Grenze International Journal of Engineering and Technology, June Issue

Enhancing Software Testing Efficiency with ELASTEST: A Distributed Systems Approach

Sagar Bahad[1], Surekha Tadse[2] and Dr. Pankaj Chandankhede[3]
[1]PG Scholar, Department of Electronics & Telecommunication Engineering, G H Raisoni College of Engineering, Nagpur, Maharashtra, India
Email: sagar.bahad.mtechcom@ghrce.raisoni.net
[2-3]Assistant Professor, Department of Electronics & Telecommunication Engineering, G H Raisoni College of Engineering, Nagpur, Maharashtra, India
Email: surekha.tadse@raisoni.net, pankaj.chandankhede@raisoni.net

***Abstract*—Computer program testing may be a basic perspective of computer program improvement, guaranteeing the unwavering quality, usefulness, and execution of applications. However, testing large-scale distributed systems poses unique challenges due to their complexity and dynamic nature. In this paper, we introduce ELASTEST, a comprehensive testing framework designed to address these challenges and enhance testing efficiency in distributed systems. ELASTEST leverages a distributed systems approach, providing a flexible and scalable platform for various testing scenarios, including functional, performance, and security testing. By seamlessly integrating with existing development workflows and supporting containerized environments, ELASTEST enables continuous and automated testing processes, improving the speed and accuracy of testing cycles. We present the architecture, key features, and implementation details of ELASTEST, along with case studies demonstrating its effectiveness in real-world testing scenarios. Our findings highlight the significance of ELASTEST in streamlining software testing practices and facilitating the development of robust and resilient distributed systems.**

***Index Terms*— Elastest, User Impersonation, Cloud-based Testing, Test Automation, Selenium Integration, Docker Containers, Scalable Infrastructure.**

I. INTRODUCTION

Software testing is crucial for ensuring the reliability of applications, especially in the context of distributed systems with their inherent complexities. ELASTEST emerges as a solution to address these challenges by offering a comprehensive testing platform tailored for distributed environments. This paper explores ELASTEST's architecture, key features, and implementation details, emphasizing its role in enhancing testing efficiency. By integrating with existing workflows and supporting containerized environments, ELASTEST enables continuous and automated testing, accelerating software delivery while maintaining high quality. Through case studies and experiments, we demonstrate ELASTEST's effectiveness in identifying and mitigating issues across distributed systems. By leveraging a distributed systems approach, ELASTEST empowers developers and testers to adopt best practices, ensuring the reliability and performance of distributed applications. This paper provides insights into ELASTEST's capabilities and its potential to revolutionize software testing in distributed systems.

Grenze ID: 01.GIJET.10.2.523

II. Background

Selenium WebDriver is widely acclaimed as an open-source tool essential for web testing. It boasts versatility across a wide range of web browsers, including Chrome, Firefox, Opera, Edge, and Safari. Moreover, it offers compatibility with various programming languages such as Java, C#, Python, Ruby, PHP, Perl, or JavaScript. By tapping into the native support of each browser, WebDriver seamlessly interacts with them, following the JSON Wire protocol and facilitating communication through JSON messages over HTTP using browser-specific binaries. The integration of W3C WebDriver recommendations further enhances this process.

Furthermore, Selenium Grid enhances its usefulness by enabling the execution of WebDriver tests remotely across distributed computers. Comprising nodes operating on different operating systems and browsers, this Grid is managed by a central hub or Selenium server. Alongside, the Selenium ecosystem has nurtured the development of numerous complementary projects.

- Protractor: Angular application testing framework based on WebDriverJS.
- Nightwatch.js: A custom API based on Node.js for W3C WebDriver.
- WebDriverIO: A custom implementation of W3C WebDriver written in JavaScript.
- Appium: A testing framework for automating native, hybrid and mobile web applications.

Many companies offer Selenium as a Service (SaaS) models (such as Saace LABS and BrowserStack) that provide cloud-based remote testing for different functions, browser types, and version solutions.

III. Challenges While Testing Cloud Applications

Testing distributed systems, particularly those deployed on cloud infrastructure, poses significantly greater challenges compared to testing monolithic applications. Even the simplest integration tests for distributed systems necessitate the startup and shutdown of numerous services, along with the requirement for various tools such as automated browsers for end-to-end (e2e) testing. The associated costs and time commitments for testing such applications can be prohibitively high, making it unfeasible for many companies to undertake [2]. Moreover, the integration of multiple tools and the setup of a testing infrastructure typically demand substantial hours of work.

The complexity escalates further when UI testing is involved. User impersonation services, like browsers, become indispensable for evaluating critical paths within applications, such as the payment process in an e-commerce application. Challenges arise in maintaining these browsers up to date, adding to the continuous efforts required for sustaining the testing infrastructure.

However, even if a minimal infrastructure capable of running these tests on a cloud application is established, it's insufficient. In the event of a test failure, it's essential to identify the specific component responsible, necessitating the collection of logs from the system's various components. However, logs may not suffice if the failure stems from underlying infrastructure issues. Determining if the failure resulted from, say, insufficient memory requires the collection of metrics as well. Thus, in addition to the requisite tooling mentioned earlier, there's a need for infrastructure capable of extracting and storing information from different components (both logs and metrics) for subsequent analysis. Furthermore, meaningful visualizations must be developed atop this information to facilitate effective analysis.

IV. Enhancing Testing Efficiency with UIaaS: A Layered Approach

The ElasTest initiative endeavors to transform the testing procedures for extensive and varied software, enhancing both efficiency and efficacy. It offers a holistic methodology for automating end-to-end testing throughout the entire development cycle, encompassing administration, deployment, testing, and maintenance.

An important feature of ElasTest is its ability to simulate user interaction with web and mobile applications. This feature is seamlessly integrated into the platform and is designed to comply with W3C WebDriver recommendations, ensuring compatibility with existing tools and technologies.

Building on the standard WebDriver protocol checks from Selenium and Appium for browsers and mobile devices, ElasTest extends this foundation to provide additional functionality. Within ElasTest, the concept of User Interface as a Service (UIaaS) is integrated as a layer, aligning with the NIST delineation of cloud computing, which includes Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

This layered process allows ElasTest to simplify and complete the development of testing processes, ultimately supporting the advancement of software testing.

A. Software as a Service

Software as a Service (SaaS) is a cloud computing approach where software is not installed upon a desktop or laptop machine, but rather hosted in the cloud and accessed via the Internet. Under this arrangement, customers pay for the programme as they use it, typically on a monthly or yearly basis. Because software developers handle

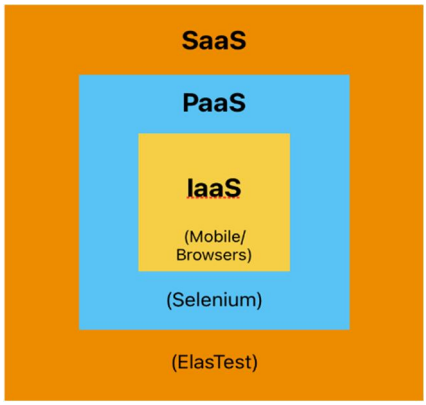

Fig. A. Vector Representation

protection, upgrades, and maintenance, SaaS relieves businesses of the requirement for buying hardware and software licences.

Scalability is just one of SaaS's benefits, as it enables companies to quickly modify their software to suit their requirements. Furthermore, it enables remote working as well as cooperation from any location with an internet connection. Productivity tools, customer relationship management (CRM), software for enterprise resource planning (ERP), and other features are all included in SaaS initiatives.

Overall, SaaS provides benefits, flexibility and convenience to businesses and individuals, making it a popular choice in today's digital environment. However, when using SaaS solutions, network connection dependency and data security concerns are an issue that must be taken into account.

ElasTest's user simulation service was created as a comprehensive service as a service (SaaS). Designers don't have to worry about installing software, scaling, or computer resources using this approach. ElasTest provides universal access to services through a web interface to ensure easy user access. The server-side architecture is built around microservices and exposes functionality through a REST API.

ElasTest's scanner service has many uses. First, ElasTest GUI supports interactive manual testing using a browser. The GUI communicates with the ElasTest backend to get the browser type (like Chrome or Firefox) and model. This information is presented to selected users and allows them to launch and interact with certain browser sections. The browser interface is built into the ElasTest GUI using the HTML5 Canvas element. Graphics browser sessions are sent via virtual network computing (VNC) desktop sharing, specifically using the VNC client. This option is good because noVNC runs on WebSocket, which supports two-way communication between client and server. This enables interaction between the user and the browser through the ElasTest GUI. Additionally, ElasTest automatically records remote sessions, which are permanently stored and accessible to testers via the GUI.

Furthermore, automated testing can be done with SaaS browsers. This is made simple by ElasTest, which offers a Hub URL for Selenium testing. Use Selenium's standard needed functionality technique to configure the type and version of the necessary browsers. Here's an illustration of a JUnit 4th test case:

B. Platform as a service

Platform on Demand (PaaS) is an architecture for cloud computing that gives users a platform to create, execute, and maintain applications without having to deal with the hassles of creating and maintaining supporting infrastructure. Tools for application creation, deployment, and management, including as databases, middleware, methods of development, and languages of programming, are frequently included in PaaS offerings.

With PaaS, developers can focus on writing code instead of managing processes, speeding up the development process and reducing time to market. PaaS also offers scalability, allows applications to easily respond to changing needs, and facilitates collaboration between development teams by providing a cyclical development and version control environment.

Overall, PaaS offers better performance, speed, and cost-effectiveness than other traditional on-premises infrastructures, making it a good choice for organizations looking to improve their application development and delivery processes.

ElasTest provides tools that broaden the W3C WebDriver guidelines on the server side in order to facilitate the Software as a Service (SaaS) paradigm previously defined. ElasTest User Emulator Service (EUS) is a solution

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.junit.Test;
import java.net.URL;

public class SeleniumTest {
    @Test
    public void testWithSaaSBrowser() throws Exception {
        // Define desired capabilities for the browser
        DesiredCapabilities capabilities = DesiredCapabilities.chrome();

        // Set Hub URL for connecting to ElasTest SaaS browser
        URL hubUrl = new URL("http://your-elasTest-hub-url");

        // Create WebDriver instance using RemoteWebDriver
        WebDriver driver = new RemoteWebDriver(hubUrl, capabilities);

        // Perform testing actions
        driver.get("https://example.com");
        // Add more testing steps as needed
        // Close the browser session
        driver.quit();
    }
}
```

intended for use with specialised Selenium/Appium servers. EUS enables further testing of web applications by enhancing the standard WebDriver protocol with additional functions. The EUS implementation of an improved version of WebDriver follows the specifications defined by the OpenAPI project. These specifications are publicly available online and are summarized in Table I. WebDriver extension reports with EUS are divided into three different categories as shown in the table. With this extension, ElasTest aims to provide testers with the ability to optimize Web application testing beyond the limitations of the WebDriver protocol.

TABLE I. COMMANDS

Method	Path	Id
Media Capabilities		
POST	/session/{sessionId}/usermedia	6
GET	/session/{sessionId}/stats	7
POST	/session/{sessionId}/element/{elementId}/latency	8
POST	/session/{sessionId}/element/{elementId}/quality	9
Recordings		
GET	/session/{sessionId /vnc	4
DELETE	/session/{sessionId /vnc	5
Event Subscriptions		
POST	/session/{sessionId}/element/{elementId /event	1
GET	/session/{sessionId}/event/{subscriptionId }	2
DELETE	/session/{sessionId}/event/{subscriptionId}	3

To allow event subscription in the ElasTest User Emulator Service (EUS), we have added a number of APIs. With Command ①, users may utilise these operations to subscribe to Document Object Model (DOM) events such as click and change. Then, Command ② makes it easier to view an event's value for a certain subscription ID, and Command ③ stops event recording. These commands are very useful for testers, and they are not included in the WebDriver suggestion.

Next up, we have a set of operations related to recordings. The URL of a recorded session that EUS has permanently retained is provided by command (12). You have to use Command ⑤ to remove a recording.

Enhancing WebRTC application testing is the main goal of the last series of activities. Web browsing is extended with WebRTC technology, which allows browsers to share media in the moment. Users can provide specific media material (audio and/or video) for WebRTC communication using the first operation in this category. Although synthetic media is available for automated tests in newer browsers such as Firefox and

Chrome, it is not necessarily dependable for realistic testing scenarios. Command ⑥ addresses this by enabling the setting of custom user media for WebRTC, allowing for more realistic testing environments.

Infrastructure as an Service

A cloud concept known as Infrastructure as a Service (IaaS) uses the Internet to deliver virtualized services. Clients can rent virtualized servers, storage, networks, and other infrastructure as needed with Information as a Service (IaaS).

In an IaaS environment, users have full control over the performance of the systems, applications and development processes they use, while the cloud provider is responsible for managing the underlying infrastructure, including data centers, servers and network hardware. .

IaaS has many advantages; among these, scalability allows users to quickly scale infrastructure resources up or down as needed, flexibly to use multiple computing environments, and reduces ongoing costs by eliminating upfront capital in hardware. increasing cost effectiveness.

Overall, IaaS provides a foundation for organizations to build and manage their IT infrastructure more efficiently and cost-effectively, making the infrastructure more dynamic and innovative in the digital age.

The layer that offers the real scanners that the EUS uses is the last one in the implementation strategy. This layer serves as the Docker container's Selenium node. EUS uses the previously mentioned WebDriver interface to control these browsers.

The ElasTest project focuses on the release of modern browsers, including stable, beta and development (nightly) versions. To achieve this goal, the project monitors new browsers such as Chrome and Firefox and creates new images as soon as they become available. Because layers of our approach are continuously applied to existing models, we can make new browsers work without requiring additional attention from testers. It's called Evergreen Docker Browser.

Docker images can be obtained on Docker Hub and are open source. This image is based on the minimum Ubuntu version with additional packages designed to support the required functionality. In addition to browser software (such as Chrome, Firefox), Table II shows the most important software installed in Docker images.

V. End to End Testing Process

A Continuous Integration (CI) server is required to build up an environment for e2e testing. This continuous integration server is in charge of executing the tests along with all of their prerequisites (software and hardware), including the system being tested (SUT). The most well-known open-source continuous integration server in the market, Jenkins, is used in this lesson. Any e2e testing will go through the following steps in terms of the testing procedure itself:

- The SUT has begun.
- The tools for monitoring have begun.
- The examination is conducted. In order to evaluate the user interface, the experiment itself will request and launch multiple browsers on demand.
- The surveillance instruments are deactivated.
- The SUT has come to a stop.
- Every piece of data gathered is linked to the test run within the CI server so that it may be examined at a later time if necessary.
- We can see four distinct things being maintained within these steps:
- SUT: It needs to be launched in advance of the testing and ended thereafter.
- Testing services: Certain services, such as browsers, are required in order for tests to evaluate the SUT.
- Evidence: SUT and environmental data, such as logs and metrics, must be gathered.
- Visualization: Root cause analysis-focused visualizations must be constructed upon raw data since it is meaningless in its raw form.

VI. ElasTest Architecture

The architecture of ElasTest is illustrated in Figure 1 and is made to work with multiple cloud infrastructures, especially OpenStack, Docker, Kubernetes, and Amazon Web Services (AWS). ElasTest makes use of multiple fundamental technologies:

- Docker serving as a container.
- The ElasTest platform's system under test (SUT) and certain services are coordinated by Open Baton.

- The platform's Open Service Broker API (OSBA) facilitates the search, registration, and cancellation of services.
- ElasticSearch and Beats are employed to track the target.
- Asynchronous services communicate with each other via RabbitMQ.

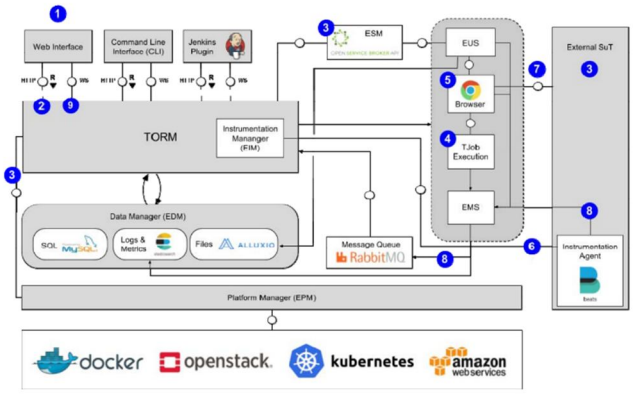

Fig. 1. Architecture of ElasTest

The ElasTest architecture has several main elements:

- Isolating the ElasTest service from the underlying cloud process is the responsibility of the ElasTest Platform Manager (EPM) component.
- ElasTest Service Manager (ESM): Charged with overseeing and locating the different services offered for testing.
- ElasTest User Emulation Service (EUS): This service is intended for use in browser-control tests, particularly those involving Web user interfaces.
- ElasTest TJobs represent tests for various languages and test models using ElasTest's services.
- Instrumentation Manager: This tool helps measure the system under test (SUT) and the platform itself. Its main purpose includes monitoring and troubleshooting such as packet loss, adjusting network bandwidth to simulate the real world, simulating CPU crash and node failure.
- ElasTest Regulation and Recommendation Manager (TORM): As a gateway to ElasTest, TORM organizes all other products and provides various interfaces. These interfaces allow ElasTest users to specify SUT deployment, test execution, and test preferences. Developers can implement end-to-end tests through APIs and access these resources through a graphical user interface (GUI), so testers can create new unit tests based on simple tests.

VII. Demonstration

A YouTube video [8] does a good job of showing a real-life scenario of ElasTest. The system under test (SUT) in this example is a WebRTC application that uses ElasTest and then runs various tests designed to evaluate its performance.

The deployment instructions show how to deploy Docker Compose's SUT on an isolated, firewall-protected server that only allows access via port 443. Custom tests (1,2) are where a web application streams video to the SUT and the SUT interacts with the loopback with up to 30% packet loss.

Applications provided by the distribution (3) work along with the ElasTest Platform ElasTest Orchestration and Recommendation Manager (TORM) is in charge of overseeing it. The administrator who supplies the testing-related resources (mostly Docker containers). To test the browser, use the ElasTest service manager (4) to request it. After that, the ElasTest platform manager is given permission by the ElasTest service manager to allot further resources for the purpose of launching a Docker container on the the testnet that contains a browser (5). The test tells the browser to access the URL corresponding to SUT (7) when it launches.

The online application under consideration is an SUT-managed video conferencing. Several limitations, including packet loss differential, were placed on the test network as part of the testing process. As a result, the video that the browser has received has some frames missing. The test fails because it cannot identify this defect using Quality of Service (QoE) evaluates that the browser gives, such as Structural Similarity (SSIM).

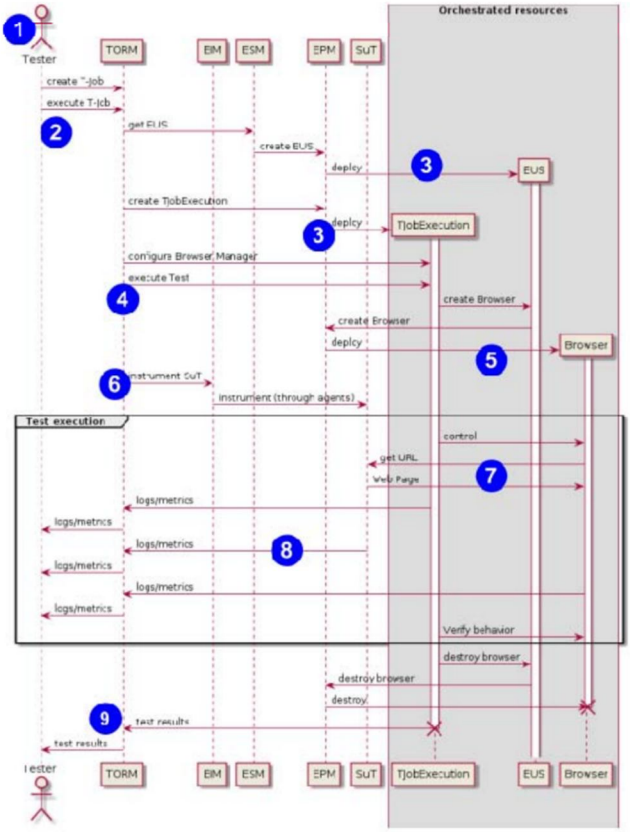

Fig. 2. ElasTest test execution - Sequence flow

Logs and measurements from every application are transmitted to ElasTest for storage in every transaction (8). The ElasTest GUI provides access to these logs and indicators, enabling comparison of various test parameters across contexts (9).

VIII. Conclusion

This article describes recommendations for creating an open-source, user-friendly cloud service. The system was developed within the ElasTest project and is available as a software-as-a-service (SaaS) solution through the ElasTest GUI or directly in Selenium testing. Internally this functionality is supported by the W3C WebDriver approved extension and is supported by external components.
At a low level, the browser probe used in Selenium testing is provided as a Docker container specifically designed to provide advanced testing capabilities to testers.

TABLE II. SOFTWARE'S

Software	*Description*
FFmpeg	Multimedia handling tool used to record browser sessions
Xvfb	X virtual framebuffer, an in–memory display server for Linux
Fluxbox	Window manager for the X Window System
PulseAudio	Sound server system for recordings with audio
x11vnc	VNC server for graphical display (X windows)
noVNC	VNC client supporting WebSocket for direct interaction with browsers from the ElasTest's GUI
Selenoid	Go–lang Selenium Hub required to control browsers

The ElasTest project continues to make progress in this area. The plan now is to release mobile devices in a layered version as Docker images. There is also a focus on increasing the scalability of container support by moving to Kubernetes-based host clusters and moving away from single Docker engine configurations. It is also

aimed to participate in the automation of sensors, actuators and smart devices, allowing their behavior to be simulated in Internet of Things (IoT) systems. Finally, it is planned to create a high-level library to facilitate the use of these functions as JUnit 5 extensions directly from the test scenario.

REFERENCES

[1] García, Boni, Micael Gallego, Carlos Santos, Eduardo Jiménez, Katia Leal, and Luis Fernánez. "Extending WebDriver: A cloud approach." In 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 143-146. IEEE, 2018.

[2] Gortazár, Francisco, Micael Gallego, Boni García, Giuseppe Antonio Carella, Michael Pauls, and Ilie-Daniel Gheorghe-Pop. "Elastest—an open source project for testing distributed applications with failure injection." In 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1-2. IEEE, 2017.

[3] Prabhu Kavin, B., Sagar Karki, S. Hemalatha, Deepmala Singh, R. Vijayalakshmi, M. Thangamani, Sulaima Lebbe Abdul Haleem et al. "Machine learning-based secure data acquisition for fake accounts detection in future mobile communication networks." Wireless Communications and Mobile Computing 2022 (2022): 1-10.

[4] Manoharan, Hariprasath, Sulaima Lebbe Abdul Haleem, S. Shitharth, Pravin R. Kshirsagar, Vineet Tirth, M. Thangamani, and Radha Raman Chandan. "A machine learning algorithm for classification of mental tasks." Computers and Electrical Engineering 99 (2022): 107785.

[5] Nikhate, Purva, Atul R. Deshmukh, and Swapna Choudhari. "Performance Analysis of 2x2 MIMO Systems for Alamouti STBC and Zero Forcing Equalization." In 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), pp. 1-6. IEEE, 2021.

[6] Wankhade, Kapil K., Kalpana C. Jondhale, and Snehlata S. Dongre. "A clustering and ensemble based classifier for data stream classification." Applied Soft Computing 102 (2021): 107076.

[7] Sanjay, Shreyasee Sulakshna, Apurba Pal, Sujit Kishor Chakraborty, and Hasim Ali Khan. "Reduction of shrinkage of self compacting concrete using polycarboxylate ether as shrinkage reducing admixture." Materials Today: Proceedings 60 (2022): 448-451.

[8] Zade, Prasanna, Pranjali Jumle, and Sachin Khade. "Miniaturized High Bandwidth Microstrip Antenna using Metamaterial concept based Technology." In 2021 International Conference on Computational Performance Evaluation (ComPE), pp. 556-559. IEEE, 2021.

[9] Choudhary, Swapna, and Sanjay Dorle. "A quality of service-aware high-security architecture design for software-defined network powered vehicular ad-hoc network s using machine learning-based blockchain routing." Concurrency and Computation: Practice and Experience 34, no. 17 (2022): e6993.

[10] Chakole, Megha, and S. S. Dorle. "Design of Advanced Encryption Standard Algorithm." In 2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA, pp. 1-5. IEEE, 2022.

[11] Patankar, Prerna, Sanjay Dorle, Nikhil Wyawahare, and Laxman P. Thakre. "Comparative Study on Design Of AI-Based Communication Protocol For VANET." In 2022 IEEE 4th International Conference on Cybernetics, Cognition and Machine Learning Applications (ICCCMLA), pp. 451-455. IEEE, 2022.

[12] Shinde, Rajeshwari, Asim Nilose, and Pankaj Chandankhede. "Design and Development of Geofencing Based Attendance System for Mobile Application." In 2022 10th International Conference on Emerging Trends in Engineering and Technology-Signal and Information Processing (ICETET-SIP-22), pp. 1-6. IEEE, 2022.

[13] Nandanwar, Yogesh N., Vednath P. Kalbande, Man Mohan, Kishor Rambhad, and Pramod V. Walke. "An approach toward higher electrical conversion efficiency of solar photovoltaic module using phase change materials." Energy Storage 4, no. 6 (2022): e379.

[14] Albishry, Nabeel, Rayed AlGhamdi, Abdulmohsen Almalawi, Asif Irshad Khan, and Pravin R. Kshirsagar. "An attribute extraction for automated malware attack classification and detection using soft computing techniques." Computational Intelligence and Neuroscience 2022 (2022).

[15] Gortázar, Francisco, and Micael Gallego. "A simple path towards testing cloud applications." In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pp. 28-29. IEEE, 2018.