MJI
Media Jurnal Informatika
Vol. 17, No. 2, December 2025
p-issn : 2088-2114, e-issn : 2477-2542

AI-Based Testing Using NLP Algorithm On Eggsperts Website Functionality Using Boundary Value Analysis Technique

Zolla Perdana Putra Harahap [a,1], Akhfa Bagas Alfarizi [a,2], Rio Ferddinansya[a,3], Adrian Fardan Andi[a,4], Muhammad Nasir[a,5*], Sofiyanti Indriasari[a,6]

[a] Software Engineering Study Program, Vocational School of IPB University, Jl. Kumbang No. 14, Bogor16128, Indonesia
[1] zollastarharahap@apps.ipb.ac.id; [2] akhfabagas@apps.ipb.ac.id; [3] rioferddinansya@apps.ipb.ac.id; [4]adriandi@apps.ipb.ac.id; [5*]m_nasir@apps.ipb.ac.id; [6] sofiyanti@apps.ipb.ac.id;
*corresponding author

ARTICLE INFORMATION;

Article history
Received 2025/09/15
Revised 2025/11/25
Accepted 2025/11/30

Keywords
Black Box Testing, Boundary Value Analysis, NLP-Based AI Testing, Artificial Intelligence Integration, Software Testing Life Cycle, Web Application

ABSTRACT

The poultry farming subsector plays a crucial role in national food security, yet remains constrained by manual recording and efficiency constraints. Digital transformation offers solutions such as the Eggspert website, designed to assist farmers in managing production and sales data quickly and in an integrated manner. This study aims to test the reliability and functionality of the Eggspert system and assess the effectiveness of integrating Artificial Intelligence (AI) into the software testing process. Unlike previous AI-assisted testing studies that primarily focus on generic software applications, this research emphasizes the application of NLP-based AI Testing within a domain-specific digital livestock management system, addressing the lack of empirical testing frameworks tailored to the poultry farming sector. This research is an experimental quantitative approach using Black Box Testing and Boundary Value Analysis (BVA) techniques, combined with Natural Language Processing (NLP)-based AI Testing. The process follows the Software Testing Life Cycle (STLC) stages to ensure systematic and measurable testing. Of the 42 test cases executed, 33 passed and 9 failed, resulting in a success rate of 78.57%. Each test case was executed repeatedly under consistent test conditions to ensure functional stability, with failures indicating specific validation weaknesses rather than random system behavior. Most system functions met specifications, although minor deficiencies remained in text validation and zero pricing. The integration of AI Testing has been shown to improve error detection efficiency. The combination of BVA and AI Testing effectively verified the functionality of the Eggspert system, increased the reliability and efficiency of the testing process, and has the potential to serve as a basis for developing an AI-based testing system in the digital livestock sector.

1. Introduction

The livestock industry has a strategic role in supporting national food security, especially in fulfilling the community's need for animal protein [1]. One of the fastest-growing subsectors is poultry farming, which encompasses both broiler and layer chicken production. However, numerous operational challenges remain, particularly for small and medium-sized farmers who rely on manual record-keeping for their production processes. Activities such as recording chicken stock, egg counts, sales revenue, and financial data are often conducted without an integrated digital system [2]. As a result, data is often inaccurate, difficult to track, and potentially leads to errors in decision-making that impact business efficiency and profitability.

In the development of digital transformation in animal husbandry, new IT-based technologies are present to improve animal health, production and welfare simultaneously [3]**.** Research from [4] emphasizes that technology-based devices, such as mobile phones, can facilitate cost-effective and efficient communication, even in areas with limited infrastructure. One of these is website Eggspert, a system based on web designed to help farmers record and manage livestock data systematically, quickly, and accurately. Through Eggspert, farmers can access important information such as chicken stock, egg production, and daily financial reports centrally and efficiently real-time. The existence of such a system is expected to improve operational efficiency, reduce recording errors, and strengthen data-driven decision-making.

Website itself is a collection of pages that are connected to each other in a network and present information in various formats, including text, images, videos and animations [5]. In the context of information systems development, website including software category (software) which operates through a web server [5]. Therefore, as with software in general, website also need to go through the testing stage (testing) to ensure quality, reliability, and suitability of functionality to end user needs.

According to [6] Software testing is a planned and systematic process to ensure that a system functions according to specifications and produces correct output. This stage plays a vital role in the software development cycle (Software Development Life Cycle/SDLC) because it is able to detect errors before the product is launched. In fact, according to [7], indicating a significant proportion of testing effort, underscores the importance of the testing process in software quality.

The testing process on the Eggsperts website is crucial because the system directly impacts the economic activities of farmers, who rely heavily on data accuracy and stable system functionality. Even the slightest error, such as an inconsistency in stock levels or prices, can result in significant financial losses. [8] in the Journal of Information and Computer Technology states that system testing is necessary to ensure the reliability of web-based application functions, especially in the context of the digital economy to minimize the potential for errors that impact user transactions. In addition, research by [9] In the Islamic Information Systems and Technology Bulletin, it emphasizes the importance of boundary value analysis as the main technique in detecting system errors related to data input that exceeds the threshold value.

One of the right approaches to test this kind of system is Black Box Testing, because it focuses on the output results from the input without needing to see the internal structure of the program code [10]. This method tests the system based on functional specifications, making it suitable for validating system behavior from a user perspective. Among the various techniques in Black Box Testing, Boundary Value Analysis (BVA) is the most relevant method. This technique focuses on testing boundary values, such as minimum and maximum values, and the values around those boundaries to identify errors that might not be detected through conventional testing [11].

According to [12], Boundary Value Analysis techniques are effective in detecting system behavior in the boundary area, because most software errors tend to appear when the system receives input that approaches extreme values. In line with that, research [12] shows that applying a combination of Boundary Value Analysis and Equivalence Partitioning can increase test case coverage and produce more comprehensive testing [2]. Study [13] also confirms that the use of this technique in sales information systems is able to detect input errors more effectively than without the limit value approach.

Technological advancements are driving the integration of Artificial Intelligence (AI) into software testing to improve efficiency and accuracy through automated error pattern identification and historical data analysis. This approach can accelerate testing cycles while reducing reliance on human intervention.

According to [14] The Boundary Value Analysis (BVA) method is an important technique in software testing because it can detect errors based on input boundary values, which are often the main source of system inaccuracy, [12] shows that the application of BVA and Equivalence Partitioning in automated web application testing can improve the effectiveness of the testing process, especially when combined with an AI-based approach. Furthermore, [15] explained that the use of Large Language Models (LLM) in boundary value testing automation demonstrates the immense potential of AI to improve the effectiveness and adaptability of software testing. In the context of black box testing, AI integration also plays a crucial role in automating the verification of key features of applications like Eggspert, with its ability to detect anomalies and error patterns that are difficult to identify manually.

2. Method

2.1 Types and Approaches of Research

This research is classified as an experimental research with a quantitative approach, as it focuses on testing system functionality through numerically measurable results. This approach was chosen to obtain objective findings by applying structured software testing methods. In line with the study [16] In the Journal of Technology and Information Systems, a quantitative approach was used to empirically measure system performance through a series of replicable tests. This research process followed the Software Testing Life Cycle (STLC) framework, which describes the systematic stages from requirements analysis to test closure as a guide for comprehensive software testing. This is also in line with the findings of [17] in the International Journal of Software Engineering and Its Applications, which emphasizes that implementing STLC helps ensure consistent, targeted testing and effectively detects system errors or anomalies. With this approach, researchers can identify system weaknesses in a measurable manner and present quantitatively valid results.

2.2 Object and Scope of Research

The object of this research is website Eggspert, a web-based system that serves as a platform for managing poultry product inventory and sales, focuses on three core functionalities: Egg Warehouse, Egg Sales, and Chicken Sales. These three features play a crucial role in supporting the system's core business processes, as each is responsible for managing crucial and interrelated data.

The Egg Warehouse feature is used to record and monitor egg stock availability in the warehouse. Testing of this feature is essential to ensure data accuracy and reflect actual conditions. The Egg Sales feature records egg sales transactions, including quantity, price, and product quality. Testing is conducted to ensure that transaction data is recorded according to business logic. Meanwhile, the Chicken Sales feature is used to record chicken sales transactions, both per chicken and per kilogram. Testing is conducted to verify input validation for quantity and price to ensure accurate sales results are recorded.

The selection of these three features is based on their level of urgency regarding the stability and accuracy of system data, because all of Eggspert's operational activities depend on valid sales and stock data.

2.3 Data Collection Techniques

The data collection technique in this study was conducted through a documentation study of the Software Requirement Specification (SRS) document and system test results on the Eggspert website. Each functional requirement defined in the SRS was mapped to corresponding test cases using a requirement-to-test-case traceability approach. This mapping ensured that every major system function tested had a clear reference to its expected behavior as specified in the SRS.

A mismatch between requirements and implementation was identified when the actual system output did not conform to the expected output defined in the SRS for a given test case, and was recorded as a Fail result. Conversely, conformity was recorded as Pass. This approach was chosen because it provides factual data regarding the alignment between software requirements and implementation results for each major functionality tested.. According to [3] documentation studies are an effective method in software testing research because they allow researchers to obtain concrete, verifiable evidence from well-documented system test results. Therefore, all test results in this study were recorded in detail to support the analysis and validation of the findings..

2.4 Tools and Materials Used

The research instruments consist of hardware in the form of a laptop or PC, and software in the form of website Eggspert is the primary test object. In addition, the software requirements specification (SRS) document is used as a reference for developing test cases and verifying test results. Additional software such as browser web, text editor, as well as test worksheets (spreadsheet) used to record, group, and analyze test results. All of these instruments support the efficient and systematic implementation of research.

2.5 Research Procedures or Stages

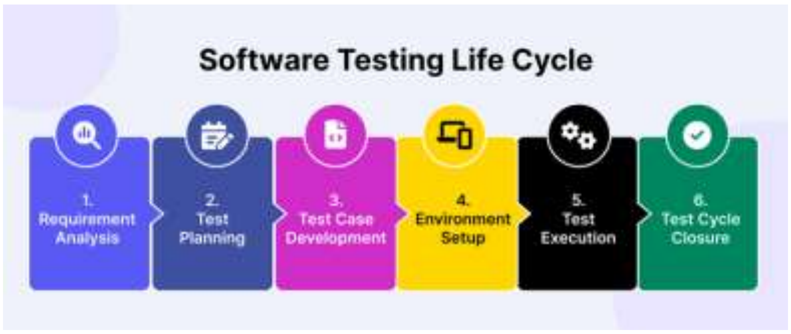


Figure 1. Software Testing Life Cycle

The research procedure was carried out with reference to the stages Software Testing Life Cycle (STLC), which consists of the following steps:

1) Requirement Analysis – Identify system requirements and determine the main functionality to be tested based on the SRS document.
2) Test Planning – Determine the testing strategy, resources, and techniques used, including method selection black box testing with technique Boundary Value Analysis (BVA).
3) Test Case Development – Create test cases based on the lower bound, upper bound, and values around those bounds. Test cases include valid, invalid, and typical inputs.
4) Environment Setup – Set up a test environment using specified hardware and software.
5) Test Execution– Run tests against each test case and record the actual results obtained from the system.
6) Test Closure – Evaluate test results, prepare final reports, and draw conclusions regarding the level of validity and reliability of the system.

Testing using the method black box testing with techniqueBoundary Value Analysis (BVA). This method focuses on testing software functionality based on input and output without paying attention to the internal structure of the program code [18]. BVA is used to test the lower limit value, upper limit value, and values around these limits in order to detect errors that often appear in the input limit area [19].

2.6 System Algorithm

The AI-based testing algorithm used in this study is designed to automate the functionality testing process of the Eggspert website using the Boundary Value Analysis (BVA) technique. The algorithm integrates Selenium automation with a rule-based Natural Language Processing (NLP) approach, where keyword-based pattern matching is applied for automatic classification of system output messages. The logical steps of the testing process are described as follows:

1) Initialize System Environment
   The system initializes the Selenium driver and opens the target URL (https://eggsperts.my.id/login).
2) Login Process
   The AI model inputs predefined credentials and verifies successful login by checking the page redirection status to the main dashboard.
3) Feature Selection
   The algorithm selects one of the three main modules to be tested: Egg Warehouse, Egg Sales, or Chicken Sales.
4) Test Case Generation
   Based on the Boundary Value Analysis (BVA) technique, the system automatically generates test cases for each input field, covering:
   - Lower boundary (invalid)
   - Minimum valid boundary
   - Typical/nominal values (equivalence partition)
   - Maximum valid boundary
   - Upper boundary (invalid)
5) Automated Test Execution
   For each test case, the system automatically fills in input values, submits the form, and retrieves the output message displayed on the website.

6) NLP-Based Output Classification
   Using a rule-based Natural Language Processing (NLP) approach, the output message is analyzed to determine whether the test result indicates “valid” or “invalid” system behavior. The classification is performed through keyword-based matching rules that map specific system response patterns to predefined labels. This approach is intentionally selected due to its interpretability, low computational complexity, and suitability for structured system-generated messages commonly found in web applications. The classification is performed based on keyword detection, for example:
   - If the message contains “successfully added” → labeled as Valid
   - If the message contains “invalid” or “error” → labeled as Invalid
7) Result Comparison and Logging
   The system compares the actual output with the expected output for each test case, and records the result as Pass or Fail into the test report (Excel format).
8) Result Summary and Closure
   After all test cases are executed, the algorithm calculates the success rate and summarizes the system’s performance in handling boundary input conditions.

The algorithm is shown schematically in Figure 2. Flowchart of Algorithm

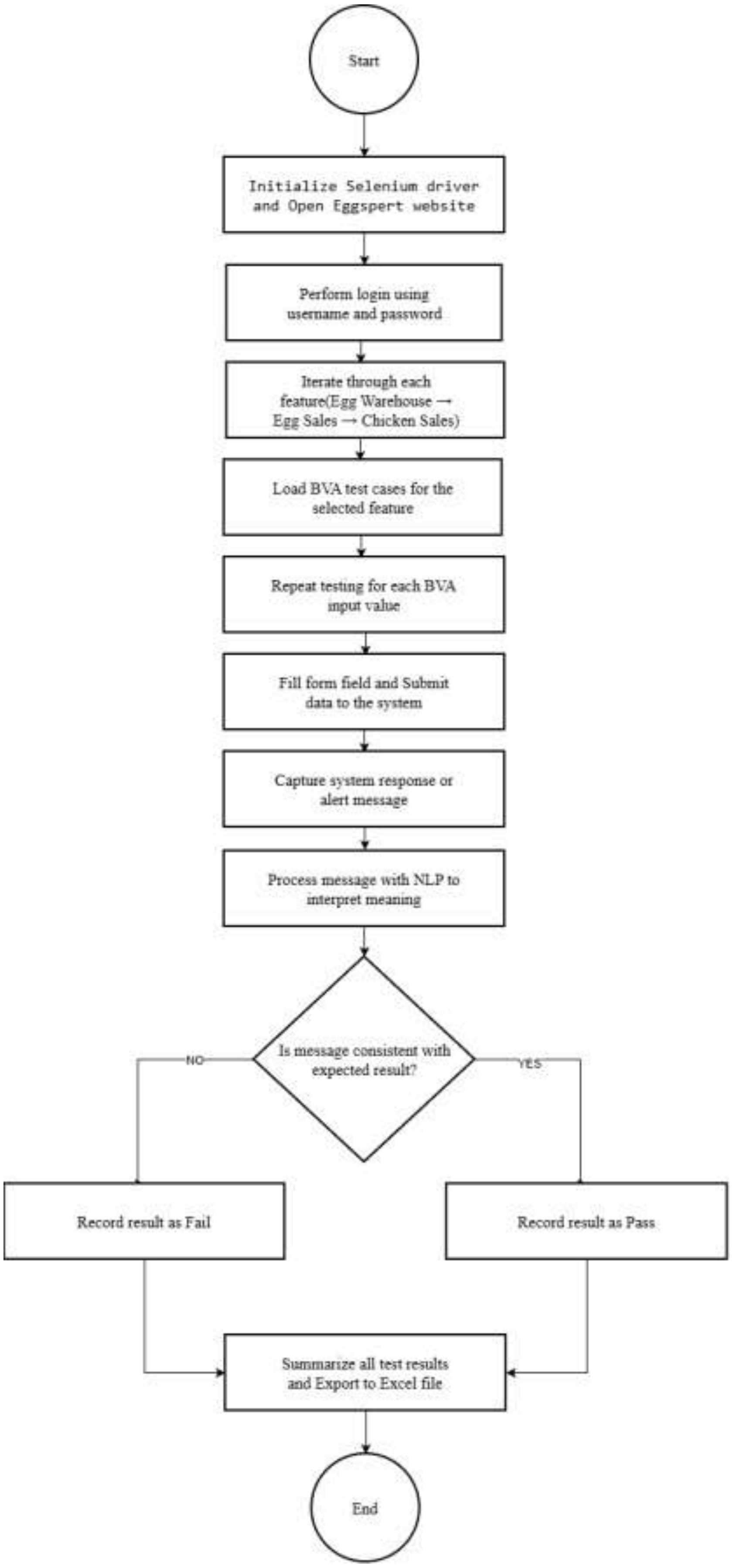

Start
Initialize Selenium driver and Open Eggspert website
Perform login using username and password
Iterate through each feature(Egg Warehouse → Egg Sales → Chicken Sales)
Load BVA test cases for the selected feature
Repeat testing for each BVA input value
Fill form field and Submit data to the system
Capture system response or alert message
Process message with NLP to interpret meaning
Is message consistent with expected result?
NO
YES
Record result as Fail
Record result as Pass
Summarize all test results and Export to Excel file
End

Figure 2. Flowchart of Algorithm

2.7 Data Analysis Techniques

Test result data is analyzed descriptively quantitatively, by comparing the actual system results against the expected results for each test case. If the actual results match the expected results, then the test case is declared successful (pass), whereas if they do not match, then it is declared failed (fail) [12]. This analysis aims to assess the extent to which the system functions correctly under the tested input boundary conditions, as well as to identify areas that require improvement. Through this approach, the validity of the system can be measured objectively, and the effectiveness of the implementation Boundary Value Analysis (BVA) can be evaluated systematically.

3. Results and Discussion

3.1 Presentation of Research Results

The Eggspert website testing process is carried out following the Software Testing Life Cycle (STLC) stages which include requirements analysis, test planning, test case development, environment setup, test execution, and test closure.

In the requirements analysis phase, testing needs were identified based on system specifications to determine the functional areas to be tested: the Egg Warehouse, Egg Sales, and Chicken Sales modules. From this analysis, it was determined that the testing focus would be on validating the boundary values of numeric and textual inputs to ensure the system could handle extreme values appropriately.

The test planning phase produces a test design using the Black Box Testing method with Boundary Value Analysis (BVA) techniques. This planning includes determining test criteria, test scenarios, and input limits to be tested, including lower limits, normal values, and upper limits.

Next, in the test case development phase, 42 test cases were created, representing a variety of valid and invalid values. Each case was designed to verify the system's behavior against input in critical areas, such as zero values, maximum characters, and 32-bit integer limits. The environment setup phase involves configuring a testing environment using a laptop with a stable internet connection, a web browser, and access to the internal test version of the Eggspert system. Once the environment is ready, the test execution phase begins by executing all the developed test scenarios. The actual results of each test are compared against the expected results based on the system specifications.

Meanwhile, the test closure phase produces a final test report that includes a summary of the test results and an analysis of the system's success rate. From a total of 42 test cases conducted, 33 were successfully passed. Therefore, the percentage of success can be calculated using the following formula:

$$\text{Success Rate}=(33 / 42)\times 100 = 78.57\%$$

This indicates that approximately 78.57% of the system's functionalities operate in accordance with the specified requirements. This value reflects a fairly good level of system reliability, although minor issues remain in the validation of character limits and zero-value inputs that require further improvement.

3.2 Meeting analysis

Analysis of the test results shows that the implementation of Boundary Value Analysis (BVA) and the Software Testing Life Cycle (STLC) stages are generally effective in detecting errors in the input boundary area. The requirements analysis and test case development stages successfully designed test scenarios that cover all variations of extreme values, both numeric and textual, ensuring that every critical boundary condition was evaluated. However, during the test execution stage, it was found that the system did not completely reject input with a value of zero and text length exceeding the maximum limit. This indicates that the front-end validation is functioning well, but the back-end validation still needs to be strengthened to prevent the storage of invalid data.

Similar weaknesses also emerged in the closure test results, where test logs showed failures at upper input boundary conditions. Nevertheless, the overall test results demonstrated a fairly good level of system reliability, with a success rate of 78.57%. This finding aligns with research [12] and [13], which highlight that the Boundary Value Analysis (BVA) method is effective in detecting input-related errors at extreme values. However, in contrast to the study by [20], which classifies systems with success rates above 90% as fit for use, the obtained result suggests that the tested system is adequately functional but still requires minor improvements, particularly in handling upper-bound input validations.

In addition, the integration of AI-based analysis, particularly through Natural Language Processing (NLP) in evaluating output messages, enhanced the objectivity and speed of result interpretation. This automated interpretation minimizes human bias and allows a more consistent mapping between expected and actual outputs. Rather than implementing adaptive or learning-based mechanisms, the NLP component applies predefined and iteratively refined keyword rules derived from observed test outcomes to improve the consistency of anomaly detection in system messages. The synergy between BVA and NLP not only simplifies the validation process but also opens opportunities for future adaptive testing frameworks that can automatically adjust test cases based on historical error trends.

Furthermore, the results indicate that applying STLC systematically ensures that each testing stage from planning, design, and execution to reporting is traceable and measurable. This structured approach allows early detection of potential bottlenecks in validation logic and supports continuous improvement of the system. It also emphasizes the importance of combining manual test logic (like BVA) with intelligent automation (AI) to achieve both depth and scalability in testing.

Therefore, this meeting analysis concludes that the AI-assisted BVA testing framework not only validates functional correctness but also contributes to improving the robustness of Eggspert's data management system. Strengthening the back-end validation layer and refining message interpretation algorithms in future iterations will further enhance data reliability, reduce potential user input errors, and ensure better decision-making accuracy for poultry farm management.

3.4 Test Results

- **Egg Warehouse Functionality Testing**

In the Egg Warehouse feature**,** testing is focused on the fieldNumber And Number of Eggs.The results show that the system can correctly reject blank input and values below the minimum threshold, and accept valid input from the minimum to the maximum threshold. However, the system still fails to reject text input exceeding the maximum character limit, indicating that the upper-bound validation mechanism is not yet fully functional.

In order for the testing process to be carried out automatically, an AI-based model is used.Natural Language Processing(NLP) to generate automated test code using the Python programming language and the Selenium library. The following is a screenshot (Figure 3. Egg Warehouse Form) of the Add Egg Warehouse form and the algorithm code using the Selenium library.

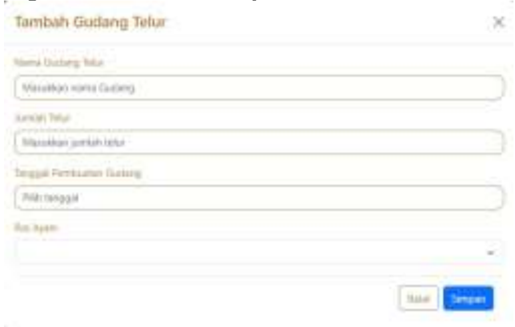
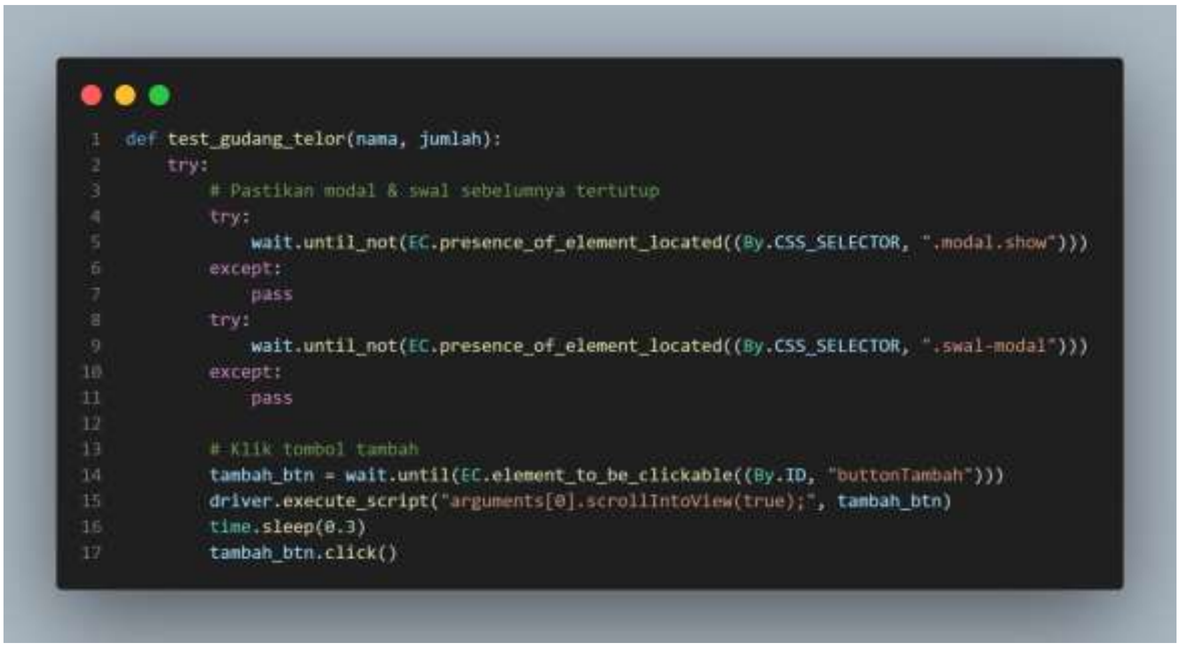


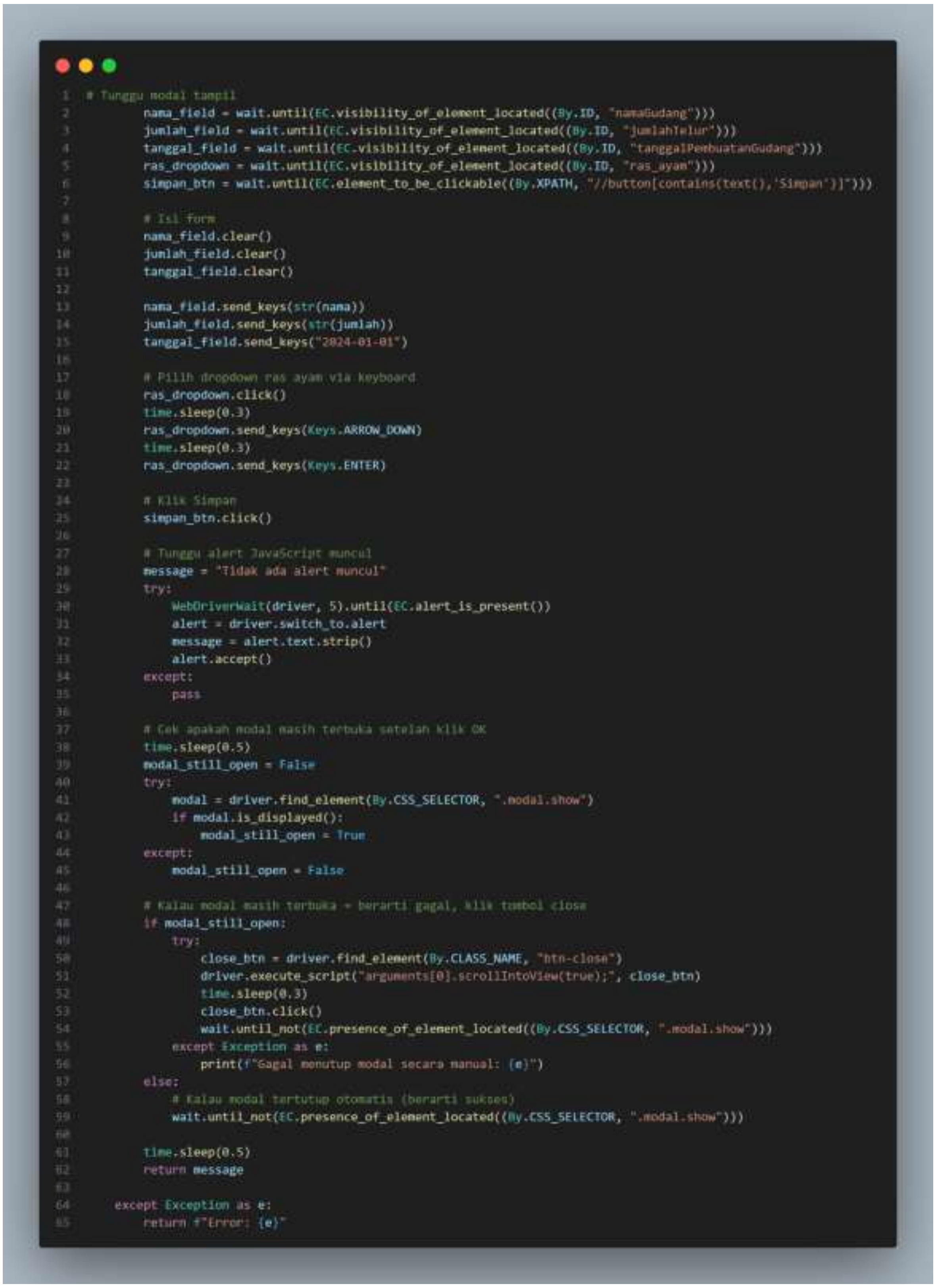
Figure 3. Egg Warehouse Form



```
def test_gudang_telor(nama, jumlah):
    try:
        # Pastikan modal & swal sebelumnya tertutup
        try:
            wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".modal.show")))
        except:
            pass
        try:
            wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".swal-modal")))
        except:
            pass

        # Klik tombol tambah
        tambah_btn = wait.until(EC.element_to_be_clickable((By.ID, "buttonTambah")))
        driver.execute_script("arguments[0].scrollIntoView(true);", tambah_btn)
        time.sleep(0.3)
        tambah_btn.click()
```

Figure 4. Code Snap AI Testing Egg Warehouse Functionality(1)



```python
# Tunggu modal tampil
        nama_field = wait.until(EC.visibility_of_element_located((By.ID, "namaGudang")))
        jumlah_field = wait.until(EC.visibility_of_element_located((By.ID, "jumlahTelur")))
        tanggal_field = wait.until(EC.visibility_of_element_located((By.ID, "tanggalPembuatanGudang")))
        ras_dropdown = wait.until(EC.visibility_of_element_located((By.ID, "ras_ayam")))
        simpan_btn = wait.until(EC.element_to_be_clickable((By.XPATH, "//button[contains(text(),'Simpan')]")))

        # Isi form
        nama_field.clear()
        jumlah_field.clear()
        tanggal_field.clear()

        nama_field.send_keys(str(nama))
        jumlah_field.send_keys(str(jumlah))
        tanggal_field.send_keys("2024-01-01")

        # Pilih dropdown ras ayam via keyboard
        ras_dropdown.click()
        time.sleep(0.3)
        ras_dropdown.send_keys(Keys.ARROW_DOWN)
        time.sleep(0.3)
        ras_dropdown.send_keys(Keys.ENTER)

        # Klik Simpan
        simpan_btn.click()

        # Tunggu alert JavaScript muncul
        message = "Tidak ada alert muncul"
        try:
            WebDriverWait(driver, 5).until(EC.alert_is_present())
            alert = driver.switch_to.alert
            message = alert.text.strip()
            alert.accept()
        except:
            pass

        # Cek apakah modal masih terbuka setelah klik OK
        time.sleep(0.5)
        modal_still_open = False
        try:
            modal = driver.find_element(By.CSS_SELECTOR, ".modal.show")
            if modal.is_displayed():
                modal_still_open = True
        except:
            modal_still_open = False

        # Kalau modal masih terbuka = berarti gagal, klik tombol close
        if modal_still_open:
            try:
                close_btn = driver.find_element(By.CLASS_NAME, "btn-close")
                driver.execute_script("arguments[0].scrollIntoView(true);", close_btn)
                time.sleep(0.3)
                close_btn.click()
                wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".modal.show")))
            except Exception as e:
                print(f"Gagal menutup modal secara manual: {e}")
        else:
            # Kalau modal tertutup otomatis (berarti sukses)
            wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".modal.show")))

        time.sleep(0.5)
        return message

    except Exception as e:
        return f"Error: {e}"
```

Figure 5. Code Snap AI Testing Egg Warehouse Functionality (2)

Table 1. BVA Test Results on the Egg Warehouse Feature

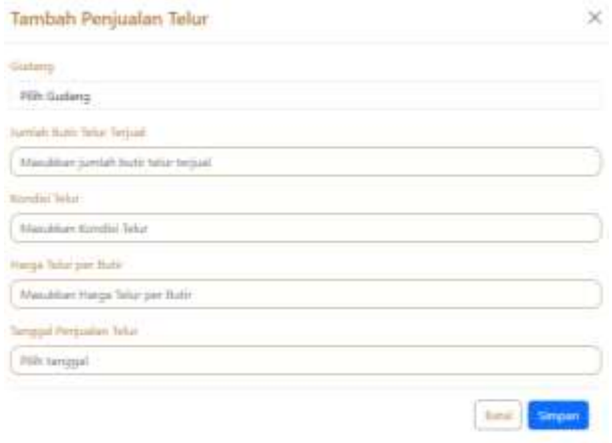
No	*Input*	*Input Value*	*Category BVA*	*Expected Output*	*Actual Output*	*Status (Pass/ Fail)*
1	Number	Empty Input	BVA Invalid (Below)	Valid/Invalid	Something went wrong. Please try again.	Fail
		A	BVA Valid (minimum)	Valid/Invalid	Warehouse Data successfully added!	Pass
		Cage A	EP/BVA (valid)	Valid/Invalid	Warehouse Data successfully added!	Pass
		(Input with 254 characters	Valid BVA (Below)	Valid/Invalid	Warehouse Data successfully added!	Pass
		(Input with 255 characters)	BVA Valid (Max)	Valid/Invalid	Warehouse Data successfully added!	Pass
		(Input with ≥ 256)	BVA Invalid (Atas)	Valid/Invalid	Something went wrong. Please try again.	Fail
2	Number of Eggs	0	BVA Invalid (Below)	Valid/Invalid	Warehouse Data successfully added!	Pass
		1	BVA Valid (minimum)	Valid/Invalid	Warehouse Data successfully added!	Pass
		2	EP/BVA (valid)	Valid/Invalid	Warehouse Data successfully added!	Pass
		Input with the number of integers before the maximum on 32-bit int (2147483646)	Valid BVA (Below)	Valid/Invalid	Warehouse Data successfully added!	Pass
		Input with the maximum integer number in 32-bit int (2147483647)	BVA Valid (Max)	Valid/Invalid	Warehouse Data successfully added!	Pass
		Input with an integer number exceeding the maximum on a 32-bit int (2147483648)	BVA Invalid (Atas)	Valid/Invalid	Warehouse Data successfully added!	Pass

Based on the automated testing results generated by the AI model (Table 1. BVA Test Results on the Egg Warehouse Feature), the system is able to correctly reject blank input and values below the minimum threshold. However, there are several anomalies in the Number of Eggs field, where the system still accepts values above the maximum threshold (2147483648) that should be rejected. Further analysis indicates that this issue occurs due to insufficient back-end validation, as the front-end input constraints are bypassed during automated submission, allowing oversized numeric values to be processed by the server. This finding suggests that upper-bound validation for numeric data types has not been strictly enforced at the server level. Meanwhile, the Warehouse Name field shows more consistent validation results, where the system only fails at extreme upper thresholds (very long text input), indicating partial reliance on front-end validation mechanisms. Therefore, improvements are needed in both back-end numeric validation and front-end character length constraints to better maintain data integrity.

- **Egg Sales Functionality Testing**

In the Egg Sales feature, testing focused on the Number of Eggs Sold, Egg Condition, and Price per Egg fields. Results showed that the system correctly rejected blank input and values below the minimum threshold, and accepted valid input ranging from the minimum to the maximum threshold. However, the system still failed to reject input with a price value of zero and egg condition text exceeding 260 characters, indicating that the upper threshold validation mechanism was not fully functional. In particular, the acceptance of a zero price value suggests a potential inconsistency between technical validation rules and underlying business logic, as the Software Requirement Specification (SRS) does not explicitly define whether a zero-price transaction is permitted or should be treated as invalid. This finding highlights the need for clearer business rule definition and alignment between requirement specifications and validation implementation.

To ensure the testing process can be carried out automatically, an AI model based on Natural Language Processing (NLP) is used which generates automatic test code using the Python programming language with the Selenium library.



Tambah Penjualan Telur

Gudang
Pilih Gudang

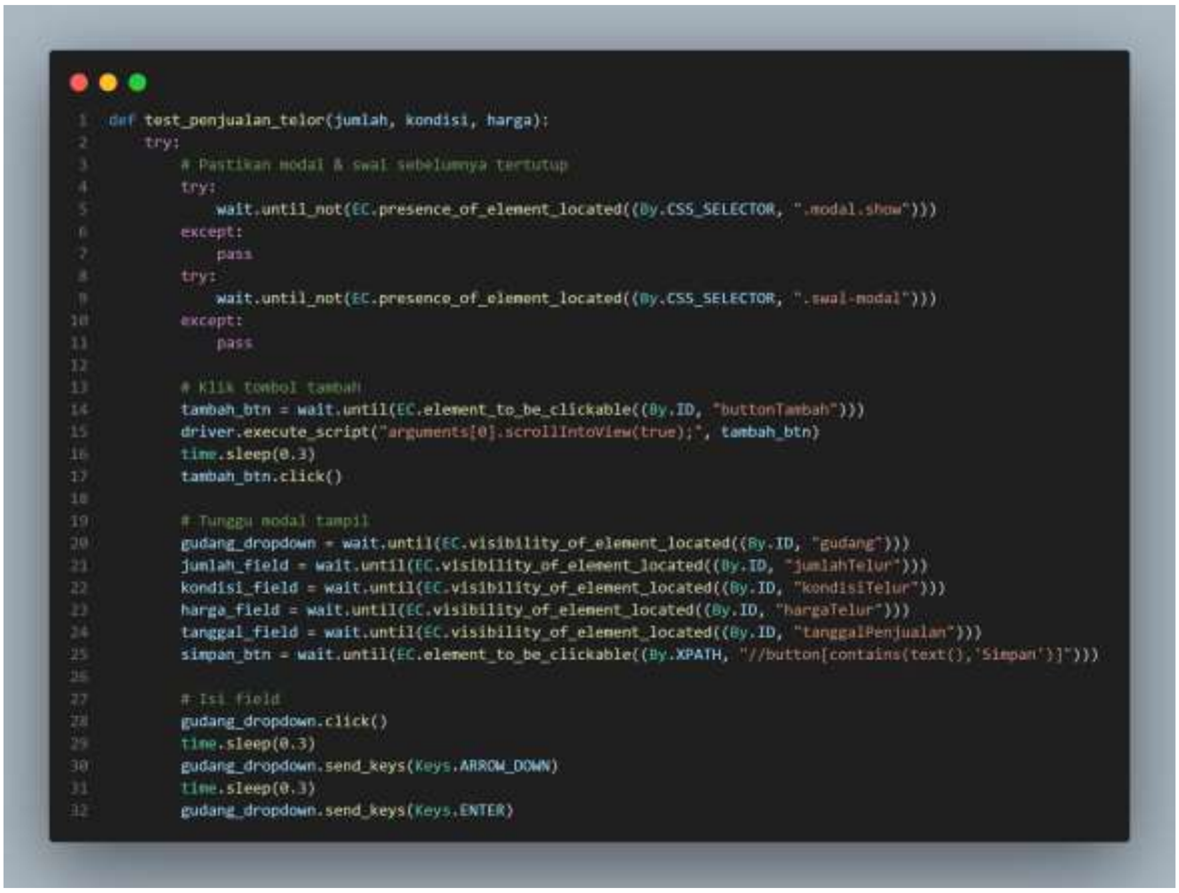
Jumlah Butir Telur Terjual
Masukkan jumlah butir telur terjual

Kondisi Telur
Masukkan Kondisi Telur

Harga Telur per Butir
Masukkan Harga Telur per Butir

Tanggal Penjualan Telur
Pilih tanggal

Figure 6. Egg Sales Form



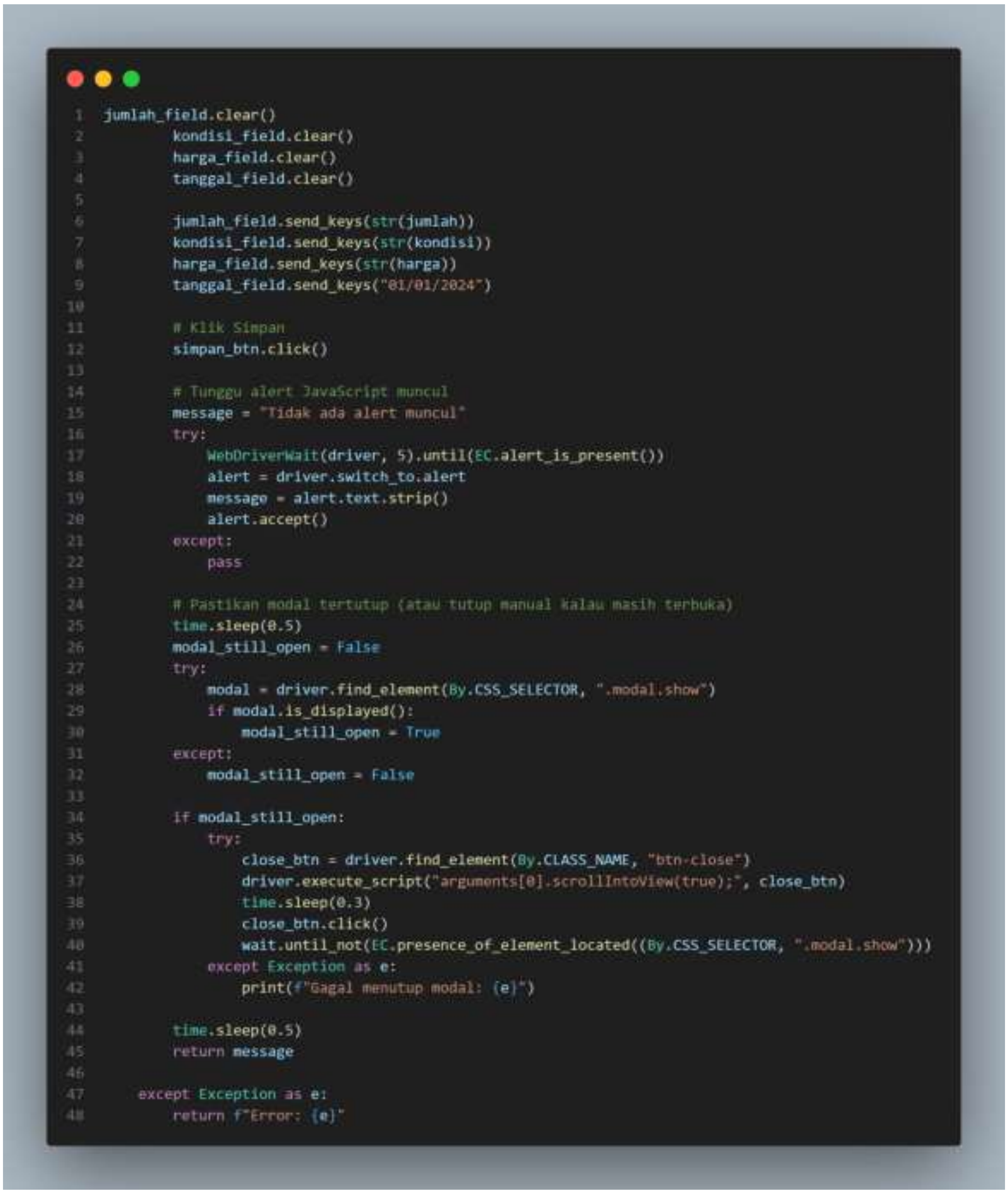
```python
def test_penjualan_telor(jumlah, kondisi, harga):
    try:
        # Pastikan modal & swal sebelumnya tertutup
        try:
            wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".modal.show")))
        except:
            pass
        try:
            wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".swal-modal")))
        except:
            pass

        # Klik tombol tambah
        tambah_btn = wait.until(EC.element_to_be_clickable((By.ID, "buttonTambah")))
        driver.execute_script("arguments[0].scrollIntoView(true);", tambah_btn)
        time.sleep(0.3)
        tambah_btn.click()

        # Tunggu modal tampil
        gudang_dropdown = wait.until(EC.visibility_of_element_located((By.ID, "gudang")))
        jumlah_field = wait.until(EC.visibility_of_element_located((By.ID, "jumlahTelur")))
        kondisi_field = wait.until(EC.visibility_of_element_located((By.ID, "kondisiTelur")))
        harga_field = wait.until(EC.visibility_of_element_located((By.ID, "hargaTelur")))
        tanggal_field = wait.until(EC.visibility_of_element_located((By.ID, "tanggalPenjualan")))
        simpan_btn = wait.until(EC.element_to_be_clickable((By.XPATH, "//button[contains(text(),'Simpan')]")))

        # Isi field
        gudang_dropdown.click()
        time.sleep(0.3)
        gudang_dropdown.send_keys(Keys.ARROW_DOWN)
        time.sleep(0.3)
        gudang_dropdown.send_keys(Keys.ENTER)
```

Figure 7. Code Snap AI Testing Egg Sales Functionality (1)



```
jumlah_field.clear()
        kondisi_field.clear()
        harga_field.clear()
        tanggal_field.clear()

        jumlah_field.send_keys(str(jumlah))
        kondisi_field.send_keys(str(kondisi))
        harga_field.send_keys(str(harga))
        tanggal_field.send_keys("01/01/2024")

        # Klik Simpan
        simpan_btn.click()

        # Tunggu alert JavaScript muncul
        message = "Tidak ada alert muncul"
        try:
            WebDriverWait(driver, 5).until(EC.alert_is_present())
            alert = driver.switch_to.alert
            message = alert.text.strip()
            alert.accept()
        except:
            pass

        # Pastikan modal tertutup (atau tutup manual kalau masih terbuka)
        time.sleep(0.5)
        modal_still_open = False
        try:
            modal = driver.find_element(By.CSS_SELECTOR, ".modal.show")
            if modal.is_displayed():
                modal_still_open = True
        except:
            modal_still_open = False

        if modal_still_open:
            try:
                close_btn = driver.find_element(By.CLASS_NAME, "btn-close")
                driver.execute_script("arguments[0].scrollIntoView(true);", close_btn)
                time.sleep(0.3)
                close_btn.click()
                wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".modal.show")))
            except Exception as e:
                print(f"Gagal menutup modal: {e}")

        time.sleep(0.5)
        return message

    except Exception as e:
        return f"Error: {e}"
```

Figure 8. Code Snap AI Testing Egg Sales Functionality (2)

Table 2. BVA Test Results on Sales Features

No	*Input*	*Input Value*	*Category BVA*	*Expected Output*	*Actual Output*	*Status (Pass/ Fail)*
1	Number of Eggs Sold	0	BVA Invalid (Below)	Valid/Invalid	Something went wrong. Please try again.	Fail
		1	BVA Valid (minimum)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		2	EP/BVA (valid)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		Input with the number of integers before the maximum on 32-bit int (2147483646)	Valid BVA (Below)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		Input with the maximum integer number in 32-bit int (2147483647)	BVA Valid (Max)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		Input with an integer number exceeding the maximum on a 32-bit int (2147483648)	BVA Invalid (Atas)	Valid/Invalid	Something went wrong. Please try again.	Fail
2	Egg Condition	Empty input	BVA Invalid (Below)	Valid/Invalid	Something went wrong. Please try again.	Fail
		A	BVA Valid (minimum)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		Condition A	EP/BVA (valid)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		(Input with 258 characters)	Valid BVA (Below)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		(Input with 259 characters)	BVA Valid (Max)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		(Input with ≥ 260)	BVA Invalid (Atas)	Valid/Invalid	Something went wrong. Please try again.	Fail
3	Price	0	BVA Invalid (Below)	Valid/Invalid	Something went wrong. Please try again.	Fail
		1	BVA Valid (minimum)	Valid/Invalid	Egg Sales Data successfully added!	Pass

No	*Input*	*Input Value*	*Category BVA*	*Expected Output*	*Actual Output*	*Status (Pass/ Fail)*
		2	EP/BVA (valid)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		Input with the number of integers before the maximum on 32-bit int (2147483646)	Valid BVA (Below)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		Input with the maximum integer number in 32-bit int (2147483647)	BVA Valid (Max)	Valid/Invalid	Egg Sales Data successfully added!	Pass
		Input with an integer number exceeding the maximum on a 32-bit int (2147483648)	BVA Invalid (Atas)	Valid/Invalid	Something went wrong. Please try again.	Fail

- **Chicken Sales Functionality Testing**

Meanwhile, in the Chicken Sales feature, testing focused on two main fields: Number of Chickens Sold and Price per Chicken. The test results showed that the system was able to reject blank input and values below the minimum limit in the Number of Chickens Sold field, and correctly accepted valid input from the minimum to maximum limits. However, the system still showed anomalies in the Chicken Price per Chicken field, where input with a price value of zero was still accepted and stored in the database even though the system displayed the message "Something went wrong. Please try again." This indicates that the minimum price value validation mechanism has not been fully consistent in this module.

To automate the testing process, a Natural Language Processing (NLP) based AI model was used to generate automated test code using the Python programming language and the Selenium library. The AI model then generated the following code snippet (Figure 10. Code Snap AI Testing Chicken Sales Functionality (1)) to automatically perform functionality testing:

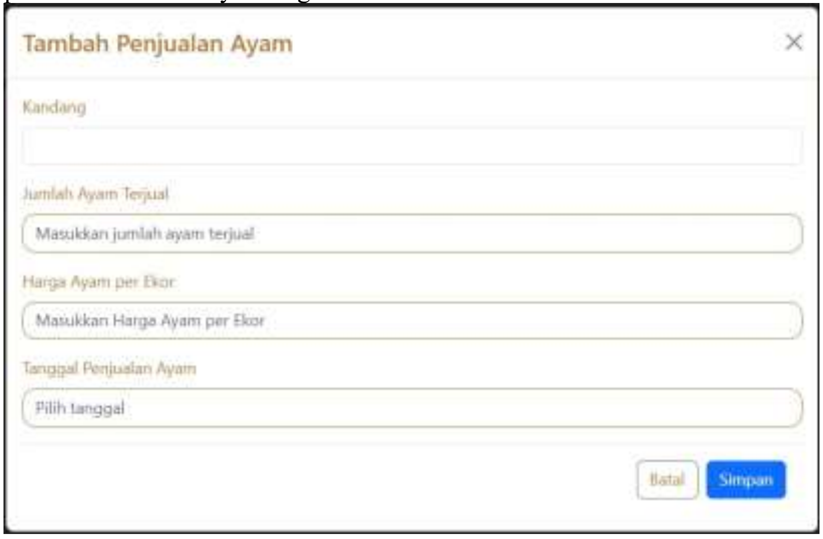
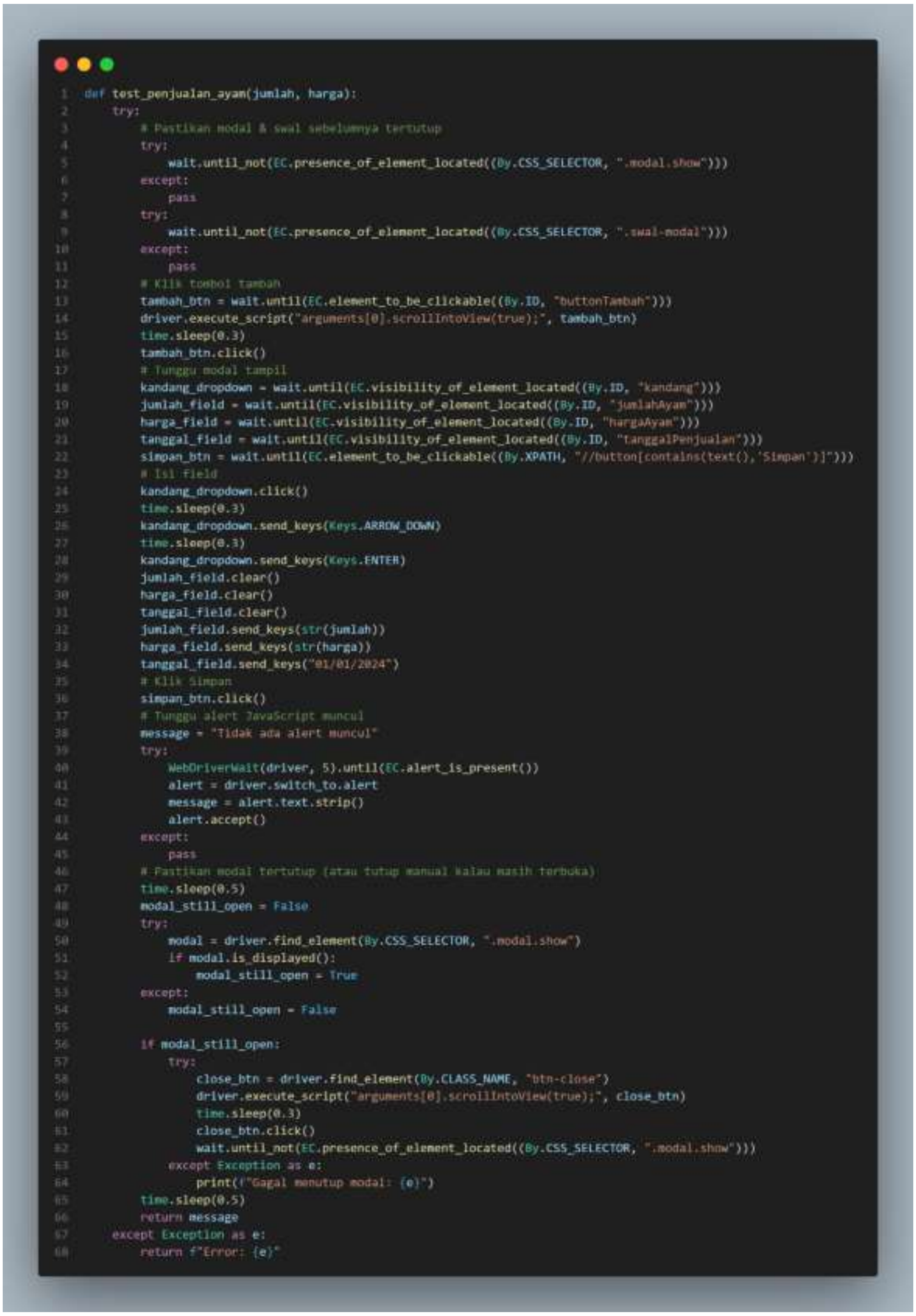


Figure 9. Chicken Sales Form



```python
def test_penjualan_ayam(jumlah, harga):
    try:
        # Pastikan modal & swal sebelumnya tertutup
        try:
            wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".modal.show")))
        except:
            pass
        try:
            wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".swal-modal")))
        except:
            pass
        # Klik tombol tambah
        tambah_btn = wait.until(EC.element_to_be_clickable((By.ID, "buttonTambah")))
        driver.execute_script("arguments[0].scrollIntoView(true);", tambah_btn)
        time.sleep(0.3)
        tambah_btn.click()
        # Tunggu modal tampil
        kandang_dropdown = wait.until(EC.visibility_of_element_located((By.ID, "kandang")))
        jumlah_field = wait.until(EC.visibility_of_element_located((By.ID, "jumlahAyam")))
        harga_field = wait.until(EC.visibility_of_element_located((By.ID, "hargaAyam")))
        tanggal_field = wait.until(EC.visibility_of_element_located((By.ID, "tanggalPenjualan")))
        simpan_btn = wait.until(EC.element_to_be_clickable((By.XPATH, "//button[contains(text(),'Simpan')]")))
        # Isi field
        kandang_dropdown.click()
        time.sleep(0.3)
        kandang_dropdown.send_keys(Keys.ARROW_DOWN)
        time.sleep(0.3)
        kandang_dropdown.send_keys(Keys.ENTER)
        jumlah_field.clear()
        harga_field.clear()
        tanggal_field.clear()
        jumlah_field.send_keys(str(jumlah))
        harga_field.send_keys(str(harga))
        tanggal_field.send_keys("01/01/2024")
        # Klik Simpan
        simpan_btn.click()
        # Tunggu alert JavaScript muncul
        message = "Tidak ada alert muncul"
        try:
            WebDriverWait(driver, 5).until(EC.alert_is_present())
            alert = driver.switch_to.alert
            message = alert.text.strip()
            alert.accept()
        except:
            pass
        # Pastikan modal tertutup (atau tutup manual kalau masih terbuka)
        time.sleep(0.5)
        modal_still_open = False
        try:
            modal = driver.find_element(By.CSS_SELECTOR, ".modal.show")
            if modal.is_displayed():
                modal_still_open = True
        except:
            modal_still_open = False

        if modal_still_open:
            try:
                close_btn = driver.find_element(By.CLASS_NAME, "btn-close")
                driver.execute_script("arguments[0].scrollIntoView(true);", close_btn)
                time.sleep(0.3)
                close_btn.click()
                wait.until_not(EC.presence_of_element_located((By.CSS_SELECTOR, ".modal.show")))
            except Exception as e:
                print(f"Gagal menutup modal: {e}")
        time.sleep(0.5)
        return message
    except Exception as e:
        return f"Error: {e}"
```

Figure 10. Code Snap AI Testing Chicken Sales Functionality (1)

Table 3. BVA Test Results on the Chicken Sales Feature

No	***Input***	***Input Value***	***Category BVA***	***Expected Output***	***Actual Output***	***Status (Pass/ Fail)***
1	Number of Chickens Sold	0	BVA Invalid (Below)	Valid/Invalid	Something went wrong. Please try again.	Pass
		1	BVA Valid (minimum)	Valid/Invalid	Chicken Sales Data successfully added!	Pass
		2	EP/BVA (valid)	Valid/Invalid	Chicken Sales Data successfully added!	Pass
		Input with the number of integers before the maximum on 32-bit int (2147483646)	Valid BVA (Below)	Valid/Invalid	Chicken Sales Data successfully added!	Pass
		Input with the maximum integer number in 32-bit int (2147483647)	BVA Valid (Max)	Valid/Invalid	Chicken Sales Data successfully added!	Pass
		Input with an integer number exceeding the maximum on a 32-bit int (2147483648)	BVA Invalid (Atas)	Valid/Invalid	Something went wrong. Please try again.	Pass
2	Chicken Price per Head	0	BVA Invalid (Below)	Valid/Invalid	Something went wrong. Please try again.	Fail
		1	BVA Valid (minimum)	Valid/Invalid	Chicken Sales Data successfully added!	Pass
		2	EP/BVA (valid)	Valid/Invalid	Chicken Sales Data successfully added!	Pass
		Input with the number of integers before the maximum on 32-bit int (2147483646)	Valid BVA (Below)	Valid/Invalid	Chicken Sales Data successfully added!	Pass
		Input with the maximum integer number in 32-bit int (2147483647)	BVA Valid (Max)	Valid/Invalid	Chicken Sales Data successfully added!	Pass

No	*Input*	*Input Value*	*Category BVA*	*Expected Output*	*Actual Output*	*Status (Pass/ Fail)*
		Input with an integer number exceeding the maximum on a 32-bit int (2147483648)	BVA Invalid (Atas)	Valid/Invalid	Something went wrong. Please try again.	Pass

Overall, the results of the three functionalities show that the websiteEggsperthas met most of the input limit testing criteria with a high success rate, although it still requires improvements in price validation and text character limits for the system to function optimally in all input scenarios.

4. Conclusion

This study aims to evaluate the reliability of the Eggspert website's functionality using the Black Box Testing method with Boundary Value Analysis (BVA) and Equivalence Partitioning techniques. Testing was conducted on three main modules: Egg Warehouse, Egg Sales, and Chicken Sales. These three modules verify the validation of numeric and textual input at the lower limit, normal value, and upper limit.

The results showed that most of the limit validation processes performed well, including accepting the maximum value of a 32-bit integer type and rejecting input exceeding that limit. Of the 42 test cases executed, 33 passed and 9 failed, resulting in a test success rate of 78.57%. This value indicates that the system is in good functional condition and is suitable for use, although some minor improvements are still needed.

The weaknesses identified include text length validation in Warehouse Name, which still accepts inputs exceeding 256 characters, Egg Condition validation that fails to reject inputs exceeding 260 characters, and pricing business rules that are not yet fully consistent in rejecting zero price values. Furthermore, the general nature of error messages reduces clarity for users. From a testing perspective, the use of AI-assisted automated classification reduced the need for manual inspection of test outputs, as all 42 test cases were automatically logged and categorized into valid and invalid results, thereby minimizing manual verification steps and improving testing efficiency.

These findings confirm that the Boundary Value Analysis technique is effective in identifying input errors at boundary conditions, and demonstrate the need for strengthened backend validation and alignment of business rules between modules. Implementing a Software Testing Life Cycle (STLC) based testing process has also been shown to help maintain traceability and systematic software testing stages.

For further development, it is recommended that validation on the backend side be tightened, error messages be clarified with specific context, and testing be expanded to non-functional aspects such as performance, system load, and data security. This expansion is particularly important considering real deployment constraints in agricultural environments, where internet connectivity, bandwidth stability, and access to computing resources may vary significantly across poultry farming locations. Addressing these constraints through non-functional testing will help ensure that the Eggspert system remains reliable and usable under real operational conditions.

Confession

The author would like to express his gratitude to the supervising lecturer and colleagues who provided constructive input and support during the research process.

Declaration

Author's contribution.
All authors contributed equally to the design, development, and testing of the Moodle plugin for Function Point analysis. All authors were responsible for the development, integration, and testing stages of the plugin. Furthermore, all authors participated in the manuscript writing, data analysis, and review process. All authors have approved the final manuscript submitted for publication.

Funding statement.
This research did not receive funding from any institution.

Conflict of interest.
The author declares that there is no conflict of interest in this research.

Additional information.
There is no additional information that can be provided for this study.

Data and Software Availability Statement

The data and test artifacts generated or analyzed in this study are available on reasonable request to the corresponding author eggsperts.my.id which is used as a research object is an internal academic prototype and is not published openly due to institutional policy.

Reference

[1] S. E. Putri, F. Majid Abdullah, R. Septiyaningsih, F. Aulia, and T. Puji Rahayu, "Nutrisi Seimbang untuk Unggas: Memahami Pentingnya Protein dan Serat," *J. Ilm. Ilmu-Ilmu Peternak.*, vol. 28, no. 1, pp. 1–11, May 2025, doi: 10.22437/jiiip.v28i1.35982.

[2] W. Wahyudi, A. I. Pradana, and H. Permatasari, "Implementasi Sistem Irigasi Otomatis Berbasis IoT untuk Pertanian Greenhouse," *J. Pendidik. Dan Teknol. Indones.*, vol. 5, no. 2, Feb. 2025, doi: 10.52436/1.jpti.656.

[3] H. A. Himu and A. Raihan, "Digital Transformation of Livestock Farming for Sustainable Development," *Int. J. Livest. Res.*, 2024 https://doi.org/10.3390/agriculture15090937.

[4] V. Rotondi, R. Kashyap, L. M. Pesando, S. Spinelli, and F. C. Billari, "Leveraging mobile phones to attain sustainable development," *Proc. Natl. Acad. Sci.*, vol. 117, no. 24, pp. 13413–13420, June 2020, doi: 10.1073/pnas.1909326117.

[5] S. H. Azhari, B. Huda, and F. Apriani, "TESTING DAN IMPLEMENTASI APLIKASI PERPUSTAKAAN DIGITAL BERBASIS WEB (STUDI KASUS SMK SEHATI KARAWANG)" Oct. 2024.

[6] M. Nasir *et al.*, "IMPLEMENTASI EQUIVALENCE PARTITIONING TESTING PADA FITUR BOOKING DAN JADWAL WEBSITE PRAKTIK GIGI MANDIRI drg. Susilawati (https://frontend-webklinik.vercel.app/)," vol. 9, no. 3, Apr. 2025.

[7] I. R. Dhaifullah, M. Muttanifudin H, A. Ananda Salsabila, and M. Ainul Yaqin, "Survei Teknik Pengujian Software," *J. Autom. Comput. Inf. Syst.*, vol. 2, no. 1, pp. 31–38, June 2022, doi: 10.47134/jacis.v2i1.42.

[8] I. G. N. A. Cahyadi Putra and A. T. A. P. Kusuma, "IMPLEMENTASI DAN PENGUJIAN SISTEM INFORMASI PENGADUAN INVENTARIS KELAS," *J. Teknol. Inf. Dan Komput.*, vol. 5, no. 1, Feb. 2019, doi: 10.36002/jutik.v5i1.635.

[9] H. N. Widiani *et al.*, "Pengujian Black Box Testing pada Website Segitiga Motor Menggunakan Teknik Boundary Value Analysis," *Bul. Sist. Inf. Dan Teknol. Islam*, vol. 5, no. 4, pp. 304–309, Dec. 2024, doi: 10.33096/busiti.v5i4.2482.

[10] Ana Nurfadilah and Ilham, "PENERAPAN SISTEM INFORMASI MANAJEMEN DI ERA DIGITAL : TANTANGAN DAN SOLUSI," *J. Cakrawala Inf.*, vol. 4, no. 2, pp. 68–87, Dec. 2024, doi: 10.54066/jci.v4i2.499.

[11] H. N. Widiani *et al.*, "Pengujian Black Box Testing pada Website Segitiga Motor Menggunakan Teknik Boundary Value Analysis," *Bul. Sist. Inf. Dan Teknol. Islam*, vol. 5, no. 4, pp. 304–309, Dec. 2024, doi: 10.33096/busiti.v5i4.2482.

[12] E. S. J. Atmadji, I. R. Sanjaya, and H. A. Putranto, "Pemanfaatan boundary value analysis dan equivalence partitioning pada automated testing aplikasi berbasis website," *Angkasa J. Ilm. Bid. Teknol.*, vol. 15, no. 1, p. 97, May 2023, doi: 10.28989/angkasa.v15i1.1645.

[13] S. Nevile, I. Andika, and R. Satya, "Analisis dan Pengujian Sistem Informasi Penjualan menggunakan Metode Boundary Value Analysis dan Metode Equivalence Partitioning," *DEVICE J. Inf. Syst. Comput. Sci. Inf. Technol.*, vol. 5, no. 1, pp. 170–187, June 2024, doi: 10.46576/device.v5i1.4534.

[14] E. Sianturi, "Boundary Value Analysis and Decision Table Testing Methods in Software Testing," *Int. J. Inf. Technol. Educ.*, vol. 1, no. 3, pp. 124–129, Aug. 2022, doi: 10.62711/ijite.v1i3.68.

[15] X. Guo, C. Li, and T. Tsuchiya, "Boundary Value Test Input Generation Using Prompt Engineering with LLMs: Fault Detection and Coverage Analysis," Jan. 24, 2025, *arXiv*: arXiv:2501.14465. doi: 10.48550/arXiv.2501.14465.

[16] D. F. Rachmawati, A. Handayanto, and R. E. Utami, "Efektivitas Media Pembelajaran Berbantu Website dengan Pendekatan Kontekstual Terhadap Kemampuan Berpikir Kreatif Siswa SMP," *Imajiner J. Mat. Dan Pendidik. Mat.*, vol. 2, no. 3, pp. 258–265, July 2020, doi: 10.26877/imajiner.v2i3.6121.

[17] K. Antonakoglou, M. Mahlouji, and T. Mahmoodi, "Bilateral Teleoperation Performance Model for Network Resource Management," *IEEE Access*, vol. 9, pp. 29106–29117, 2021, doi: 10.1109/ACCESS.2021.3059233.

[18] F. Dobslaw, F. G. de O. Neto, and R. Feldt, "Boundary Value Exploration for Software Analysis," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Oct. 2020, pp. 346–353. doi: 10.1109/ICSTW50294.2020.00062.

[19] F. Amalia, M. Kurniawan, and D. T. Setiawan, “Pengujian Blackbox pada Desain Antarmuka Sistem Informasi Traceability Rantai Pasok Apel,” *J. Teknol. Inf. Dan Ilmu Komput.*, vol. 8, no. 5, p. 853, Dec. 2021, doi: 10.25126/jtiik.2021823487.

[20] S. H. Azhari, B. Huda, and F. Apriani, “TESTING DAN IMPLEMENTASI APLIKASI PERPUSTAKAAN DIGITAL BERBASIS WEB (STUDI KASUS SMK SEHATI KARAWANG),” vol. 4, no. 3, pp. 368–379, July 2023, doi: https://doi.org/10.36312/jcm.v4i3.1779.