dataset sequences names, nor the uploaded files' content. - The user should be able to modify only these dataset's information: ▪ Dataset Description ▪ Dataset Evaluation Description ▪ Result Type ▪ Keywords
007	**Delete a Dataset**	- User should be able to delete own datasets
008	**Search for IPAs**	- User can search for any IPA(s) in AEIPA, select and execute it (without modifying anything). User can search by IPA name, type, or even using a keyword in the description. When a user enters search values, all the IPA(s) with these values should be displayed to the user, even the IPA(s) that the user did not create. Any user

		should be able to search and display the IPA(s) created by other users. - The user can select among the resulted IPA(s) and proceed in executing them. - Search results is a table that contains: ▪ Name ▪ Description ▪ Type ▪ Programming Language Name ▪ Programming Language Version Number ▪ User Name
009	**Search for Datasets**	- The user can search by certain keywords that would describe a dataset and dataset result types. Like the IPA Search use case, the user can display any dataset, even the one(s) that was created by other users.
010	**Execute IPA(s)**	- User should be able to select among IPAs and proceed in executing them. - The user can execute any IPA(s) with any dataset(s) that exist in AEIPA. First, user search among all the IPAs and all the datasets. Then, the user selects the needed IPA(s) and dataset(s) to get executed. Each selected IPA is executed against each selected

		dataset. While executing an IPA progress information is displayed to the user. - During the execution, the evaluation algorithm(s) should be executed to evaluate the IPA being executed and compare it against the associated ground-truth dataset. An evaluation algorithm can be as simple as calculating the confusion matrix (calculating the TNR: True Negative Rate, FPR: False Positive Rate, FNR: False Negative Rate, TPR: True Positive Rate). - Finally, an execution report is displayed at the end of the execution. The execution report contains a table with the confusion matrix for each of the datasets executed using an IPA as depicted below in table 4.1.

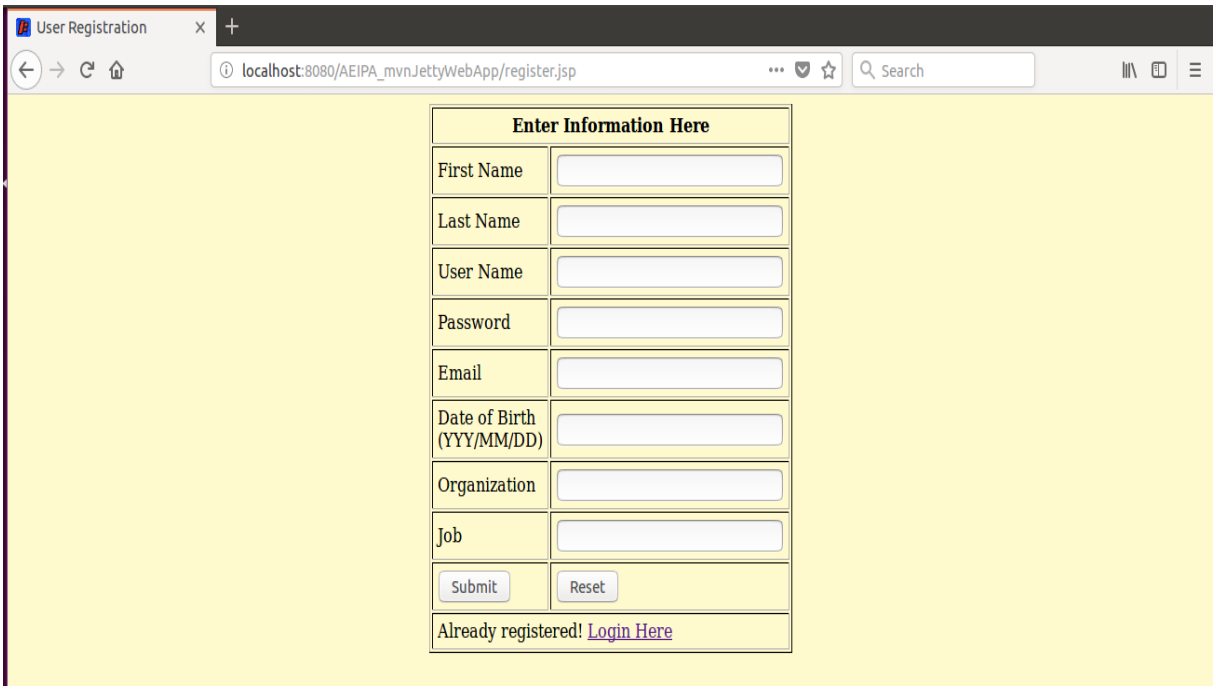


Figure 4.1: AEIPA User Registration

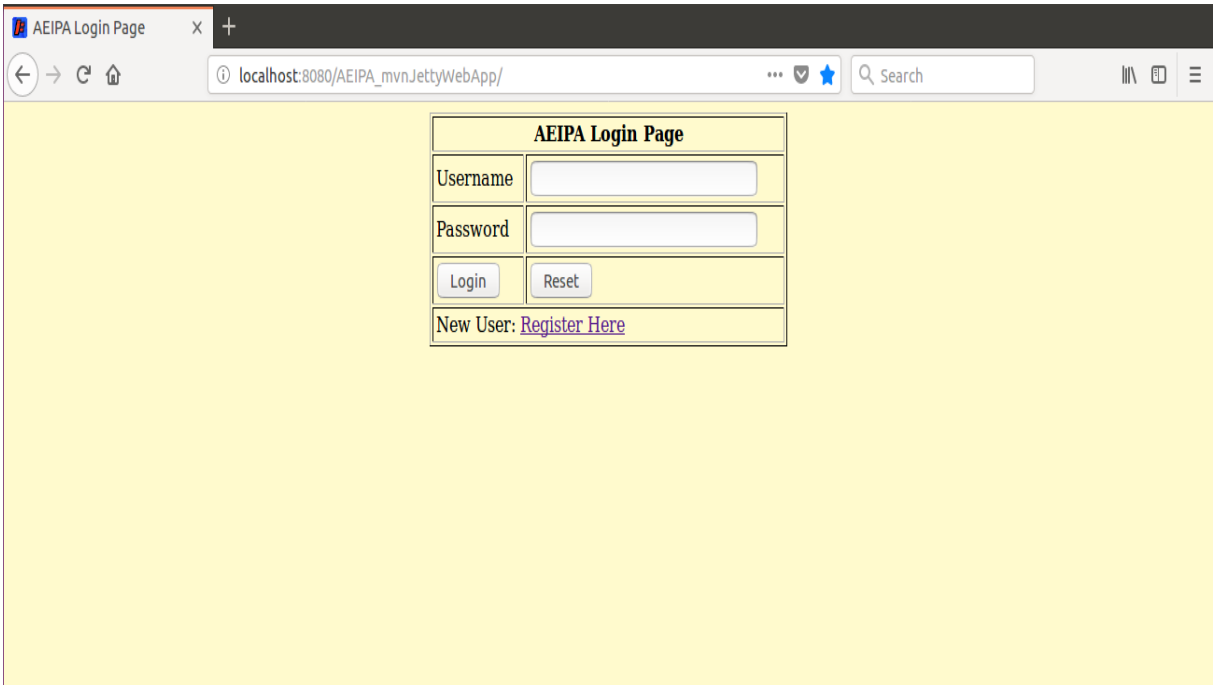


Figure 4.2: AEIPA Login Page

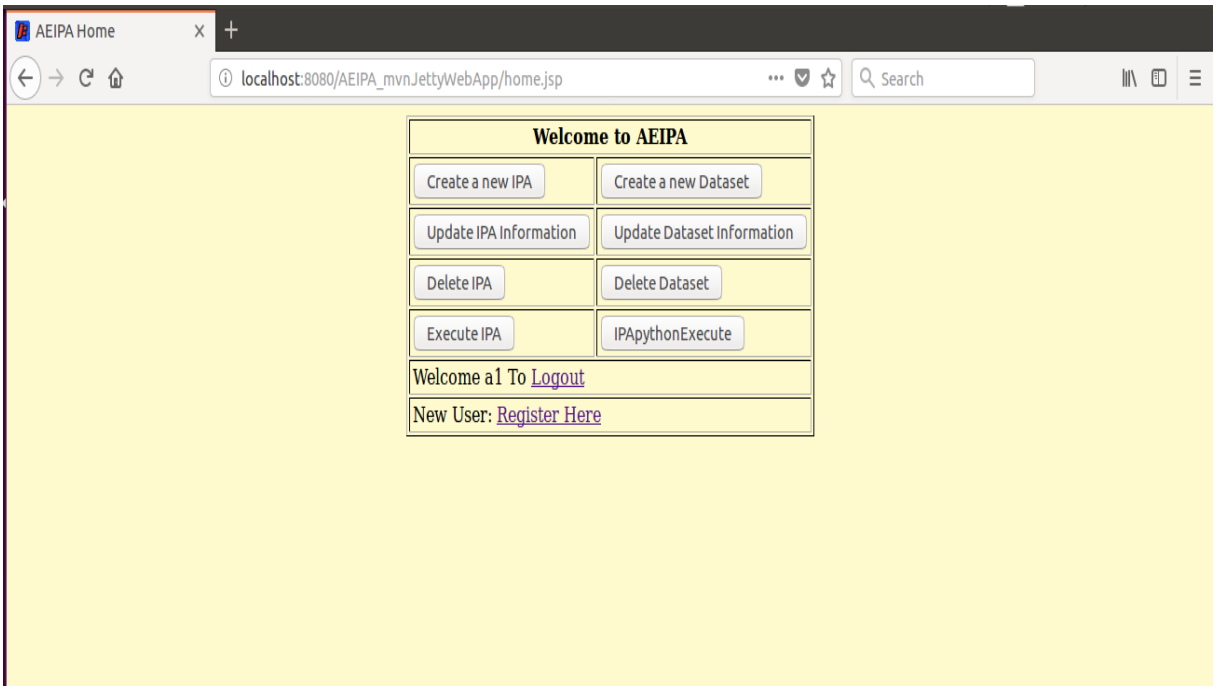


Figure 4.3: AEIPA Actions

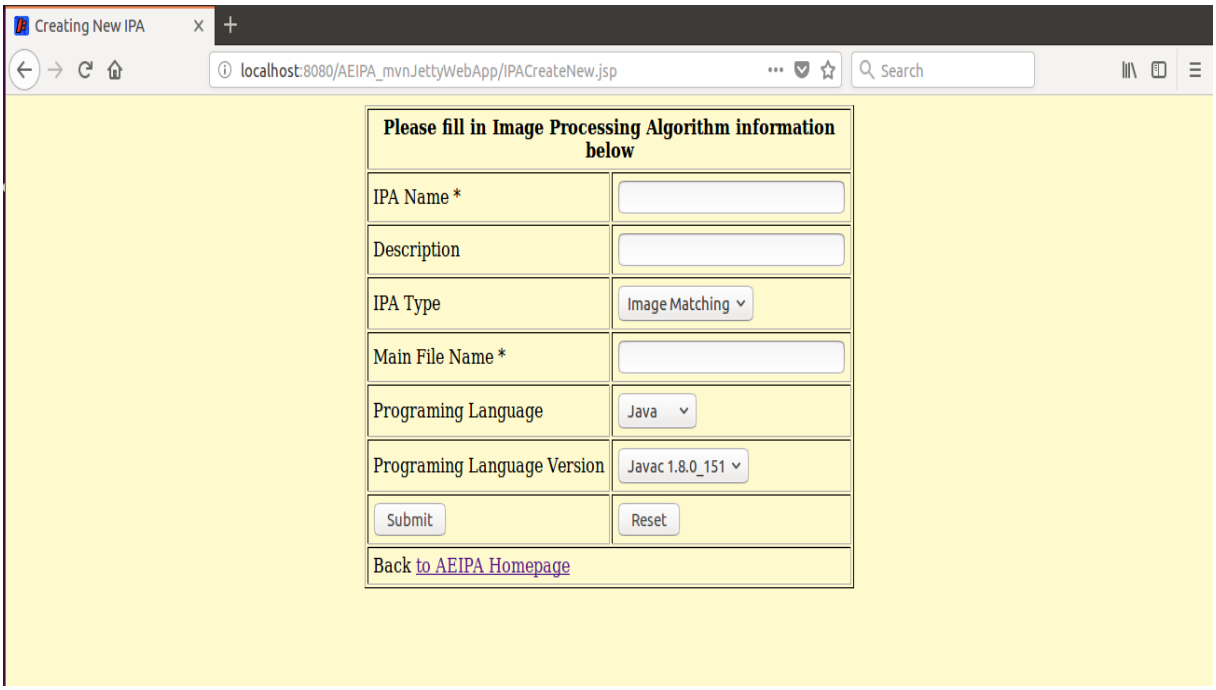


Figure 4.4: AEIPA Create a new IPA

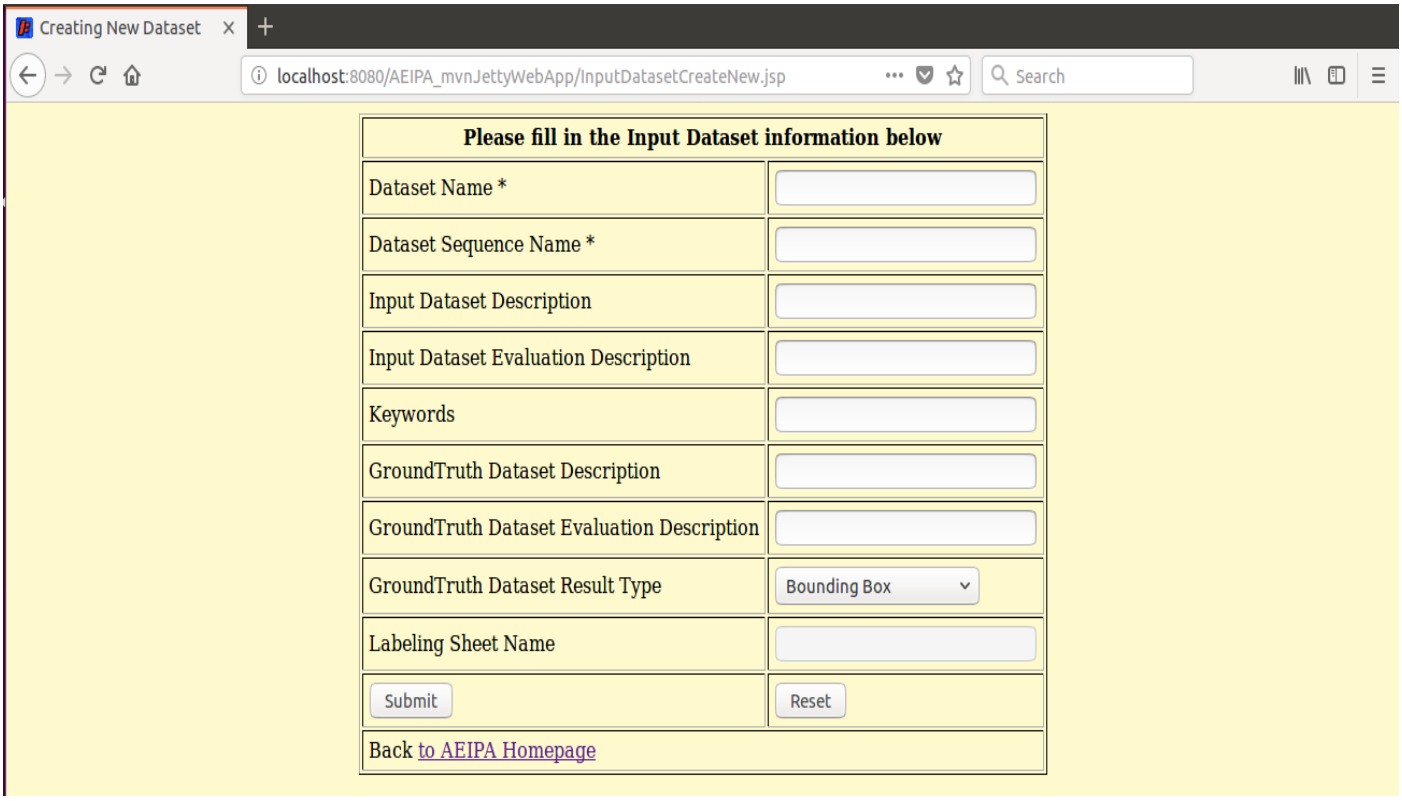


Figure 4.5: AEIPA Create a new Dataset Sequence

Table 4.1: Execution Results Example

Dataset Name	Dataset Sequences	IPA 1				IPA 2			
Dataset 1	Sequence 1	TNR	FPR	FNR	TPR	TNR	FPR	FNR	TPR
	Sequence 2								
	Sequence 3								
Dataset 2	Sequence 1								
Dataset 3	Sequence 1								
	Sequence 2								

4.5 AEIPA System Design

4.5.1 Spotify Docker Client

This is a Docker client written in Java. It is used in many critical production systems at Spotify. Spotify Docker-client is tested using the six most recent releases of Docker, as mentioned in [91]. Docker-client can be built on any platform with JDK8+, Docker 1.6+, and a recent version of Maven 3. Spotify Docker client is added to the maven project using the below dependency:

```
<dependency>
  <groupId>com.spotify</groupId>
  <artifactId>docker-client</artifactId>
  <version>LATEST-VERSION</version>
</dependency>
```

Spotify Docker client is used in AEIPA in order to avail the functionality of creating Docker images and building a Docker container for each execution of a software algorithm with one input. Below code simplifies how Spotify Docker Client is used in AEIPA [91]:

```
// Create a client based on DOCKER_HOST and DOCKER_CERT_PATH env
vars
final DockerClient docker =
DefaultDockerClient.fromEnv().build();
// Pull an image
docker.pull("busybox");
// Bind container ports to host ports
final String[] ports = {"80", "22"};
```

```
final Map<String, List<PortBinding>> portBindings = new
HashMap<>();
for (String port : ports) {
    List<PortBinding> hostPorts = new ArrayList<>();
    hostPorts.add(PortBinding.of("0.0.0.0", port));
    portBindings.put(port, hostPorts);
}
// Bind container port 443 to an automatically allocated
available host port.
List<PortBinding> randomPort = new ArrayList<>();
randomPort.add(PortBinding.randomPort("0.0.0.0"));
portBindings.put("443", randomPort);
final HostConfig hostConfig =
HostConfig.builder().portBindings(portBindings).build();
// Create container with exposed ports
final ContainerConfig containerConfig =
ContainerConfig.builder()
    .hostConfig(hostConfig)
    .image("busybox").exposedPorts(ports)
    .cmd("sh", "-c", "while :; do sleep 1; done")
    .build();
final ContainerCreation creation =
docker.createContainer(containerConfig);
final String id = creation.id();
```

```
// Inspect container
final ContainerInfo info = docker.inspectContainer(id);
// Start container
docker.startContainer(id);
// Exec command inside running container with attached STDOUT
and STDERR
final String[] command = {"sh", "-c", "ls"};
final ExecCreation execCreation = docker.execCreate(
    id, command, DockerClient.ExecCreateParam.attachStdout(),
    DockerClient.ExecCreateParam.attachStderr());
final LogStream output = docker.execStart(execCreation.id());
final String execOutput = output.readFully();
// Kill container
docker.killContainer(id);
// Remove container
docker.removeContainer(id);
// Close the docker client
docker.close();
```

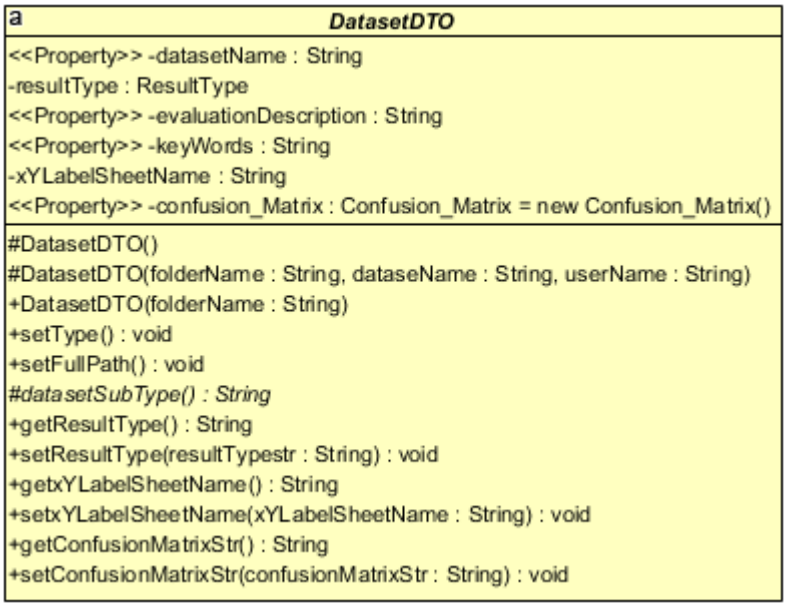


Figure 4.6: AEIPA Dataset

DTOs in AEIPA are datasets with all the attributes shown in figure 4.6, where each dataset has a ground truth and an evaluation type. More details about AEIPA DTOs hierarchical design is in figure 4.7 for both datasets and IPAs. Each DatasetContainerDTO contains the input dataset and the ground truth and saves the result datasets per each execution, see figure 4.8.

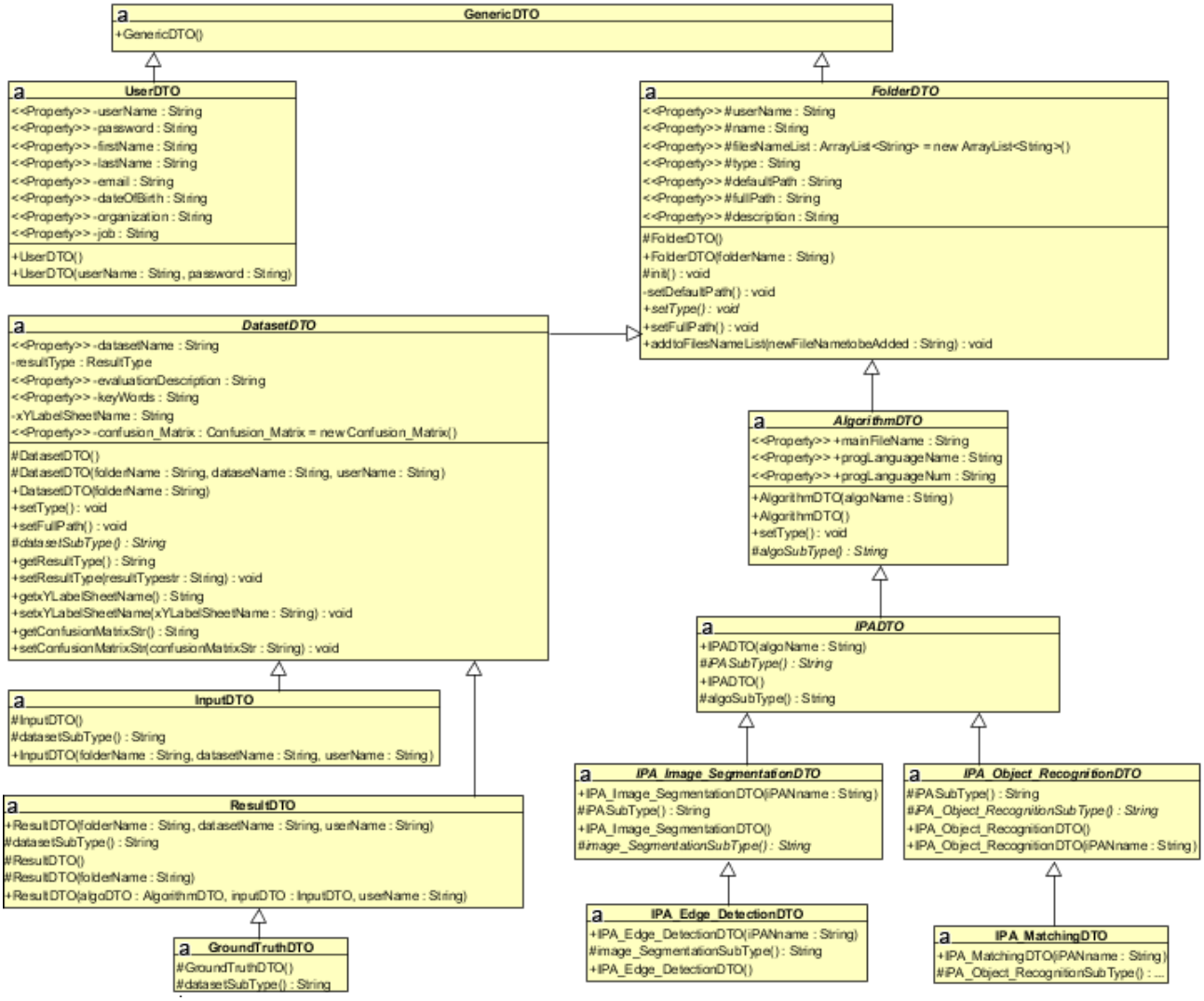


Figure 4.7: AEIPA DTOs Hierarchy

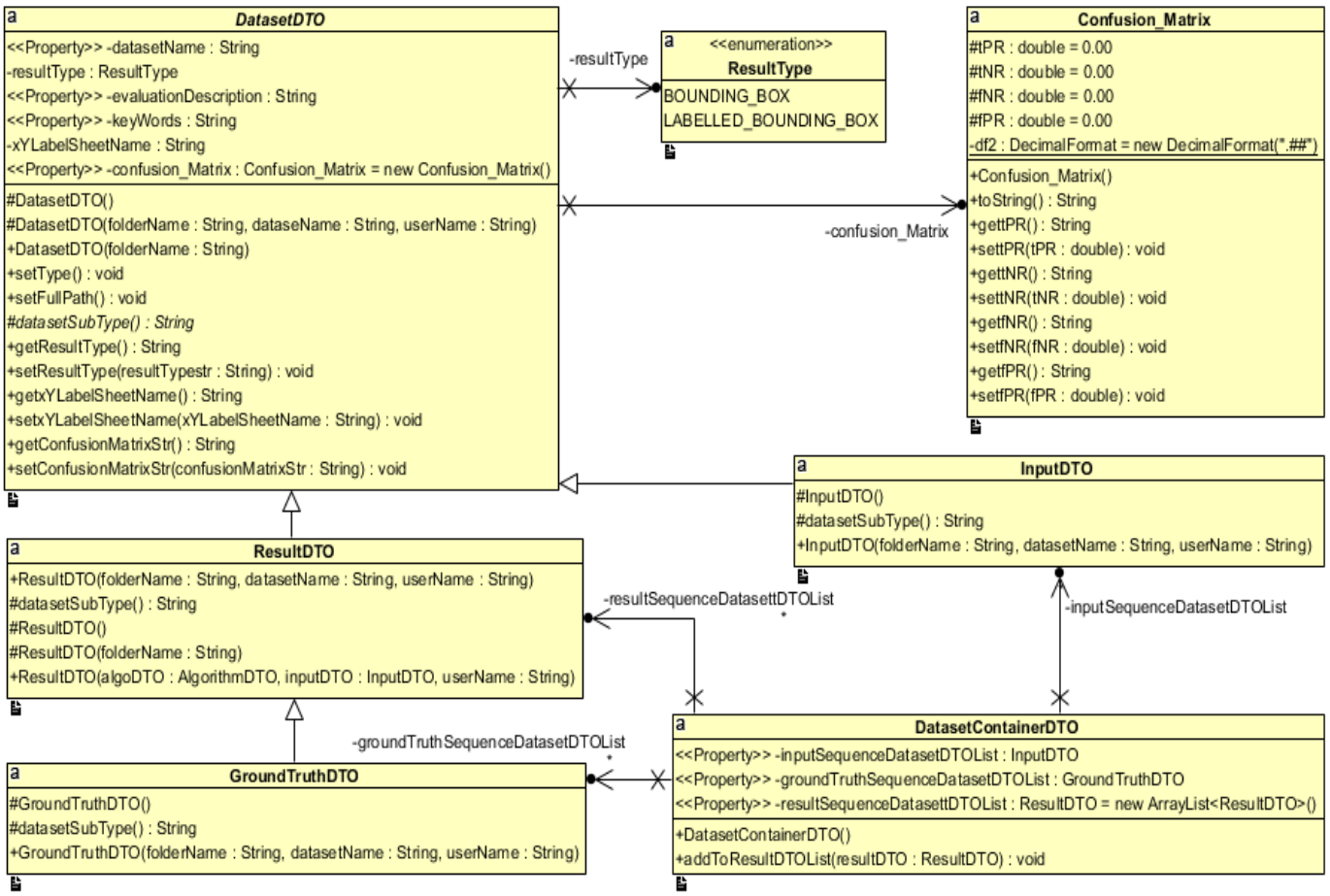


Figure 4.8: AEIPA Input Supplier Dataset

Figure 4.9 shows the hierarchy of AEIPA DAOs, while the relationship between DTOs and DAOs is in figure 4.10. DAOs process DTOs generically by using templates in Java and do not include any details about the technicality of the used database. Database's connectivity and queries are written in the DBConnection class, as explained in the previous chapter in figure 3.8.

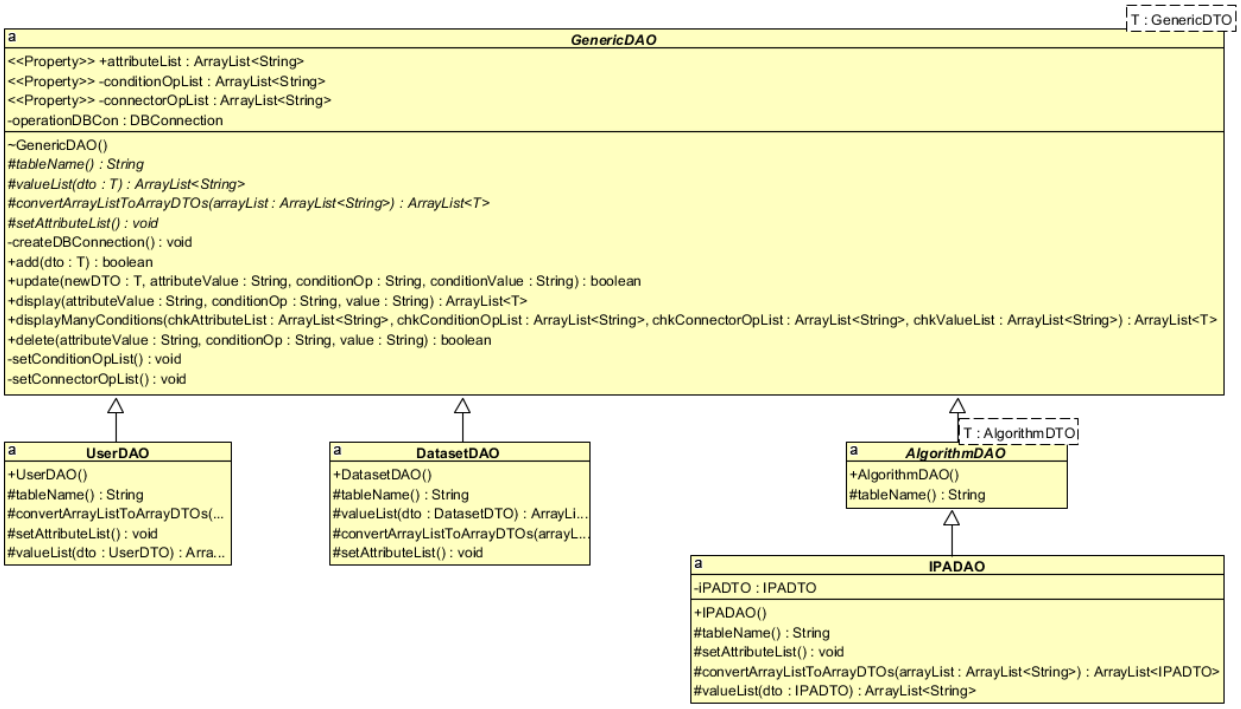


Figure 4.9: AEIPA DAOs Hierarchy

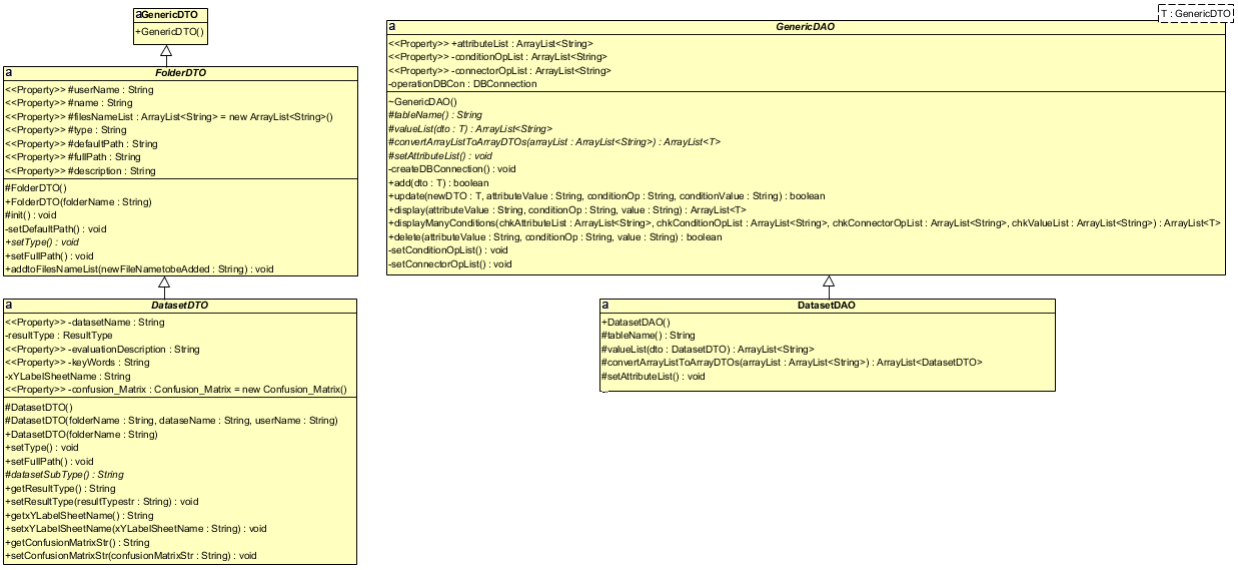


Figure 4.10: AEIPA DTOs - DAOs relationship

4.6 AEIPA Execution and Testing

User starts with registering to AEIPA and can do any of the following (as was in figure 4.2):

- Create a new IPA
- Update an IPA
- Delete an IPA
- Create a new Dataset
- Update a Dataset
- Delete a Dataset
- Execute IPA(s)
- Logout

Though, in this section only executing an already uploaded IPA is addressed. User can search AEIPA Data Repository using the IPA's category and the dataset evaluation type as in figures 4.11 and 4.12. All the matching IPAs and datasets are listed in the searching results page, and the user can select one or many IPAs in addition to the datasets as in figure 4.13. Both figures 4.14 and figure 4.15 shows the execution progress based on the number of IPAs and datasets. Evaluation measurements are calculated during the execution of the IPAs, and the confusion matrix is displayed after all the executions are done, as in figure 4.16. A confusion matrix is a table that describes the performance of the IPA according to the four values: true-positive rate, false-positive rate, true-negative rate, and false-negative rate.

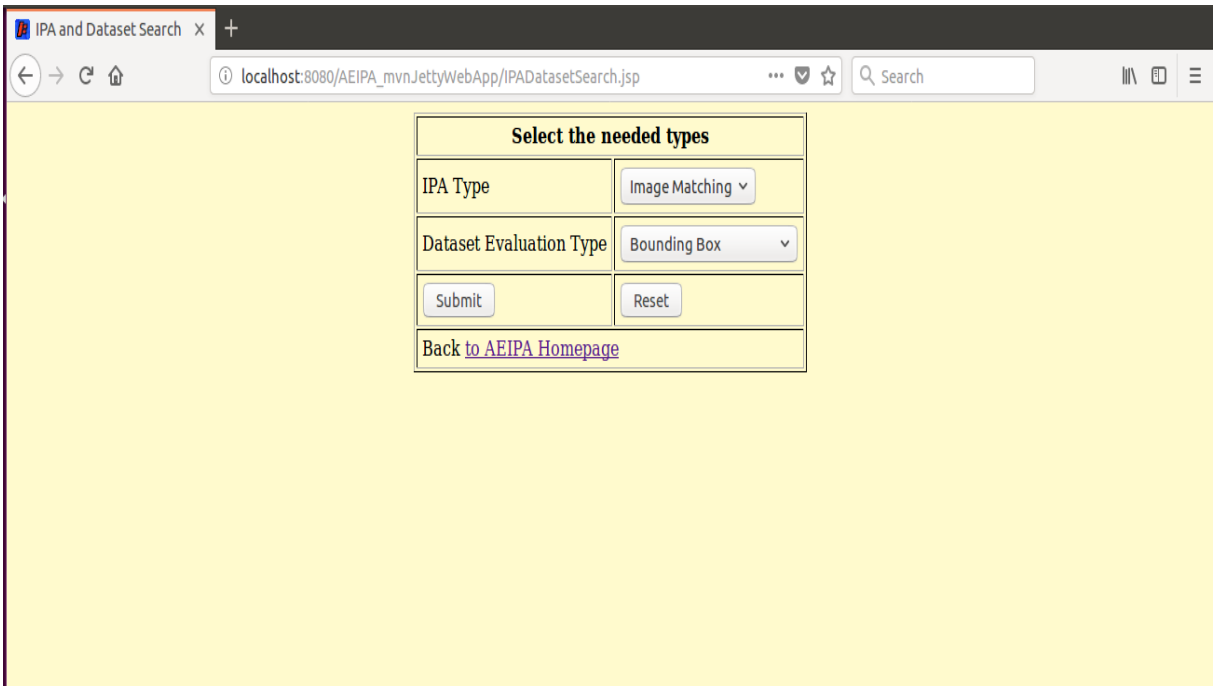


Figure 4.11: Searching for IPAs generating bounding boxes

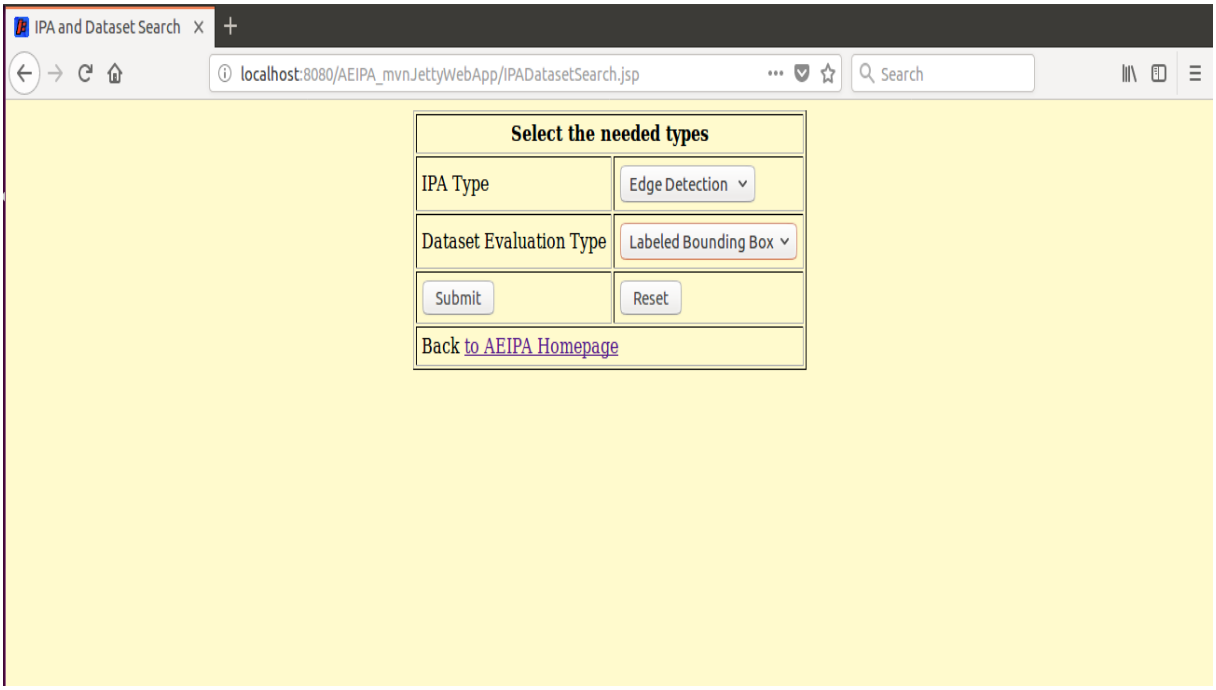


Figure 4.12: Searching for IPAs generating bounding boxes

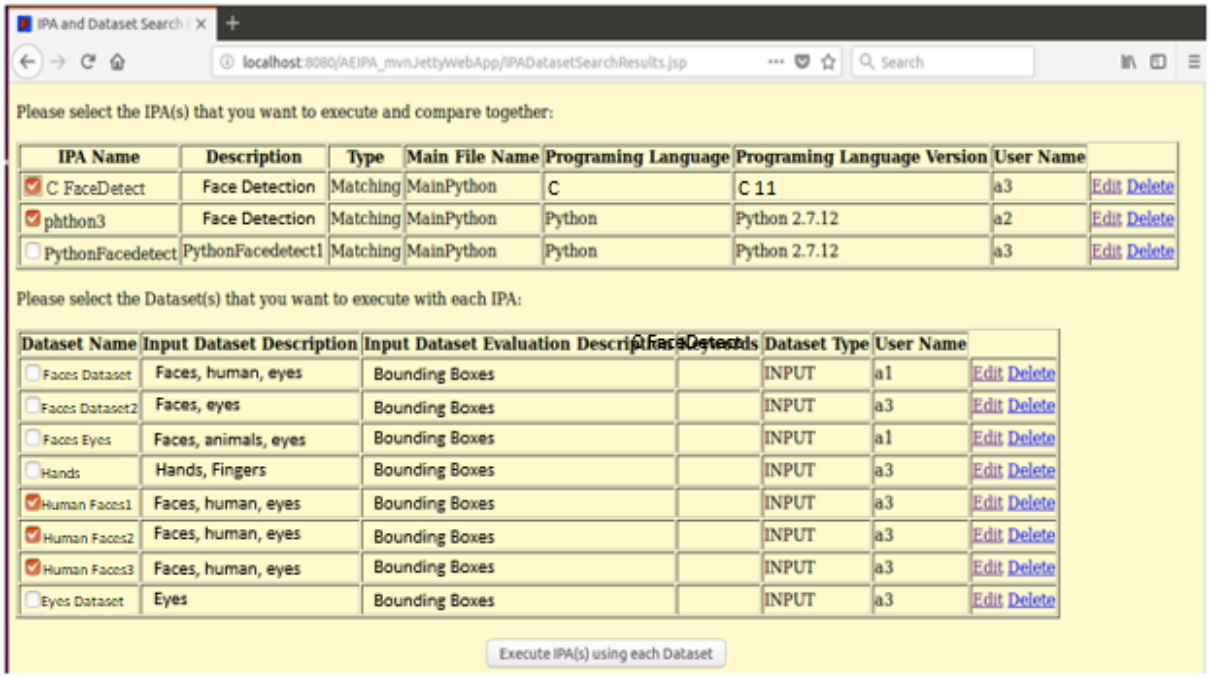


Figure 4.13: AEIPA Searching results

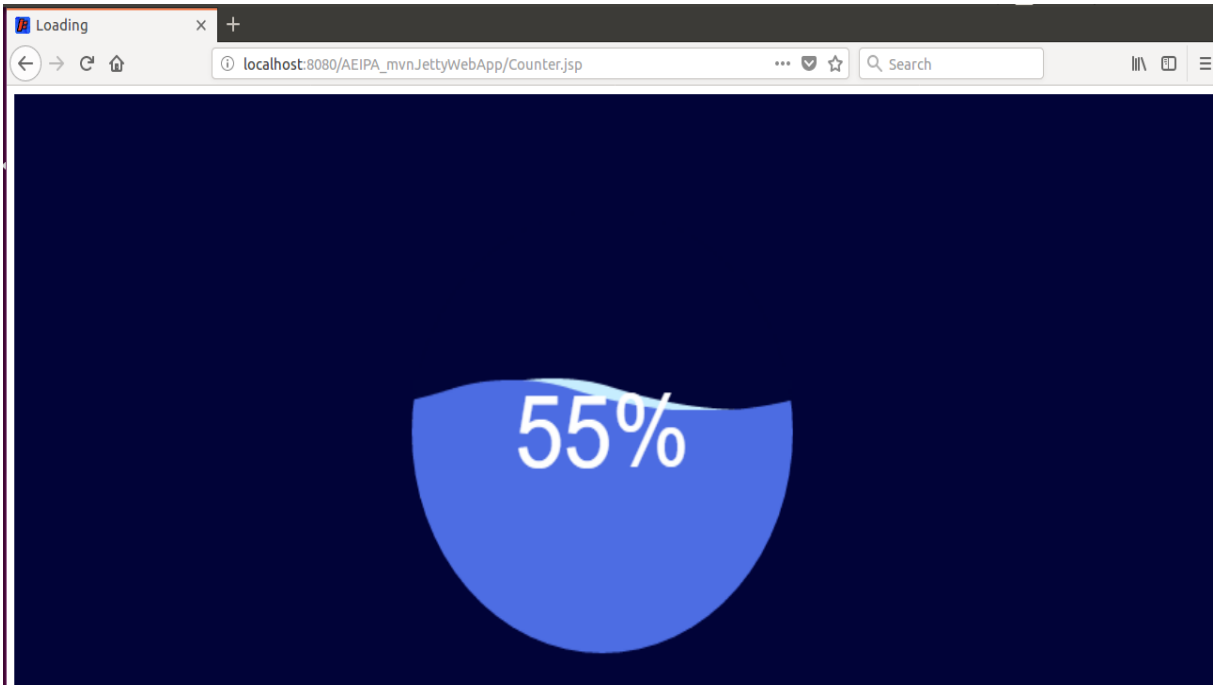


Figure 4.14: AEIPA execution counter 55%

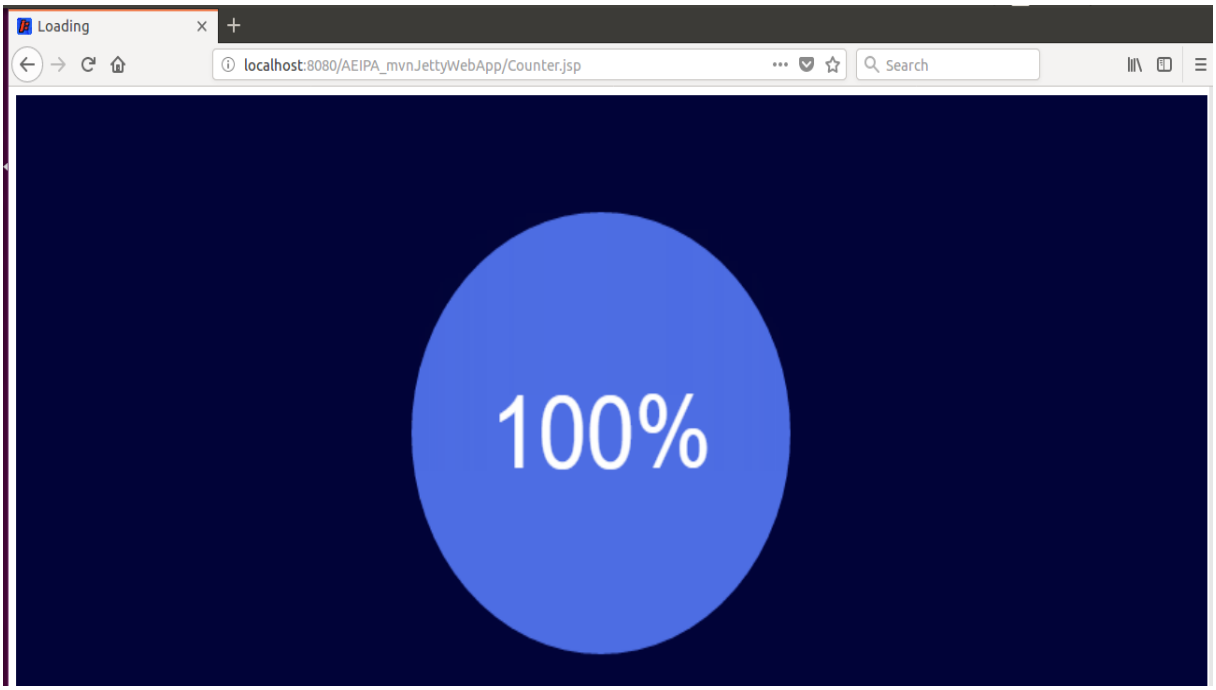
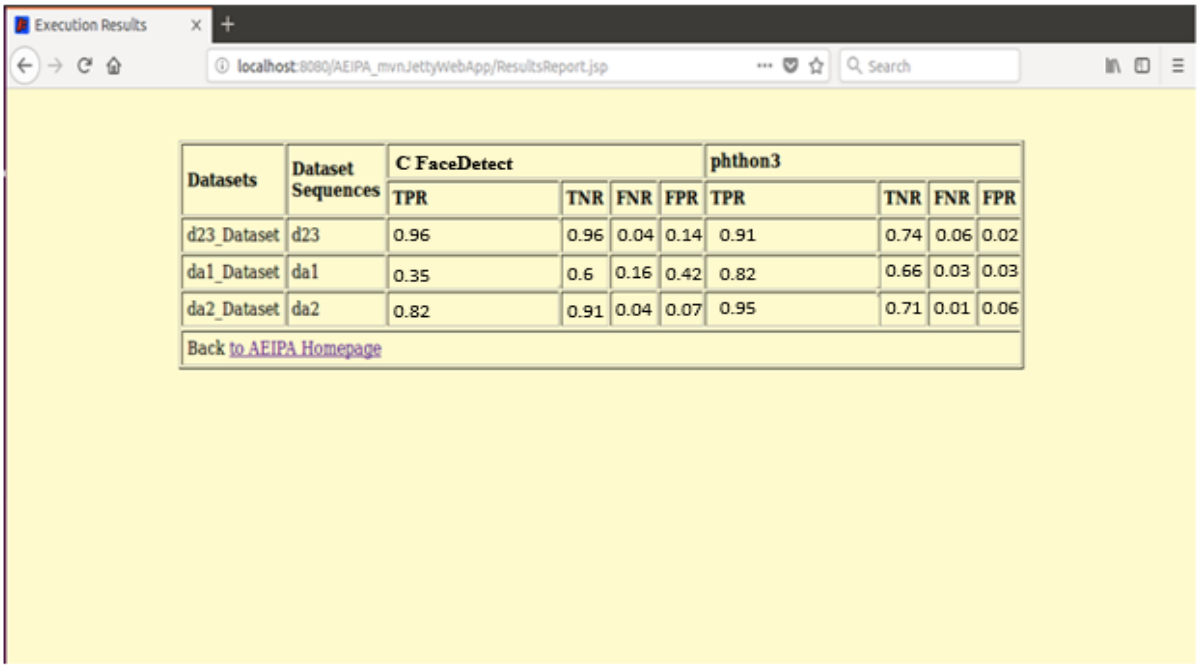


Figure 4.15: AEIPA execution counter 100%



Datasets	Dataset Sequences	C FaceDetect				phthon3			
		TPR	TNR	FNR	FPR	TPR	TNR	FNR	FPR
d23_Dataset	d23	0.96	0.96	0.04	0.14	0.91	0.74	0.06	0.02
da1_Dataset	da1	0.35	0.6	0.16	0.42	0.82	0.66	0.03	0.03
da2_Dataset	da2	0.82	0.91	0.04	0.07	0.95	0.71	0.01	0.06
Back to AEIPA Homepage									

Figure 4.16: AEIPA Confusion Matrix results

AEIPA has been tested on many IPAs, but only two IPAs are presented in this thesis. Appendix A shows Haar Cascade Face Detection IPA using C and OpenCV and with a sample of the XML file for determining the frontal face specifications (Appendix B). A resulting image of executing the algorithm using AEIPA is shown in Appendix C. Another implementation for the Haar Cascade Face Detection IPA using Python is shown in Appendix D, and the used XML training file is in Appendix E. A sample result of executing the second IPA using AEIPA is shown in Appendix F. The execution time of the first algorithm written using C and OpenCV (3.51 seconds per image) is much faster than Python and OpenCV (4.423 seconds per image). The difference in the end results between the two algorithms is due to using a built-in library in the first algorithm that uses C and OpenCV.

Chapter 5

Conclusion and Future Work

This final chapter summarizes the proposed framework, ADESA, and implementation, AEIPA, discusses the results, highlights the research contributions, lists system limitations, and presents the potential future research directions in the last section.

5.1. Summary

The goals of this research are to address the fundamental issues related to automating the deployment, execution, and evaluation of software applications. Software developers and researchers used to go through hectic steps in order to have their developed software algorithms executed, search for datasets to test different challenges, search for all the related benchmark software algorithms, implement and execute all these benchmark algorithms in order to compared to their own algorithm, evaluate the execution of the algorithms against each dataset, and report the evaluation results in a readable format. The increasing complexity of software systems leads to the necessity of using automated approaches to develop and evolve economically.

To fulfill the goals mentioned above, a web-based framework for automating the execution and evaluation of software algorithms has been implemented, ADESA. This framework allows users to extend it for implementing a system serving a specific algorithm's category, e.g., AEIPA system for IPAs. AEIPA is a web application for everyone to share IPAs, datasets, and ground truth, IPAs' executions, and evaluation results.

5.2. System Discussion

This work has aimed to address the fundamental issues related to automating the execution of software algorithms and automating the evaluation of the executed algorithms. To accomplish these goals, a Docker-based software framework for automating the execution and evaluation of software algorithms has been introduced, ADESA. This web-application framework allows users, such as software researchers, system analysts, and developers, to gather all the matching software algorithms, test data, and evaluation algorithms in one system.

Each Docker image generates a Docker container for each expected working environment that can execute the software algorithm. Each software algorithm executes only a single input inside a Docker container, while, threading allows executing all the created Docker containers simultaneously. Evaluation algorithms are performed during and after executing all the software algorithms. Evaluation matrices, e.g., confusion matrix in the image processing field, as explained in the previous chapter, are collected and displayed in reports. Docker containerization gained importance as it offers flexibility, less overhead, reusability, and portability of applications [61]. Also, the Docker containers have remarkable superior performance compared to virtualization using a full-scale system, as discussed in [21].

ADESA framework adopts a Docker engine for executing the software algorithms inside Docker containers. The Docker container constructs a working environment ready for executing a software algorithm using a single input. Users are kept away from any system installation and environment configuration problems. Also, using Docker containers provided agility and mobility to the framework that Docker containers can be executed into distributed servers. The Input Supplier module stores algorithms' source code, input data, and evaluation criteria in the Data

Repository. The Execution submodule creates a Docker container with all the needed dependencies to execute an algorithm with one input. Threading has been used in this phase in order to execute all the created Docker containers simultaneously. The Execution results are stored back to the Data Repository for users and evaluation. The executed algorithms execute only once using input data. Meaning that once an algorithm is executed using an input, the results are stored, and that algorithm does not get executed with this input again by different users.

Additionally, ADESA collects evaluation measurements during and after the software algorithm's execution and converts the algorithm's results into meaningful comparison report. As described in AEIPA system, the output of the Execution phase is visual results in the form of either bounding boxes or labeled bounding boxes. Consequently, quantitative results are required for an accurate comparison of the IPAs. The Evaluation phase allows the visual results of IPAs to be examined according to a set of standard evaluation matrices. The Evaluation phase automatically calculates the evaluation matrix: TPR, TNR, FNR, and FPR, for the visual results of IPAs. The reporting phase is concerned with representing the output results in a readable format such as the comparison tables, statistical plots, resources consumption, performance graphs, and pie charts according to the user's choices. ADESA framework generally evaluates each software algorithm being executed and compare the results to other software algorithms. The framework is open to adding new evaluation methods and new reports.

5.3. Research Contributions

The main contributions of this research are:

- Automating of software algorithms' execution and evaluation and comparing the evaluation results.

- User has the option to pick the working environment they would like to execute and have it up and running within seconds.
- Using Docker containers provided mobility and security to the framework that software algorithm execution is self-contained and can be executed into distributed servers.
- ADESA framework has scalability with only configuring the shared files.

Software algorithms are developed using different working environments (different programming languages, dependencies, libraries, and OS). Users such as software developers, researchers, and analysts spend much effort searching for software algorithms under a specific category, setting up working environments for executing the found algorithms, searching for the dataset inputs for evaluating the algorithms from different perspectives, which was an unexplored area of study. Thus, the fundamental research question here was: Is it possible to design a common system that would be able to gather, execute, and compare all the different algorithms? This work presents a software design of ADESA framework , and it outlines the aspects of automating the execution and evaluation of software algorithms. ADESA framework has contributed to unifying the way to deal with executing and evaluating different software algorithms. ADESA is a web-application framework that gathers all the software algorithms and the input datasets in one common web application. Software algorithms comparison become shared publicly to all users. Moreover, ADESA builds a working environment and executes each software algorithm with one, and only one, input at a time. The working environment is built inside a Docker container using a prepared Docker Image.

5.4. System Limitations

Although the implementation and testing of the AEIPA system showed that ADESA automation framework is useful for automating the execution and evaluation of software algorithms, this work has some limitations. Software evaluation is a wide concept representing the quality of software. All the software resources affect the performance of the software, besides the software itself, such as CPU, video card, hard drive, memory, middleware, operating system, and communication networks. [90]. The resources evaluation of the software surroundings had not been implemented into ADESA. Thus, this may limit software algorithms' evaluation that can be discovered by using this framework. However, ADESA has been implemented using the open-closed principle that new evaluation criteria can be injected into the framework. New performance measurements can be implemented and added to ADESA.

An efficient way to evaluate the software algorithm is to treat every input's execution individually [91]. Alternatively, when the software algorithm is executing a set of inputs, evaluation measurements should be applied to a single execution of the software algorithm using one input. Despite that this is the case in ADESA, there are some drawbacks when some algorithms require interdependencies between inputs (like exchanging some data between separate executions). The evaluation would not be realistic in this case when executing each input separately. A set of the inputs (or all of the inputs depending on the algorithm's nature) need to be executed together in a Docker container. ADESA framework is scalable. A different execution mechanism can be added to ADESA for the more complicated software algorithms that require special execution of inputs.

Another limitation in ADESA is that evaluation algorithms are added in the code of the framework. New evaluation algorithms have to be coded into the framework and the systems that implement the framework such as AEIPA system. Though, ADESA framework can obtain, store, and execute evaluation algorithms in the same way the software algorithms are obtained, stored, and executed. Evaluation algorithms can be uploaded as source code and executed using Docker containers. This will be easier for all ADESA users to create, modify, or delete evaluation algorithms.

5.5. Generalizability

Although the prototype system, AEIPA, was implemented by analyzing the relationship between image processing algorithms and datasets, ADESA was designed as a reusable abstraction software environment providing generic automatic execution of software algorithms with inputs and evaluating outputs. ADESA framework provides a standard way to build systems for a specific software algorithm category. ADESA code accepts the user's implementation extensions. The framework aims to facilitate the development of common web applications for software's deployment and evaluation. Using the ADESA framework, future development of software's automation web applications like AEIPA are effective, faster, and easier to complete. New software can extend ADESA by overriding or adding code for a specific software algorithm category.

However, executing and evaluating software algorithms get more complicated with algorithms such as the databases, distributed systems, networking, and operating systems algorithms. The different nature of the inputs and outputs cannot be implemented in ADESA as it is. Although, such complicated algorithms can be executed and evaluated using ADESA

framework by simulating inputs and outputs using data files. ADESA framework can process any software algorithms as long as inputs are presented using data files and outputs are presented using data files as well.

Also, ADESA evaluates each software algorithm being executed and compare the results to other software algorithms. The framework is open for adding new evaluation methods and new reports, as explained in the previous section due to MVC. Using MVC design pattern promoted organizing the code by the purpose of each part of the code, which made it easy to extend and reuse the code. That is why in ADESA any software algorithm, either input algorithm or an evaluation algorithm, can be executed in Docker containers that are built using prepared Docker images.

5.6. Future Work

Addressing the limitations of this work is a good start for future work for the system. Using a distributed system is going to show how powerful the framework is. Spreading the execution of the Docker containers over a distributed system is very encouraged for ADESA framework. Each execution of a software algorithm using a single input is portable and ready to be executed on independent machines. Using a distributed system will add more scalability that ADESA can be expanded as needed by adding more machines. Also, the distributed system will add high availability for ADESA as several machines provide the same functionality of executing Docker containers, so if one becomes unavailable, other machines can substitute. Moreover, replacing the physical server with cloud servers will add a great value to the system. Using cloud servers will provide an on-demand allocation of resources and data storages and will increase the availability of the system. Another great extension is that users can upload their own docker files.

Users will be able to customize their preferred working environment by their own and upload it to ADESA system to be used publicly or privately depending on their preferences.

Chapter 6

References

[1] E. Kit, Software Testing in the Real World: Improving the Process, Addison-Wesley, Reading, Mass, USA, 1995.

[2] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," RTI Project 7007.011, U.S. National Institute of Standards and Technology, Gaithersburg, Md, USA, 2002.

[3] R. Ramler and K. Wolfmaier, "Observations and lessons learned from automated testing," in Proceedings of the International Workshop on Automation of Software Testing (AST '06), pp. 85–91, Shanghai, China, May 2006.

[4] K. Karhu, T. Repo, O. Taipale, and K. Smolander, "Empirical observations on software testing automation," in Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation (ICST '09), pp. 201–209, Denver, Colo, USA, April 2009.

[5] O. Taipale and K. Smolander, "Improving software testing by observing causes, effects, and associations from practice," in Proceedings of the International Symposium on Empirical Software Engineering (ISESE '06), Rio de Janeiro, Brazil, September 2006.

[6] B. Shea, "Software testing gets new respect," InformationWeek, July 2000.

[7] E. Dustin, J. Rashka, and J. Paul, Automated Software Testing: Introduction, Management, and Performance, Addison-Wesley, Boston, Mass, USA, 1999.

[8] S. Berner, R. Weber, and R. K. Keller, “Observations and lessons learned from automated testing,” in Proceedings of the 27th International Conference on Software Engineering (ICSE ’05), pp. 571–579, St. Louis, Mo, USA, May 2005.

[9] J. A. Whittaker, “What is software testing? And why is it so hard?” IEEE Software, vol. 17, no. 1, pp. 70–79, 2000.L. J. Osterweil, “Software processes are software too, revisited: an invited talk on the most influential paper of ICSE 9,” in Proceedings of the 19th IEEE International Conference on Software Engineering, pp. 540–548, Boston, Mass, USA, May 1997.

[10] J. Kasurinen, O. Taipale, K. Smolander, “Software Test Automation in Practice: Empirical Observations,” Advances in Software Engineering, vol. 2010, Article 4 (January 2010), 13 pages.

[11] Ian Sommerville. 2004. Software Engineering (7th Edition). Pearson Addison Wesley.

[12] D. Powers, Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation, Journal of Machine Learning Technologies, vol. 2, no. 1, pp. 3763, 2011.

[13] Steven S. Skiena. 2008. The Algorithm Design Manual (2nd ed.). Springer Publishing Company, Incorporated.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press.

[15] Karsten Weihe. 1997. Reuse of algorithms: still a challenge to object-oriented programming. In Proceedings of the 12th ACM SIGPLAN conference on Object-oriented

programming, systems, languages, and applications (OOPSLA '97), A. Michael Berman (Ed.). ACM, New York, NY, USA, 34-48. DOI=http://dx.doi.org.qe2a-proxy.mun.ca/10.1145/263698.263704

[16] Padhy, Neelamadhab & Singh, R.P. & Satapathy, Suresh. (2017). Software reusability metrics estimation: Algorithms, models and optimization techniques. Computers & Electrical Engineering. 69. 10.1016/j.compeleceng.2017.11.022.

[17] R. Dua, A. R. Raja, and D. Kakadia, ”Virtualization vs containerization to support paas,” in Cloud Engineering (IC2E), 2014 IEEE International Conference on, 2014, pp. 610-614.

[18] T. Bui, ”Analysis of docker security,” arXiv preprint arXiv:1501.02967, 2015.

[19] A. Azab, "Enabling Docker containers for High-Performance and Many-Task Computing," 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC, 2017, pp. 279-285.

[20] Pankaj Saha, Angel Beltre, Piotr Uminski, and Madhusudhan Govindaraju. 2018. Evaluation of Docker containers for Scientific Workloads in the Cloud. In Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18). ACM, New York, NY, USA, Article 11, 8 pages.

[21] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. Linux Journal 2014, 239 (2014), 2.

[22] A. Abdel Naby, M. S. Shehata and T. S. Norvell, "AEIPA: Docker-based system for Automated Evaluation of Image Processing Algorithms," 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), Windsor, ON, 2017, pp. 1-4.

[23] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113.

[24] J. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," in *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 639-648, Nov. 1983.

[25] K. Christensen, G. P. Fitsos, and C. P. Smith, "A perspective on software science," IBM Syst. J., vol. 20, no. 4, pp. 372-387, 1981.

[26] OSMAN, Hafeez; CHAUDRON, Michel R.V.. Correctness and Completeness of CASE Tools in Reverse Engineering Source Code into UML Model. GSTF Journal on Computing (JoC), [S.l.], v. 2, n. 1, Jan. 2018. ISSN 2010-2283.

[27] Robert C. Martin. 2000. The open-closed principle. In More C++ gems, Robert C. Martin (Ed.). Cambridge University Press, New York, NY, USA 97-112.

[28] A. Eberlein, G. Succi and J. W. Paulson, "An Empirical Study of Open-Source and Closed-Source Software Products," in IEEE Transactions on Software Engineering, vol. 30, no., pp. 246-256, 2004.

[29] R. S. Freedman, "Testability of software components," in *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 553-564, June 1991.

[30] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and Linux containers,” in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw., 2015, pp. 171–172.

[31] R. Dua, A. R. Raja, and D. Kakadia, “Virtualization versus containerization to support PaaS,” in Proc. IEEE Int. Conf. Cloud Eng., 2014, pp. 610–614.

[32] J. Bhimani *et al*., "Docker container Scheduler for I/O Intensive Applications Running on NVMe SSDs," in *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 313-326, 1 July-Sept. 2018.

[33] C. Anderson, “Docker software engineering,” IEEE Softw., vol. 32, no. 3, p. 102–c3, 2015.

[34] P. Di Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame, “The impact of Docker containers on the performance of genomic pipelines,” PeerJ, vol. 3, 2015, Art. no. e1273.

[35] J. Fink, “Docker: A software as a service, operating system-level virtualization framework,” Code4Lib J., vol. 25, p. 29, 2014.

[36] A. M. Joy, "Performance comparison between Linux containers and virtual machines," *2015 International Conference on Advances in Computer Engineering and Applications*, Ghaziabad, 2015, pp. 342-346.

[37] W. Felter, A. Ferreira, R. Rajamony and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, PA, 2015, pp. 171-172.

[38] S. Kamijo, Y. Matsushita, K. Ikeuchi, and M. Sakauchi, ”Traffic monitoring and accident detection at intersections,” IEEE transactions on Intelligent transportation systems, vol. 1, pp. 108-118, 2000.

[39] T. Brosnan and D.-W. Sun, ”Improving quality inspection of food products by computer vision - a review,” Journal of food engineering, vol. 61, pp. 3-16, 2004.

[40] M. Trajkovic, A. J. Colmenarez, S. Gutta, and K. I. Trovato, ”Computer vision based parking assistant,” ed: Google Patents, 2004.

[41] W. W. Boles and B. Boashash, ”A human identification technique using images of the iris and wavelet transform,” IEEE transactions on signal processing, vol. 46, pp. 1185-1188, 1998.

[42] CVonline: Image Databases. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm

[43] M. J. McAuliffe, F. M. Lalonde, D. McGarry, W. Gandler, K. Csaky and B. L. Trus, "Medical Image Processing, Analysis and Visualization in clinical research," Proceedings 14th IEEE Symposium on Computer-Based Medical Systems. CBMS 2001, Bethesda, MD, USA, 2001, pp. 381-386.

[44] A. Leff and J. T. Rayfield, "Web-application development using the Model/View/Controller design pattern," *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, Seattle, WA, USA, 2001, pp. 118-127.

[45] E. Curry and P. Grace, "Flexible Self-Management Using the Model-View-Controller Pattern," in IEEE Software, vol. 25, no. 3, pp. 84-90, May-June 2008.

[46] J. Wojciechowski, B. Sakowicz, K. Dura and A. Napieralski, "MVC model, struts framework and file upload issues in web applications based on J2EE platform," Proceedings of the International Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004., Lviv-Slavsko, Ukraine, 2004, pp. 342-345.

[47] Nikolas Havrikov, Alessio Gambi, Andreas Zeller, Andrea Arcuri, and Juan Pablo Galeotti. 2017. Generating unit tests with structured system interactions. In Proceedings of the 12th International Workshop on Automation of Software Testing (AST '17). IEEE Press, Piscataway, NJ, USA, 30-33.

[48] M. Rodríguez-Martinez, "Experiences with the Twitter Health Surveillance (THS) System," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 376-383.

[49] K. Tüzes-Bölöni, Z. Borsay, C. Sulyok and K. Simon, "Diaspora Mapping and Collaboration Platform for Expatriates," 2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, 2018, pp. 000027-000032.

[50] H. Pu, S. Su, M. Lee and C. Lin, "Generation of personalized learning paths with a concept map: A case study of the IEEE floating-point standard," 2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, 2018, pp. 184-187.

[51] C. Pan and C. Lin, "Designing and implementing a computerized adaptive testing system with an MVC framework: A case study of the IEEE floating-point standard," 2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, 2018, pp. 609-612.

[52] Anurag Dwarakanath, Upendra Chintala, Shrikanth N. C., Gurdeep Virdi, Alex Kass, Anitha Chandran, Shubhashis Sengupta, and Sanjoy Paul. 2015. CrowdBuild: a methodology for enterprise software development using crowdsourcing. In Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering (CSI-SE '15). IEEE Press, Piscataway, NJ, USA, 8-14.

[53] G. Orsini, D. Bade and W. Lamersdorf, "Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading," 2015 8th IFIP Wireless and Mobile Networking Conference (WMNC), Munich, 2015, pp. 112-119.

[54] M. Shetty and S. R. Naik, "Ontology representation of Amazon S3 objects for metadata enrichment," 2016 International Conference on Computing, Analytics and Security Trends (CAST), Pune, 2016, pp. 507-512.

[55] Verginadis, Y., Michalas, A., Gouvas, P. et al. J Grid Computing (2017) 15: 219. https://doi.org/10.1007/s10723-017-9394-2

[56] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015.

[57] Manjunath R., Tejus, Channabasava R.K and Balaji S., "A Big Data MapReduce Hadoop distribution architecture for processing input splits to solve the small data problem," 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Bangalore, 2016, pp. 480-487.

[58] Wenwen Li, Sizhe Wang, Vidit Bhatia, PolarHub: A large-scale web crawling engine for OGC service discovery in cyberinfrastructure, Computers, Environment and Urban Systems, Volume 59, 2016, Pages 195-207, ISSN 0198-9715.

[59] J. Toth *et al.*, "An online benchmark system for image processing algorithms," *2014 5th IEEE Conference on Cognitive Infocommunications (CogInfoCom)*, Vietri sul Mare, 2014, pp. 377-382.

[60] http://www.docker.com

[61] F. A. Masoud, D. H. Halabi and D. H. Halabi, "ASP.NET and JSP Frameworks in Model View Controller Implementation," *2006 2nd International Conference on Information & Communication Technologies*, Damascus, 2006, pp. 3593-3598.

[62] I. Sarker and K. Apu, "MVC Architecture Driven Design and Implementation of Java Framework for Developing Desktop Application," International Journal of Hybrid Information Technology, pp. Vol.7, No.5, 3 17-322, 20 14.

[63] Chhikara, "A Web Architectural Study of HTML5 with MVC Framework," International Journal oj Advanced Research in Computer Science and Software engineering, pp. 45 1 - 454, 20 13.

[64] H. Komara, B. Hendradjaya and G. A. P. Saptawati, "Dynamic generic web pattern for multi platform," *2016 International Conference on Data and Software Engineering (ICoDSE)*, Denpasar, 2016, pp. 1-5.

[65] W. Ning, L. Liming, W. Yanzhang, W. Yi-bing and W. Jing, "Research on the Web Information System Development Platform Based on MVC Design Pattern," *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Sydney, NSW, 2008, pp. 203-206.

[66] N. Kupp and Y. Makris, "Applying the Model-View-Controller Paradigm to Adaptive Test," in *IEEE Design & Test of Computers*, vol. 29, no. 1, pp. 28-35, Feb. 2012.

[67] A. Holzinger, K. H. Struggl and M. Debevc, "Applying Model-View-Controller (MVC) in design and development of information systems: An example of smart assistive script breakdown in an e-Business application," *2010 International Conference on e-Business (ICE-B)*, Athens, 2010, pp. 1-6.

[68] Yonglei Tao, "Component- vs. application-level MVC architecture," *32nd Annual Frontiers in Education*, Boston, MA, USA, 2002, pp. T2G-T2G.

[69] H. Mcheick and Y. Qi, "Dependency of components in MVC distributed architecture," *2011 24th Canadian Conference on Electrical and Computer Engineering (CCECE)*, Niagara Falls, ON, 2011, pp. 000691-000694.

[70] M. S. Levin, "Modular system synthesis: example for composite packaged software," in *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 544-553, Nov. 2005.

[71] J. Grundy, G. Kaefer, J. Keung and A. Liu, "Guest Editors' Introduction: Software Engineering for the Cloud," in *IEEE Software*, vol. 29, no. 2, pp. 26-29, March-April 2012.

[72] S. Patel, W. Chu and R. Baxter, "A measure for composite module cohesion," *International Conference on Software Engineering*, Melbourne, Australia, 1992, pp. 38-48.

[73] P. S. Sandhu, Aashima, P. Kakkar and S. Sharma, "A survey on Software Reusability," *2010 International Conference on Mechanical and Electrical Technology*, Singapore, 2010, pp. 769-773.

[74] V. Kadary, S. Markel, S. Koenig and A. Yehudai, "Software and system reusability," *[1988] Proceedings. The Third Israel Conference on Computer Systems and Software Engineering*, Tel-Aviv, Israel, 1988, pp. 115-.

[75] T. Xin and L. Yang, "A framework of software reusing engineering management," *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, London, 2017, pp. 277-282.

[76] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," in IEEE Transactions on Industrial Informatics, vol. 9, no. 3, pp. 1234-1249, Aug. 2013.

[77] G. F. Hoffnagle and W. E. Beregi, "Automating the software development process," in IBM Systems Journal, vol. 24, no. 2, pp. 102-120, 1985.

[78] R. Jetley, A. Nair, P. Chandrasekaran and A. Dubey, "Applying software engineering practices for development of industrial automation applications," 2013 11th IEEE International Conference on Industrial Informatics (INDIN), Bochum, 2013, pp. 558-563.

[79] D. Winkler, M. Schönbauer and S. Biffl, "Towards Automated Process and Workflow Management: A Feasibility Study on Tool-Supported and Automated Engineering Process Modeling Approaches," 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, 2014, pp. 102-110.

[80] U. Katzke and B. Vogel-Heuser, "Design and application of an engineering model for distributed process automation," Proceedings of the 2005, American Control Conference, 2005., Portland, OR, USA, 2005, pp. 2960-2965 vol. 4.

[81] C. M. de la Osa, G. G. Anagnostopoulos, M. Togneri, M. Deriaz and D. Konstantas, "Positioning evaluation and ground truth definition for real life use cases," *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Alcala de Henares, 2016, pp. 1-7.

[82] E. Pasolli, F. Melgani, N. Alajlan and N. Conci, "Optical Image Classification: A Ground-Truth Design Framework," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 51, no. 6, pp. 3580-3597, June 2013.

[83] E. Pasolli, F. Melgani, N. Alajlan and N. Conci, "Optical Image Classification: A Ground-Truth Design Framework," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 51, no. 6, pp. 3580-3597, June 2013.

[84] E. H. B. Smith and C. An, "Effect of "Ground Truth" on Image Binarization," *2012 10th IAPR International Workshop on Document Analysis Systems*, Gold Cost, QLD, 2012, pp. 250-254.

[85] M. Zeeshan, M. Majid, I. F. Nizami, S. M. Anwar, I. Ud Din and M. Khurram Khan, "A Newly Developed Ground Truth Dataset for Visual Saliency in Videos," in *IEEE Access*, vol. 6, pp. 20855-20867, 2018.

[86] A. Levin, Y. Weiss, F. Durand and W. T. Freeman, "Understanding and evaluating blind deconvolution algorithms," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, 2009, pp. 1964-1971.

[87] Aruna Devi B., Pallikonda Rajasekaran M. (2019) Performance Evaluation of MRI Pancreas Image Classification Using Artificial Neural Network (ANN). In: Satapathy S., Bhateja V., Das S. (eds) Smart Intelligent Computing and Applications. Smart Innovation, Systems and Technologies, vol 104. Springer, Singapore

[88] S. Boorboor, S. Jadhav, M. Ananth, D. Talmage, L. Role and A. Kaufman, "Visualization of Neuronal Structures in Wide-Field Microscopy Brain Images," in *IEEE*

Transactions on Visualization and Computer Graphics, vol. 25, no. 1, pp. 1018-1028, Jan. 2019.

[89] Murray Woodside, Greg Franks, and Dorina C. Petriu. 2007. The Future of Software Performance Engineering. In 2007 Future of Software Engineering (FOSE '07). IEEE Computer Society, Washington, DC, USA, 171-187. DOI: https://doi-org.qe2a-proxy.mun.ca/10.1109/FOSE.2007.32

[90] U. Krishnaswamy and I. D. Scherson, "A framework for computer performance evaluation using benchmark sets," in *IEEE Transactions on Computers*, vol. 49, no. 12, pp. 1325-1338, Dec. 2000.

[91] https://github.com/spotify/docker-client

[92] H. Zhu and I. Bayley, "If Docker is the Answer, What is the Question?," 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), Bamberg, 2018, pp. 152-163

[93] S. M. Shariff, H. Li, C. Bezemer, A. E. Hassan, T. H. D. Nguyen and P. Flora, "Improving the Testing Efficiency of Selenium-Based Load Tests," 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), Montreal, QC, Canada, 2019, pp. 14-20

[94] P. Ramya, V. Sindhura and P. V. Sagar, "Testing using selenium web driver," 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, 2017, pp. 1-7

[95] I. Altaf, J. A. Dar, F. u. Rashid and M. Rafiq, "Survey on selenium tool in software testing," 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), Noida, 2015, pp. 1378-1383

[96] R. Chen and H. Miao, "A Selenium based approach to automatic test script generation for refactoring JavaScript code," 2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS), Niigata, 2013, pp. 341-346,

[97] R. A. Razak and F. R. Fahrurazi, "Agile testing with Selenium," 2011 Malaysian Conference in Software Engineering, Johor Bahru, 2011, pp. 217-219

[98] C. McMahon, "History of a Large Test Automation Project Using Selenium," 2009 Agile Conference, Chicago, IL, 2009, pp. 363-368

[99] A. Bruns, A. Kornstadt and D. Wichmann, "Web Application Tests with Selenium," in IEEE Software, vol. 26, no. 5, pp. 88-91, Sept.-Oct. 2009

[100] A. Holmes and M. Kellogg, "Automating functional tests using Selenium," AGILE 2006 (AGILE'06), Minneapolis, MN, 2006, pp. 6 pp.-275

[101] https://hackr.io/blog/complete-guide-selenium-webdriver

[102] https://www.jenkins.io/

[103] V. Armenise, "Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery," 2015 IEEE/ACM 3rd International Workshop on Release Engineering, Florence, 2015, pp. 24-27

[104] https://codefresh.io/continuous-deployment/codefresh-versus-jenkins/#:~:text=Codefresh%20has%20a%20bigger%20scope,Continuous%20Integration%20and%20Continuous%20Delivery.

[105] https://www.jenkins.io/blog/2017/09/25/declarative-1/

Appendix

A. Haar Cascade Face Detection Application using C and OpenCV

```
#include "opencv2/objdetect/objdetect.hpp"

#include "opencv2/highgui/highgui.hpp"

 #include "opencv2/imgproc/imgproc.hpp"

#include <fstream>

 #include <iostream>

 #include <stdio.h>


 using namespace std;

 using namespace cv;


/** Function Headers */

 void detectAndDisplay( String outputFramePath, Mat frame , const char* outputtxtFilePath);


/** Global variables */

 String face_cascade_name = "haar_cascade_frontalface_alt.xml";

 //String eyes_cascade_name = "haar_cascade_eye_tree_eyeglasses.xml";

 CascadeClassifier face_cascade;

 //CascadeClassifier eyes_cascade;

// string window_name = "Capture - Face detection";
```

```
// RNG rng(12345);


 /** @function main */
 int main( int argc, const char** argv )
 {
   CvCapture* capture;
   Mat frame;


   //-- 1. Load the cascades
   if( !face_cascade.load( face_cascade_name ) ){ printf("--(!)Error loading
haar_cascade_frontalface_alt.xml\n"); return -1; };
  //  if( !eyes_cascade.load( eyes_cascade_name ) ){ printf("--(!)Error loading
haar_cascade_eye_tree_eyeglasses.xml\n"); return -1; };
frame = imread(argv[1], CV_LOAD_IMAGE_COLOR); // here we'll know the method used
(allocate matrix)
detectAndDisplay(argv[2], frame, argv[3]);
   return 0;
 }


/** @function detectAndDisplay */
void detectAndDisplay(String outputFramePath , Mat frame, const char* outputtxtFilePath)
{
```

```
 std::vector<Rect> faces;
 Mat frame_gray;

 cvtColor( frame, frame_gray, CV_BGR2GRAY );
 equalizeHist( frame_gray, frame_gray );

 //-- Detect faces
 face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(30,
30) );
std::ofstream myfile;
 myfile.open (outputtxtFilePath, std::ios_base::app);
 if (myfile.is_open())
 {
 for( size_t i = 0; i < faces.size(); i++ )
 {
  // Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );
   //ellipse( frame, center, Size( faces[i].width*0.5, faces[i].height*0.5), 0, 0, 360, Scalar( 255, 0,
255 ), 4, 8, 0 );
        Rect r=Rect(faces[i].x,faces[i].y,faces[i].width, faces[i].height);
        rectangle(frame, r,Scalar( 255, 0, 255 ), 4, 8, 0  );
        Mat faceROI = frame_gray( faces[i] );
myfile << faces[i].x << " " <<  faces[i].y << " " << faces[i].width << " " << faces[i].height <<"\n";
```

```
/*      std::vector<Rect> eyes;

   //-- In each face, detect eyes
   eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |CV_HAAR_SCALE_IMAGE, Size(30,
30) );

  for( size_t j = 0; j < eyes.size(); j++ )
    {
     Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5, faces[i].y + eyes[j].y +
eyes[j].height*0.5 );
     int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
     circle( frame, center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
    }
*/
 }

//myfile << "----------------------------\n";
   myfile.close();
 }
 else cout << "Unable to open the output file";
 imwrite(outputFramePath,frame);
```

```
 //-- Show what you got

 //imshow( window_name, frame );

}
```

B. A Sample for the haar_cascade_frontalface_alt.xml

The xml file needed for the previous Software Application for face detection using Haar Cascade using C and OpenCV. These are only parts of the file as it is a huge file.

```
<opencv_storage>
<cascade type_id="opencv-cascade-classifier"><stageType>BOOST</stageType>
 <featureType>HAAR</featureType>
 <height>20</height>
 <width>20</width>
 <stageParams>
  <maxWeakCount>213</maxWeakCount></stageParams>
 <featureParams>
  <maxCatCount>0</maxCatCount></featureParams>
 <stageNum>22</stageNum>
 <stages>
  <_>
   <maxWeakCount>3</maxWeakCount>
   <stageThreshold>8.2268941402435303e-01</stageThreshold>
   <weakClassifiers>
    <_>
     <internalNodes>
```

```
      0 -1 0 4.0141958743333817e-03</internalNodes>
      <leafValues>
      3.3794190734624863e-02 8.3781069517135620e-01</leafValues></_>
    <_>
      <internalNodes>
      0 -1 1 1.5151339583098888e-02</internalNodes>
      <leafValues>
      1.5141320228576660e-01 7.4888122081756592e-01</leafValues></_>
    <_>
      <internalNodes>
      0 -1 2 4.2109931819140911e-03</internalNodes>
      <leafValues>
      9.0049281716346741e-02 6.3748198747634888e-
01</leafValues></_></weakClassifiers></_>
  <_>
    <maxWeakCount>16</maxWeakCount>
    <stageThreshold>6.9566087722778320e+00</stageThreshold>
    <weakClassifiers>
    <_>
      <internalNodes>
      0 -1 3 1.6227109590545297e-03</internalNodes>
      <leafValues>
```

6.9308586418628693e-02 7.1109461784362793e-01</leafValues></_>
<_>
<internalNodes>
0 -1 4 2.2906649392098188e-03</internalNodes>
<leafValues>
1.7958030104637146e-01 6.6686922311782837e-01</leafValues></_>
<_>
<internalNodes>
0 -1 5 5.0025708042085171e-03</internalNodes>
<leafValues>
1.6936729848384857e-01 6.5540069341659546e-01</leafValues></_>

.
.
.
.
.

<_>
<rects>
<_>
4 3 8 3 -1.</_>
<_>

8 3 4 3 2.</_></rects></_>
<_>
<rects>
<_>
0 4 20 6 -1.</_>
<_>
0 4 10 6 2.</_></rects></_>
<_>
<rects>
<_>
9 14 1 3 -1.</_>
<_>
9 15 1 1 3.</_></rects></_>
<_>
<rects>
<_>
8 14 4 3 -1.</_>
<_>
8 15 4 1 3.</_></rects></_>
<_>
<rects>
<_>

```
      0 15 14 4 -1.</_>
      <_>
      0 17 14 2 2.</_></rects></_>
    <_>
     <rects>
      <_>
      1 14 18 6 -1.</_>
      <_>
      1 17 18 3 2.</_></rects></_>
    <_>
     <rects>
      <_>
      0 0 10 6 -1.</_>
      <_>
      0 0 5 3 2.</_>
      <_>
      5 3 5 3 2.</_></rects></_></features></cascade>
</opencv_storage>
```

C. Result of Haar Cascade Face Detection Application using C and OpenCV

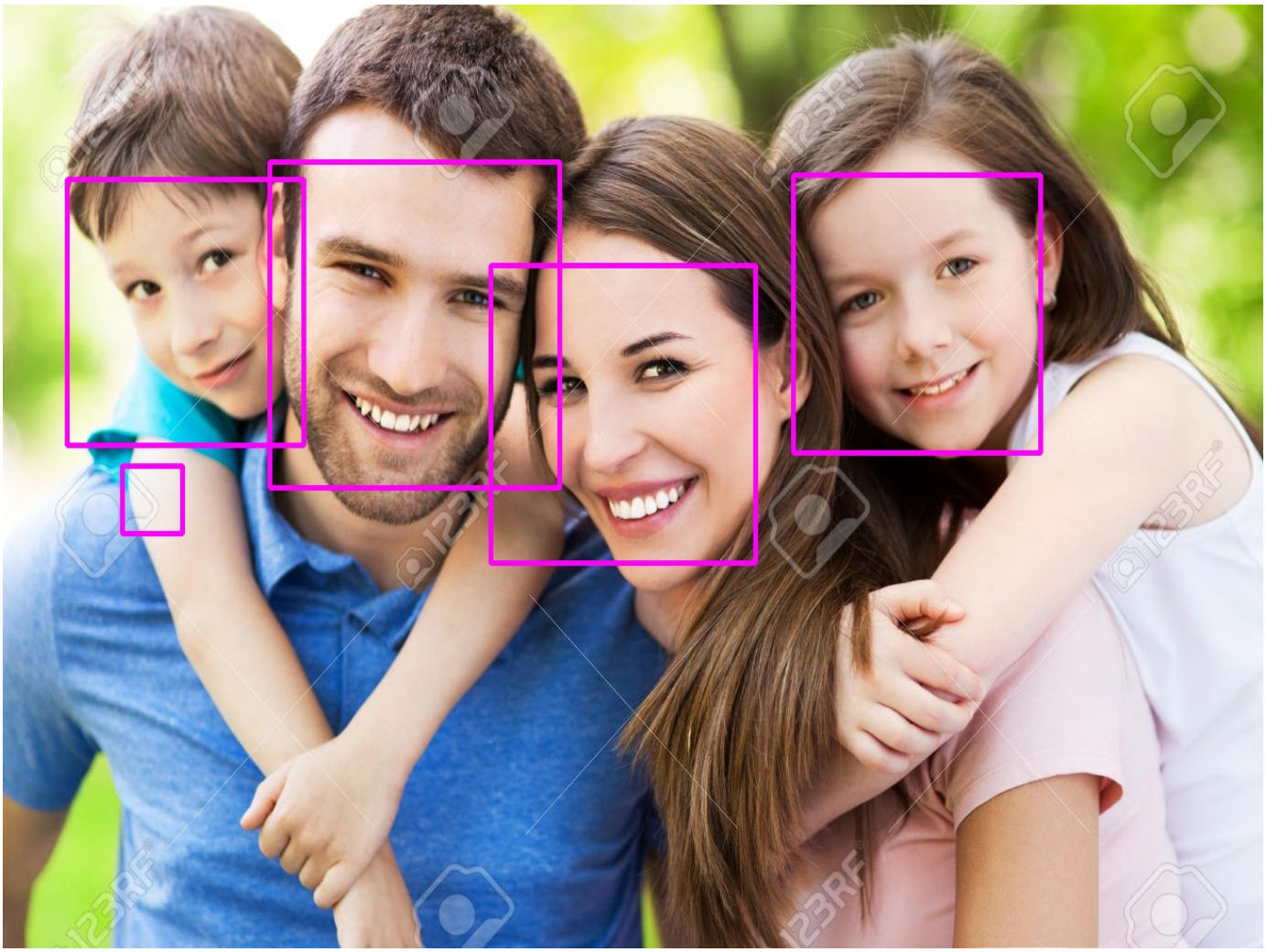


Figure C.1: Result of Haar Cascade Face Detection Application using C and OpenCV

D. Haar Cascade Face Detection Application using Python

```
import cv2

import sys

# Get user supplied values

imagePath = sys.argv[1]

cascPath = "haar_cascade_frontalface_default.xml"


# Create the haar cascade

faceCascade = cv2.CascadeClassifier(cascPath)


# Read the image

image = cv2.imread(imagePath)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# Detect faces in the image

faces = faceCascade.detectMultiScale(

    gray,

    scaleFactor=1.1,

    minNeighbors=5,
```

```
    minSize=(30, 30),
    flags = cv2.cv.CV_HAAR_SCALE_IMAGE
)
print("Found {0} faces!".format(len(faces)))
f = open(sys.argv[3],"a+b")
#f = open("output.txt","a+b")
# Draw a rectangle around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
    f.write("%d %d %d %d \r\n" % (x , y, w, h) )
    #cv2.circle(image, (x+w/2, y+h/2), (h/2), (0, 255, 0), 3)
#cv2.imwrite("imageoutput.png", image);

cv2.imwrite(sys.argv[2], image);
#f.write("----------------------------\n");
#cv2.imshow("Faces found", image)
f.close()
cv2.waitKey(0)
```

E. A Sample of haar_cascade_frontalface_default.xml


```
<opencv_storage>
<cascade type_id="opencv-cascade-classifier"><stageType>BOOST</stageType>
 <featureType>HAAR</featureType>
 <height>24</height>
 <width>24</width>
 <stageParams>
  <maxWeakCount>211</maxWeakCount></stageParams>
 <featureParams>
  <maxCatCount>0</maxCatCount></featureParams>
 <stageNum>25</stageNum>
 <stages>
  <_>
   <maxWeakCount>9</maxWeakCount>
   <stageThreshold>-5.0425500869750977e+00</stageThreshold>
   <weakClassifiers>
    <_>
     <internalNodes>
      0 -1 0 -3.1511999666690826e-02</internalNodes>
     <leafValues>
      2.0875380039215088e+00 -2.2172100543975830e+00</leafValues></_>
```

<_>

<internalNodes>

0 -1 1 1.2396000325679779e-02</internalNodes>

<leafValues>

-1.8633940219879150e+00 1.3272049427032471e+00</leafValues></_>

<_>

<internalNodes>

0 -1 2 2.1927999332547188e-02</internalNodes>

<leafValues>

-1.5105249881744385e+00 1.0625729560852051e+00</leafValues></_>

<_>

<internalNodes>

0 -1 3 5.7529998011887074e-03</internalNodes>

<leafValues>

-8.7463897466659546e-01 1.1760339736938477e+00</leafValues></_>

<_>

<internalNodes>

0 -1 4 1.5014000236988068e-02</internalNodes>

<leafValues>

-7.7945697307586670e-01 1.2608419656753540e+00</leafValues></_>

<_>

<internalNodes>

0 -1 5 9.9371001124382019e-02</internalNodes>
<leafValues>
5.5751299858093262e-01 -1.8743000030517578e+00</leafValues></_>
<_>
<internalNodes>
0 -1 6 2.7340000960975885e-03</internalNodes>
<leafValues>
-1.6911929845809937e+00 4.4009700417518616e-01</leafValues></_>
<_>
<internalNodes>
0 -1 7 -1.8859000876545906e-02</internalNodes>
<leafValues>
-1.4769539833068848e+00 4.4350099563598633e-01</leafValues></_>
.
.
.
.
.
<_>
<rects>
<_>
0 18 18 2 -1.</_>

<_>

0 19 18 1 2.</_></rects></_>

<_>

<rects>

<_>

3 15 19 3 -1.</_>

<_>

3 16 19 1 3.</_></rects></_>

<_>

<rects>

<_>

0 13 18 3 -1.</_>

<_>

0 14 18 1 3.</_></rects></_>

<_>

<rects>

<_>

15 17 9 6 -1.</_>

<_>

15 19 9 2 3.</_></rects></_>

<_>

<rects>

<_>
<_>
0 17 9 6 -1.</_>
<_>
0 19 9 2 3.</_></rects></_>
<_>
<rects>
<_>
12 17 9 6 -1.</_>
<_>
12 19 9 2 3.</_></rects></_>
<_>
<rects>
<_>
3 17 9 6 -1.</_>
<_>
3 19 9 2 3.</_></rects></_>
<_>
<rects>
<_>
16 2 3 20 -1.</_>
<_>
17 2 1 20 3.</_></rects></_>

<_>

<rects>

<_>

0 13 24 8 -1.</_>

<_>

0 17 24 4 2.</_></rects></_>

<_>

<rects>

<_>

9 1 6 22 -1.</_>

<_>

12 1 3 11 2.</_>

<_>

9 12 3 11 2.</_></rects></_></features></cascade>

</opencv_storage>

F. Image Result of Haar Cascade Face Detection Application using Python

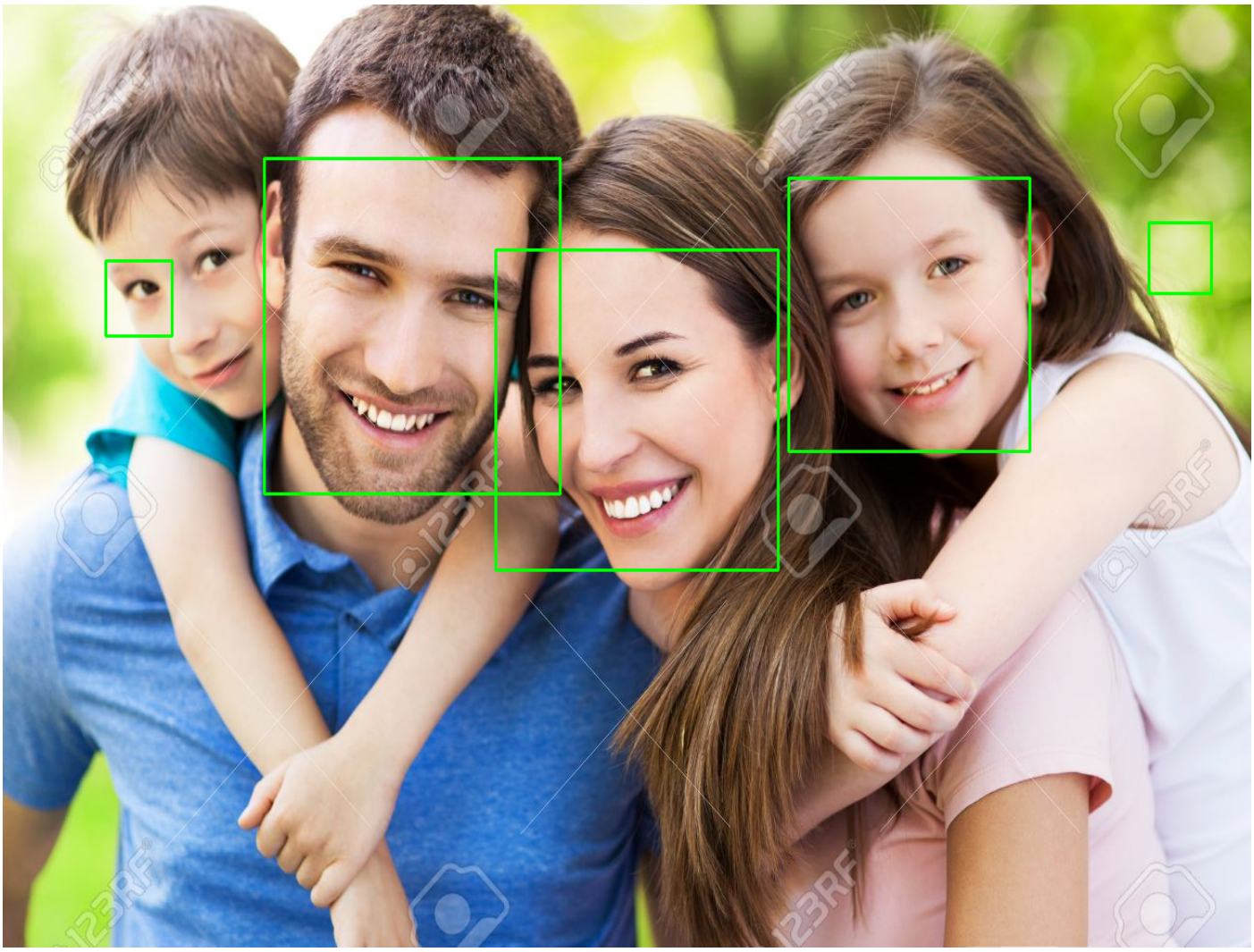


Figure F: Result of Haar Cascade Face Detection Application using Python