

IMPLEMENTASI OTOMATISASI PENGUJIAN FUNGSIONAL PADA PEMBELIAN BARANG DI APLIKASI *E-COMMERCE* MENGGUNAKAN PLAYWRIGHT DAN GITHUB ACTIONS DALAM *CONTINUOUS INTEGRATION*

Nicky Pascal Tambunan¹, Sufajar Butsianto²

^{1,2}Universitas Pelita Bangsa, Bekasi, Indonesia

Email: nicky.312210474@mhs.pelitabangsa.ac.id

Abstrak

Penelitian ini bertujuan mengimplementasikan otomatisasi pengujian fungsional menggunakan Playwright yang terintegrasi dengan GitHub Actions pada proses Continuous Integration serta mengetahui hasil penerapannya berdasarkan status pengujian, jumlah test case, waktu pelaksanaan, dan log pengujian. Metode pengumpulan data dilakukan melalui studi literatur, observasi, dan eksperimen. Pengujian dilakukan terhadap empat fitur aplikasi e-commerce, yaitu login, pencarian produk, keranjang belanja, dan checkout, dengan total sepuluh test case. Pengujian dilakukan secara manual sebagai data awal dan secara otomatis menggunakan Playwright melalui peramban Chromium, sedangkan GitHub Actions digunakan untuk menjalankan workflow pengujian ketika terjadi aktivitas push ke branch main atau pull request yang ditujukan ke branch main. Hasil penelitian menunjukkan bahwa seluruh sepuluh test case memperoleh status passed, sedangkan failed, flaky, dan skipped masing-masing sebanyak 0. Pengujian manual membutuhkan waktu 157 detik atau 2 menit 37 detik, sedangkan pengujian otomatis menggunakan Playwright membutuhkan waktu 15,0 detik, sehingga waktu pelaksanaan pengujian otomatis sekitar 90,4% lebih singkat pada skenario yang digunakan dalam penelitian. GitHub Actions berhasil menjalankan workflow dengan status success serta menghasilkan log dan artifact laporan pengujian. Berdasarkan hasil tersebut, otomatisasi pengujian fungsional menggunakan Playwright dan GitHub Actions berhasil diterapkan sebagai bagian dari proses Continuous Integration serta mampu menjalankan skenario pengujian secara lebih cepat, konsisten, dan dapat diulang.

Kata Kunci: *Continuous Integration, e-commerce, GitHub Actions, Pengujian Fungsional, Playwright.*

Abstract

This study aims to implement functional test automation using Playwright integrated with GitHub Actions in the Continuous Integration process and to evaluate its implementation based on test status, the number of test cases, execution time, and test logs. Data were collected through literature review, observation, and experimentation. Testing was conducted on four e-commerce application features, namely login, product search, shopping cart, and checkout, with a total of ten test cases. The tests were initially performed manually as baseline data and then automated using Playwright with the Chromium browser, while GitHub Actions was used to execute the testing workflow when a push to the main branch or a pull request targeting the main branch occurred. The results show that all ten test cases achieved a passed status, while the numbers of failed, flaky, and skipped tests were all zero. Manual testing required 157 seconds or 2 minutes and 37 seconds, whereas automated testing using Playwright required 15.0 seconds, indicating that the automated testing execution time was approximately 90.4% shorter for the scenarios used in this study. GitHub Actions successfully executed the workflow with a Success status and generated test logs and report artifacts. Based on these results,

functional test automation using Playwright and GitHub Actions was successfully implemented as part of the Continuous Integration process and was able to execute the defined test scenarios more quickly, consistently, and repeatedly.

Keywords: *Continuous Integration, E-Commerce, Functional Testing, GitHub Actions, Playwright.*

A. PENDAHULUAN

Pengembangan aplikasi *e-commerce* terus mengalami perubahan karena adanya kebutuhan pengguna, perbaikan kesalahan, dan penambahan fitur baru. Perubahan tersebut membuat kode aplikasi sering diperbarui dan digabungkan ke dalam repositori utama. Penelitian (Wang et al., 2022) menjelaskan perubahan kebutuhan pengguna dan perkembangan teknologi mendorong perangkat lunak untuk terus mengalami pembaruan dalam waktu yang lebih cepat. Oleh karena itu, kualitas aplikasi perlu dijaga selama proses pengembangan berlangsung, bukan hanya diperiksa pada saat aplikasi telah selesai dibuat.

Salah satu cara untuk menjaga kualitas aplikasi adalah dengan melakukan pengujian fungsional. Penelitian Riza et al. (2024) menjelaskan bahwa pengujian fungsional merupakan tahap penting dalam pengembangan aplikasi untuk memastikan aplikasi dapat berjalan dengan benar dan sesuai dengan spesifikasi. Penelitian (Balsam & Mishra, 2025) juga menjelaskan bahwa perkembangan aplikasi web yang semakin kompleks menimbulkan berbagai tantangan dalam proses pengujian. Oleh karena itu, pengujian perlu dilakukan menggunakan metode yang sesuai agar fungsi aplikasi web dapat diperiksa secara menyeluruh dan kesalahan dapat diketahui sebelum aplikasi digunakan oleh pengguna. Apabila salah satu fungsi utama mengalami kesalahan, proses transaksi dapat terganggu dan keandalan aplikasi dapat menurun. Pada aplikasi *e-commerce* yang menjadi objek penelitian, pengujian fungsional masih dilakukan secara manual.

Pada proses pengembangan yang sering mengalami perubahan kode, pengujian fungsional secara manual dapat menimbulkan beberapa masalah. (Technology, 2024) juga menunjukkan bahwa pengujian otomatis dapat mengurangi waktu pengujian, meningkatkan efisiensi, dan menghasilkan laporan yang lebih akurat. Penelitian (Y. Wang et al., 2022) menjelaskan bahwa otomatisasi pengujian yang lebih matang berkaitan dengan kualitas perangkat lunak yang lebih baik dan siklus rilis yang lebih singkat dalam lingkungan *Continuous Integration*.

Penelitian (Soares et al., 2022) menjelaskan bahwa *Continuous Integration* meliputi beberapa praktik, seperti melakukan *commit* secara rutin, menjalankan pengujian otomatis, melakukan *build* secara berkala, memperbaiki *build* yang gagal, dan memberikan umpan balik kepada tim pengembang. Dengan adanya *Continuous Integration*, kesalahan pada aplikasi dapat diketahui lebih awal setelah perubahan kode dilakukan. Oleh karena itu, pengujian fungsional perlu diintegrasikan ke dalam proses *Continuous Integration* agar pengujian dapat berjalan secara konsisten pada setiap perubahan kode.

Salah satu platform yang dapat digunakan untuk menjalankan *Continuous Integration* adalah GitHub Actions. Penelitian (Wessel et al., 2023) juga menemukan bahwa hampir 30% dari 5.000 repositori populer yang diteliti telah menggunakan GitHub Actions. Namun, pengembang masih sering membutuhkan bantuan dalam proses implementasinya. Hal ini menunjukkan bahwa GitHub Actions memiliki kemampuan untuk mendukung otomatisasi, tetapi tetap memerlukan konfigurasi *workflow* yang sesuai.

Untuk mendukung proses pengujian fungsional dalam otomatisasi pengujian, digunakan *framework* pengujian berbasis web yang dapat mengeksekusi *test case* secara otomatis. Salah satu *framework* yang digunakan adalah Playwright. Berdasarkan penelitian oleh (Gosik & Miłosz, 2025), Playwright memiliki performa yang lebih baik dibandingkan

Cypress dalam pengujian *end-to-end*, khususnya pada aspek kecepatan eksekusi, penggunaan memori yang lebih rendah, serta stabilitas pengujian pada lingkungan *multi-browser*. Hal ini menunjukkan bahwa Playwright lebih efisien dan cocok digunakan dalam pengujian aplikasi web modern.

Meskipun GitHub Actions dapat digunakan untuk menjalankan berbagai proses otomatisasi, pemanfaatan alat otomatisasi dalam *workflow* masih belum merata. Penelitian (Faqih et al., 2024) menganalisis lima kategori alat yang digunakan pada *workflow* GitHub Actions, yaitu *version control management*, *static code analysis*, *build automation*, *test automation*, dan *CI server*. Hasil penelitian tersebut menunjukkan bahwa *build automation* merupakan kategori yang paling sedikit digunakan. Temuan ini menunjukkan bahwa pemanfaatan berbagai alat otomatisasi dalam *workflow* GitHub Actions masih perlu ditingkatkan. Selain itu, penerapan pengujian fungsional otomatis tetap membutuhkan penentuan fitur yang diuji, pembuatan *test case*, pemilihan *testing framework*, pengaturan *trigger*, dan pengelolaan hasil pengujian.

Berdasarkan kondisi tersebut, terdapat kebutuhan untuk menerapkan pengujian fungsional yang dapat berjalan secara otomatis setiap kali terjadi perubahan kode pada aplikasi *e-commerce*. Tanpa adanya pengujian otomatis yang terintegrasi, pengembang harus menjalankan pengujian secara terpisah sehingga hasil pengujian tidak dapat diketahui secara langsung setelah proses integrasi kode. Hal tersebut dapat menyebabkan kesalahan pada fungsi utama aplikasi terlambat terdeteksi dan proses pengembangan menjadi kurang efisien. Oleh karena itu, penelitian ini mengimplementasikan otomatisasi pengujian fungsional menggunakan Playwright yang terintegrasi dengan GitHub Actions pada proses *Continuous Integration* aplikasi *e-commerce*. *Workflow* yang dibuat digunakan untuk menjalankan *test case* secara otomatis serta menampilkan status keberhasilan, kegagalan, dan *log* hasil pengujian. Berdasarkan permasalahan tersebut, penelitian ini berjudul “Implementasi Otomatisasi Pengujian Fungsional Menggunakan Playwright dan GitHub Actions pada Proses *Continuous Integration* Aplikasi *E-Commerce*”.

Pengujian fungsional merupakan salah satu aspek penting dalam pengembangan aplikasi *e-commerce* untuk memastikan bahwa setiap fitur berjalan sesuai dengan kebutuhan dan spesifikasi yang telah ditetapkan. Seiring dengan perubahan kode yang dilakukan selama proses pengembangan, pengujian perlu dilaksanakan secara konsisten untuk mengidentifikasi potensi kesalahan yang dapat memengaruhi fungsi aplikasi. Namun, pengujian yang dilakukan secara manual berpotensi membutuhkan waktu dan sumber daya yang lebih besar, terutama ketika pengujian harus diulang setiap kali terjadi perubahan kode. Oleh karena itu, otomatisasi pengujian fungsional yang terintegrasi dengan proses *Continuous Integration* (CI) menjadi salah satu pendekatan yang dapat mendukung pelaksanaan pengujian secara lebih sistematis dan berulang.

Melalui penerapan tersebut, penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan pengetahuan dan praktik pengujian perangkat lunak, khususnya terkait penyusunan *test case*, konfigurasi *workflow*, pelaksanaan pengujian otomatis, dan analisis hasil pengujian dalam lingkungan CI. Bagi lingkungan akademik, khususnya Universitas Pelita Bangsa, hasil penelitian ini dapat menjadi referensi pembelajaran di bidang rekayasa perangkat lunak, pengujian perangkat lunak, dan *Continuous Integration*. Sementara itu, bagi pengembang perangkat lunak, penelitian ini diharapkan dapat memberikan gambaran praktis mengenai pemanfaatan Playwright dan GitHub Actions untuk menjalankan pengujian secara otomatis ketika terjadi perubahan kode, memperoleh informasi hasil pengujian, serta memanfaatkan status dan *log* pengujian dalam mengidentifikasi dan memperbaiki kesalahan fungsional aplikasi.

B. TINJAUAN PUSTAKA

1. Otomatisasi Pengujian

Otomatisasi pengujian merupakan proses pengujian perangkat lunak yang dilakukan dengan menggunakan alat dan skrip untuk menjalankan *test case* secara otomatis. Otomatisasi digunakan untuk mengurangi kegiatan pengujian yang dilakukan secara manual, terutama ketika skenario yang sama perlu dijalankan kembali setelah terjadi perubahan pada kode aplikasi.

Penelitian yang dilakukan oleh (Bauer et al., 2024) menjelaskan bahwa penggunaan alat untuk membantu proses pengujian dapat meningkatkan efisiensi dibandingkan pengujian antarmuka yang dilakukan sepenuhnya secara manual. Hasil penelitian tersebut menunjukkan bahwa pengujian berbantuan alat dapat mempersingkat durasi pelaksanaan pengujian. Hal ini menunjukkan bahwa pemanfaatan alat pengujian dapat membantu penguji dalam menjalankan skenario secara lebih cepat dan terarah.

Penelitian (Ricca et al., 2025) juga menjelaskan bahwa otomatisasi pengujian dapat mendukung berbagai kegiatan, seperti pengujian unit, integrasi, penerimaan, dan pengujian aplikasi berbasis web. Namun, penerapan otomatisasi juga memiliki tantangan, terutama dalam proses pembuatan dan pemeliharaan skrip pengujian. Oleh karena itu, *test case* dan skrip pengujian perlu dirancang dengan baik agar dapat digunakan kembali ketika aplikasi mengalami perubahan. Pengujian dilakukan terhadap empat fitur utama, yaitu *login*, pencarian produk, keranjang belanja, dan *checkout*. Gambaran mengenai kasus, skenario, alur, dan hasil otomatisasi pengujian ditunjukkan pada Gambar 1.



Gambar 1. Skenario Otomatisasi Pengujian Fungsional pada Aplikasi E-Commerce

Berdasarkan Gambar 1, skenario pengujian dimulai dari fitur *login*, yang diuji menggunakan akun valid, kata sandi tidak valid, dan format email tidak valid. Fitur pencarian produk diuji dengan mencari produk yang tersedia dan produk yang tidak tersedia. Pada fitur keranjang belanja, pengujian dilakukan dengan menambahkan produk serta mengubah jumlah

produk. Sementara itu, fitur *checkout* diuji menggunakan data lengkap, tanpa alamat pengiriman, dan kondisi keranjang kosong.

2. *Continuous Integration*

Continuous Integration merupakan praktik dalam pengembangan perangkat lunak yang dilakukan dengan menggabungkan perubahan kode secara rutin ke dalam repositori utama. Setiap perubahan kode kemudian diperiksa melalui proses *build* dan pengujian otomatis.

Penelitian yang dilakukan oleh (Santos et al., 2025) menjelaskan bahwa penerapan *Continuous Integration* perlu disertai dengan pemantauan terhadap praktik-praktik yang dijalankan. Beberapa bagian yang perlu diperhatikan meliputi pengujian otomatis, cakupan pengujian, kondisi *build*, dan waktu yang dibutuhkan untuk memperbaiki *build* yang gagal. Pemantauan tersebut dapat membantu pengembang mengetahui kondisi sistem serta menemukan bagian dari proses integrasi yang perlu diperbaiki.

Penelitian (Amanda, 2024) juga menjelaskan bahwa waktu pelaksanaan *build* yang terlalu lama dapat membuat pengembang menunggu lebih lama untuk memperoleh hasil atau umpan balik dari proses *Continuous Integration*. Penelitian tersebut menunjukkan bahwa penggunaan proses *build* secara paralel dan *incremental* dapat membantu mempercepat waktu *build*.

3. *GitHub Actions*

GitHub Actions merupakan layanan otomatisasi yang terintegrasi dengan GitHub untuk menjalankan berbagai proses dalam pengembangan perangkat lunak. Proses otomatisasi tersebut diatur melalui berkas *workflow* dengan format YAML yang disimpan di dalam repositori. *Workflow* dapat dijalankan berdasarkan aktivitas tertentu, seperti *push*, *pull request*, atau jadwal yang telah ditentukan.

Penelitian yang dilakukan oleh (Rostami et al., 2026) menjelaskan bahwa GitHub Actions menggunakan berkas *workflow* berformat YAML untuk mengatur proses otomatisasi pada repositori. Penelitian Onori (Delicheh et al., 2026) juga menjelaskan bahwa GitHub Actions dapat digunakan untuk mengotomatisasi berbagai tugas dalam pengembangan perangkat lunak, seperti proses *build*, pengujian, rilis, dan *deployment*. Penelitian tersebut juga menunjukkan bahwa penggunaan GitHub Actions banyak difokuskan pada tugas utama dalam *Continuous Integration*. Meskipun demikian, pengembang tetap menghadapi beberapa kendala dalam proses penulisan, pengujian, *debugging*, dan pemeliharaan *workflow*.

Penelitian (Zheng et al., 2026) juga menjelaskan bahwa kegagalan *workflow* GitHub Actions dapat disebabkan oleh berbagai masalah yang terjadi selama proses eksekusi. Penelitian tersebut menunjukkan bahwa pemeriksaan *log workflow* diperlukan untuk mengetahui jenis kegagalan dan menentukan bagian yang perlu diperbaiki. Oleh karena itu, hasil dan *log* pengujian perlu disimpan agar pengembang dapat menemukan penyebab kegagalan dengan lebih mudah.

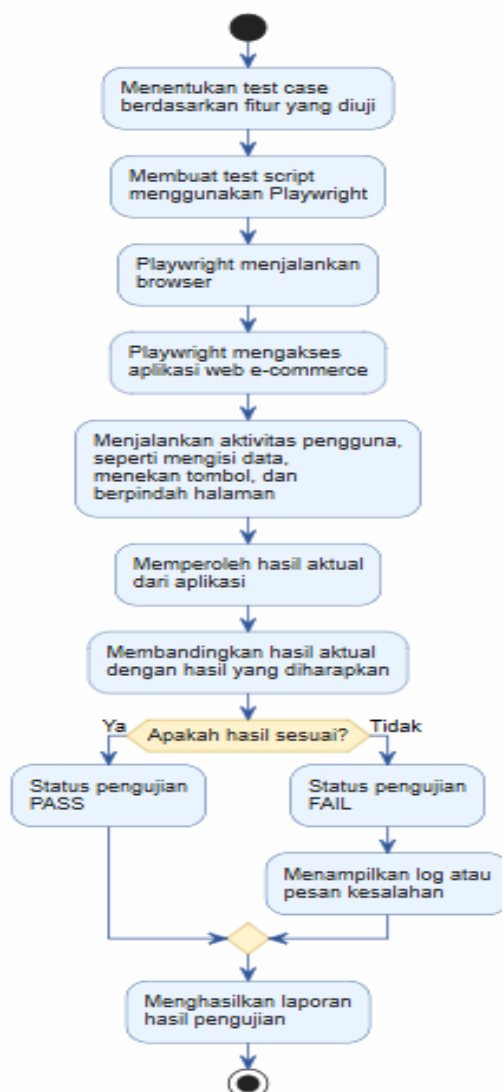
4. *Playwright*

Penelitian yang dilakukan oleh (Tymoshchuk, 2025) menjelaskan bahwa Playwright merupakan salah satu *framework* pengujian modern yang dapat digunakan untuk menguji aplikasi web dinamis. Hasil perbandingan antara Playwright, Selenium, dan Cypress menunjukkan bahwa Playwright memiliki kinerja yang baik dalam menjalankan skenario pengujian *end-to-end*.

Penelitian (Almabruk et al., 2026) juga membandingkan efisiensi Playwright dan Selenium berdasarkan waktu respons, waktu pemuatan, dan *throughput*. Hasil penelitian tersebut menunjukkan bahwa Playwright memiliki waktu respons yang lebih cepat dan stabil

pada kondisi pengujian yang dilakukan. Meskipun demikian, pemilihan *framework* pengujian tetap perlu mempertimbangkan kondisi aplikasi, perangkat keras, dan kebutuhan pengujian.

Penelitian (Rezeki et al., 2026) juga membandingkan kinerja Selenium, Cypress, dan Playwright dalam menjalankan pengujian *end-to-end* pada aplikasi berbasis React dan Vue. Pengujian dilakukan menggunakan 25 *test case* pada beberapa peramban dengan mengukur waktu eksekusi, tingkat keberhasilan, penggunaan prosesor, dan penggunaan memori. Alur pengujian menggunakan Playwright ditunjukkan pada Gambar 2.



Gambar 2. Alur Pengujian Menggunakan Playwright

Gambar 2 menunjukkan bahwa proses pengujian dimulai dengan menentukan *test case* berdasarkan fitur aplikasi yang akan diuji. *Test case* kemudian diimplementasikan menjadi skrip Playwright yang digunakan untuk menjalankan aktivitas pengguna dan memeriksa respons aplikasi. Hasil aktual dibandingkan dengan hasil yang diharapkan melalui *assertion* untuk menentukan status *pass* atau *fail*.

5. Aplikasi E-commerce

Aplikasi *e-commerce* merupakan aplikasi yang digunakan untuk mendukung kegiatan jual beli barang atau jasa melalui jaringan internet. Melalui aplikasi tersebut, pengguna dapat memperoleh informasi produk, mencari dan memilih produk, memasukkan produk ke dalam keranjang belanja, serta melakukan proses pemesanan dan transaksi secara daring.

Penelitian yang dilakukan oleh (Hutahaean et al., 2024) menjelaskan bahwa kualitas fungsional menjadi bagian penting dari kegunaan aplikasi *e-commerce*. Aspek fungsional tersebut meliputi pencarian informasi, tampilan antarmuka, dan proses transaksi. Fungsi pencarian membantu pengguna menemukan informasi produk, tampilan antarmuka digunakan untuk menampilkan produk dan membantu pengguna berinteraksi dengan aplikasi, sedangkan fungsi transaksi digunakan untuk menyelesaikan proses pembelian.

Penelitian (Arujisaputra et al., 2025) juga menjelaskan bahwa aplikasi *e-commerce* berbasis web dapat membantu pelanggan memperoleh informasi mengenai produk, melakukan pemesanan, dan menyelesaikan pembelian secara daring. Perancangan aplikasi yang sesuai dengan kebutuhan pengguna dapat mempermudah proses promosi dan penjualan serta memberikan pengalaman penggunaan yang lebih baik kepada pelanggan.

C. METODE

Pengumpulan data dalam penelitian ini dilakukan untuk memperoleh informasi yang diperlukan dalam proses perancangan, implementasi, dan evaluasi otomatisasi pengujian fungsional menggunakan Playwright dan GitHub Actions. Pengumpulan data dilakukan melalui studi literatur, observasi, dan eksperimen.

Studi literatur dilakukan dengan mempelajari jurnal ilmiah, buku, artikel penelitian, dan dokumentasi resmi yang berkaitan dengan pengujian perangkat lunak, pengujian fungsional, otomatisasi pengujian, Playwright, GitHub Actions, dan Continuous Integration. Kegiatan ini bertujuan untuk memperoleh pemahaman mengenai konsep, metode, alat, dan tahapan penelitian yang digunakan sebagai dasar dalam menentukan fitur yang diuji, menyusun test case, membuat skrip pengujian menggunakan Playwright, merancang workflow otomatisasi melalui GitHub Actions, serta menganalisis hasil pengujian.

Observasi dilakukan dengan mengamati proses otomatisasi pengujian fungsional melalui GitHub Actions, mulai dari pemicu workflow oleh aktivitas push ke branch main atau pull request hingga seluruh tahapan pengujian selesai. Pengamatan meliputi penyiapan lingkungan pengujian, pelaksanaan aplikasi, eksekusi test case menggunakan Playwright, serta hasil laporan dan artifact yang dihasilkan. Data yang dicatat mencakup status keberhasilan atau kegagalan workflow, jumlah test case yang berhasil dan gagal, waktu pelaksanaan, serta log atau pesan kesalahan. Hasil observasi digunakan untuk mengevaluasi kesesuaian pelaksanaan workflow otomatisasi dengan rancangan dan kemampuannya dalam menjalankan seluruh test case pada lingkungan Continuous Integration.

Eksperimen dilakukan dengan menjalankan pengujian fungsional pada aplikasi *e-commerce* melalui pengujian manual dan otomatis. Pengujian manual dilakukan secara langsung melalui antarmuka untuk memperoleh data awal, sedangkan pengujian otomatis menggunakan skrip Playwright yang dijalankan melalui workflow GitHub Actions ketika terjadi aktivitas push ke branch main atau pull request yang ditujukan ke branch main. Kedua pengujian menggunakan test case dan hasil yang diharapkan sama agar dapat dibandingkan secara setara. Data yang dikumpulkan meliputi status keberhasilan atau kegagalan, jumlah test case yang berhasil dan gagal, waktu pelaksanaan, serta log atau pesan kesalahan. Data tersebut digunakan untuk mengevaluasi penerapan otomatisasi pengujian fungsional dan membandingkan pelaksanaannya dengan pengujian manual.

Pengujian fungsional manual dilakukan sebagai tahap awal untuk memperoleh gambaran mengenai proses dan waktu pengujian sebelum diterapkannya otomatisasi menggunakan Playwright dan GitHub Actions. Pengujian dilakukan dengan menjalankan setiap test case berdasarkan skenario dan data uji yang telah ditentukan tanpa alat otomatisasi. Data yang dicatat meliputi hasil pengujian, status berhasil atau gagal, waktu pelaksanaan, serta kendala yang ditemukan. Seluruh data disusun secara terstruktur sebagai bahan evaluasi dan perbandingan dengan pengujian otomatis, sebagaimana ditunjukkan pada Tabel 2.

Tabel 1. Rancangan Pencatatan Data Pengujian Fungsional Manual

No./ID	Fitur dan Skenario Pengujian	Hasil yang Diharapkan	Hasil Aktual dan Evaluasi
1. TC-01	Login: Menggunakan akun valid.	Login berhasil dan pengguna diarahkan ke halaman utama.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
2. TC-02	Login: Menggunakan kata sandi tidak valid.	Login ditolak dan pesan kesalahan ditampilkan.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
3. TC-03	Login: Menggunakan format email tidak valid.	Login ditolak dan pesan validasi format email ditampilkan.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
4. TC-04	Pencarian produk: Mencari produk yang tersedia.	Sistem menampilkan produk yang sesuai dengan kata kunci.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
5. TC-05	Pencarian produk: Mencari produk yang tidak tersedia.	Sistem menampilkan informasi bahwa produk tidak ditemukan.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
6. TC-06	Keranjang belanja: Menambahkan produk ke keranjang.	Produk berhasil ditambahkan dan ditampilkan dalam keranjang.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
7. TC-07	Keranjang belanja: Mengubah jumlah produk.	Jumlah produk dan total harga diperbarui sesuai jumlah terbaru.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
8. TC-08	Checkout: Melakukan checkout dengan data lengkap.	Checkout berhasil dan halaman konfirmasi pesanan ditampilkan.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
9. TC-09	Checkout: Melakukan checkout tanpa alamat pengiriman.	Checkout ditolak dan pengguna diminta melengkapi alamat pengiriman.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...
10. TC-10	Checkout: Melakukan checkout dengan keranjang kosong.	Sistem mencegah checkout dan menampilkan informasi bahwa keranjang kosong.	Hasil aktual: ...; Status: ...; Waktu: ... detik; Catatan: ...

D. HASIL DAN PEMBAHASAN

Objek dalam penelitian ini adalah aplikasi *e-commerce* berbasis web bernama ShopFlow. Aplikasi ini dikembangkan menggunakan bahasa pemrograman PHP dan basis data MySQL serta dijalankan pada lingkungan lokal menggunakan XAMPP. Kode sumber aplikasi disimpan dan dikelola melalui repositori GitHub untuk mendukung pengelolaan perubahan kode serta penerapan proses *Continuous Integration*. Tampilan halaman utama aplikasi ShopFlow yang digunakan sebagai objek penelitian ditunjukkan pada Gambar 4.

**Gambar 4. Halaman Utama Aplikasi ShopFlow**

Gambar 4 menunjukkan halaman utama aplikasi ShopFlow yang digunakan sebagai objek penelitian. Halaman tersebut menampilkan navigasi utama, kolom pencarian produk, daftar produk, akses menuju keranjang belanja, serta menu akun pengguna. Melalui halaman utama tersebut, pengguna dapat mencari dan memilih produk sebelum melanjutkan ke proses keranjang belanja dan *checkout*.

ShopFlow menyediakan berbagai fungsi yang mendukung kegiatan transaksi secara daring, mulai dari autentikasi pengguna, pencarian produk, pengelolaan keranjang belanja, pengisian alamat pengiriman, pemilihan layanan pengiriman, pemilihan metode pembayaran, hingga pembuatan pesanan. Aplikasi juga menyediakan halaman administrasi untuk mendukung pengelolaan data produk, pesanan, pengiriman, promosi, ulasan, retur, dan laporan transaksi.

Meskipun ShopFlow memiliki berbagai fungsi, pengujian dalam penelitian ini dibatasi pada empat fitur utama, yaitu *login*, pencarian produk, keranjang belanja, dan *checkout*. Fitur *login* digunakan untuk melakukan autentikasi pengguna berdasarkan data akun yang dimasukkan serta memberikan validasi apabila data autentikasi tidak sesuai, termasuk kata sandi dan format email yang tidak valid. Fitur pencarian produk digunakan untuk menampilkan produk berdasarkan kata kunci. Fitur keranjang belanja digunakan untuk menambahkan produk serta mengubah jumlah produk yang akan dibeli. Fitur *checkout* digunakan untuk memproses pembuatan pesanan, memeriksa kelengkapan alamat pengiriman, serta mencegah proses *checkout* apabila keranjang belanja masih kosong.

Keempat fitur tersebut diuji menggunakan sepuluh *test case*. Skenario pengujian meliputi *login* menggunakan akun valid, *login* menggunakan kata sandi tidak valid, *login* menggunakan format email tidak valid, pencarian produk yang tersedia dan tidak tersedia, penambahan produk ke keranjang, perubahan jumlah produk, *checkout* dengan data lengkap, *checkout* tanpa alamat pengiriman, serta *checkout* dengan keranjang kosong. Pengujian dilakukan untuk mengetahui kesesuaian antara respons aplikasi dengan hasil yang diharapkan pada setiap skenario.

2. Lingkungan Implementasi dan Pengujian

Implementasi dan pengujian dilakukan pada dua lingkungan, yaitu lingkungan lokal dan lingkungan *Continuous Integration*. Lingkungan lokal digunakan untuk menjalankan aplikasi, melakukan pengujian fungsional secara manual, menyusun skrip pengujian Playwright, serta memeriksa setiap *test case* sebelum perubahan kode dikirimkan ke repositori GitHub. Sementara itu, lingkungan *Continuous Integration* digunakan untuk menjalankan pengujian fungsional secara otomatis melalui GitHub Actions ketika terjadi aktivitas *push* ke *branch* main atau *pull request* yang ditujukan ke *branch* main. Perangkat keras yang digunakan selama proses implementasi dan pengujian ditunjukkan pada Tabel 4.1

Tabel 2. Perangkat Keras yang Digunakan Selama Proses Implementasi dan Pengujian

No.	Komponen	Spesifikasi	Penggunaan
1	Laptop	Prosesor Intel Core i5	Mengelola kode sumber, menjalankan aplikasi, dan menyusun skrip pengujian
2	Memori	RAM 8 GB	Mendukung proses pengembangan dan pelaksanaan pengujian
3	Penyimpanan	238 GB	Menyimpan kode sumber, basis data, dependensi, dan laporan pengujian
4	Koneksi Internet	Stabil	Mengakses GitHub serta menjalankan aktivitas <i>push</i> , <i>pull request</i> , dan GitHub Actions

Berdasarkan Tabel 3, perangkat keras yang digunakan mendukung pengelolaan kode sumber, pelaksanaan aplikasi, penyusunan skrip pengujian Playwright, pengelolaan repositori GitHub, serta pemantauan hasil eksekusi *workflow*. Spesifikasi tersebut sesuai dengan perangkat keras yang direncanakan dalam metode penelitian.

Selain perangkat keras, proses implementasi memerlukan perangkat lunak dan layanan yang mendukung pelaksanaan aplikasi, pengujian otomatis, serta proses *Continuous Integration*. Perangkat lunak yang digunakan ditunjukkan pada Tabel 4.

Tabel 3. Perangkat Lunak yang Digunakan

No.	Perangkat Lunak	Fungsi
1	Windows	Sistem operasi yang digunakan sebagai lingkungan kerja lokal
2	Visual Studio Code	Mengelola kode aplikasi, skrip pengujian, dan konfigurasi workflow
3	XAMPP	Menjalankan lingkungan server lokal untuk aplikasi
4	PHP	Menjalankan aplikasi e-commerce berbasis web
5	MySQL	Menyimpan dan mengelola data aplikasi serta data pengujian
6	Git	Mencatat dan mengelola perubahan kode sumber
7	GitHub	Menyimpan repositori dan mengelola aktivitas push serta pull request
8	Node.js dan npm	Menjalankan Playwright dan mengelola dependensi pengujian
9	Playwright	Menjalankan sepuluh test case secara otomatis
10	Chromium	Peramban yang digunakan dalam pelaksanaan pengujian otomatis

Berdasarkan Tabel 4, perangkat lunak yang digunakan mendukung proses pelaksanaan aplikasi, pengelolaan kode sumber, penyusunan skrip pengujian, dan otomatisasi pengujian melalui GitHub Actions. Pada lingkungan lokal, aplikasi dijalankan menggunakan XAMPP dan diakses melalui alamat dasar: <http://localhost/shopflow-php>. Alamat digunakan sebagai baseURL pada konfigurasi Playwright sehingga skrip pengujian dapat mengakses halaman aplikasi menggunakan alamat relatif.

Pengujian Playwright dijalankan secara berurutan menggunakan satu *worker* karena seluruh *test case* menggunakan basis data pengujian yang sama. Konfigurasi Playwright juga menghasilkan laporan HTML serta menyimpan *screenshot* dan video apabila terjadi kegagalan, sedangkan *trace* disimpan pada pengulangan pertama setelah pengujian mengalami kegagalan.

Pada lingkungan GitHub Actions, aplikasi dan basis data disiapkan kembali sebelum pengujian dijalankan. Basis data yang digunakan berisi data sintesis khusus pengujian, seperti akun pelanggan, alamat pengiriman, kategori, produk, dan data pendukung transaksi. Penggunaan data sintesis dilakukan agar pengujian tidak menggunakan data pelanggan sebenarnya dan dapat dijalankan dari kondisi data yang telah ditentukan.

a. Implementasi Skenario Pengujian Tabel

Skenario pengujian fungsional diterapkan pada empat fitur utama aplikasi, yaitu *login*, pencarian produk, keranjang belanja, dan *checkout*. Fitur *login* dan *checkout* masing-masing diuji menggunakan tiga skenario, sedangkan fitur pencarian produk dan keranjang belanja masing-masing diuji menggunakan dua skenario sehingga keseluruhan pengujian terdiri atas sepuluh *test case*. Skenario pengujian tersebut digunakan pada pengujian manual maupun pengujian otomatis.

Setiap *test case* memiliki ID, fitur yang diuji, skenario pengujian, data uji, serta hasil yang diharapkan. Hasil aktual dari setiap skenario dibandingkan dengan kondisi yang telah ditentukan. Apabila hasil aktual sesuai dengan hasil yang diharapkan, *test case* dinyatakan *pass*. Sebaliknya, apabila kondisi yang diperiksa tidak terpenuhi, *test case* dinyatakan *fail*. Skenario pengujian fungsional yang diterapkan ditunjukkan pada Tabel 5.

Tabel 5. Skenario Pengujian Fungsional yang Diterapkan

No.	ID Test Case	Fitur	Skenario Pengujian	Data Uji	Hasil yang Diharapkan
1	TC-01	Login	Melakukan login menggunakan akun yang valid	Email dan kata sandi akun pengujian yang valid	Pengguna berhasil masuk dan diarahkan ke halaman utama aplikasi
2	TC-02	Login	Melakukan login menggunakan kata sandi yang tidak valid	Email akun pengujian yang valid dan kata sandi yang salah	Sistem menolak proses login, pengguna tetap berada pada halaman login, dan pesan kesalahan ditampilkan

3	TC-03	Login	Melakukan login menggunakan format email yang tidak valid	Email dengan format tidak valid dan kata sandi pengujian	Sistem menolak proses login, pengguna tetap berada pada halaman login, dan pesan validasi format email ditampilkan
4	TC-04	Pencarian produk	Mencari produk yang tersedia	Nama produk yang tersedia pada aplikasi	Sistem menampilkan produk yang sesuai dengan kata kunci pencarian
5	TC-05	Pencarian produk	Mencari produk yang tidak tersedia	Nama atau kata kunci produk yang tidak terdapat pada aplikasi	Sistem menampilkan jumlah hasil nol dan informasi bahwa produk tidak ditemukan
6	TC-06	Keranjang belanja	Menambahkan produk ke keranjang	Produk yang tersedia dan memiliki stok	Produk berhasil ditambahkan dan ditampilkan pada halaman keranjang
7	TC-07	Keranjang belanja	Mengubah jumlah produk dari satu menjadi dua	Produk pada keranjang dengan jumlah awal satu	Jumlah produk, total harga produk, dan subtotal keranjang diperbarui sesuai jumlah terbaru
8	TC-08	Checkout	Melakukan checkout dengan data lengkap	Akun yang memiliki alamat, produk, layanan pengiriman, dan metode pembayaran	Sistem berhasil memproses checkout, membuat pesanan dan menampilkan halaman konfirmasi pesanan
9	TC-09	Checkout	Melakukan checkout tanpa alamat pengiriman	Akun pengujian yang belum memiliki alamat pengiriman	Sistem menampilkan informasi bahwa alamat pengiriman belum tersedia, mencegah proses checkout, dan pesanan tidak dibuat
10	TC-10	Checkout	Melakukan checkout dengan keranjang kosong	Akun pengujian dengan kondisi keranjang belanja kosong	Sistem mencegah proses checkout dan menampilkan informasi bahwa keranjang belanja masih kosong

Berdasarkan Tabel 5, TC-01, TC-02, dan TC-03 digunakan untuk memeriksa proses autentikasi pengguna. TC-01 menggunakan email dan kata sandi yang valid sehingga pengguna diharapkan berhasil masuk dan diarahkan ke halaman utama aplikasi. Pada TC-02, email akun pengujian tetap valid tetapi kata sandi yang digunakan tidak sesuai sehingga sistem diharapkan menolak proses *login* dan menampilkan pesan kesalahan. Sementara itu, TC-03 menggunakan format email yang tidak valid sehingga sistem diharapkan menolak proses *login* dan menampilkan pesan validasi format email.

TC-04 dan TC-05 digunakan untuk memeriksa fungsi pencarian produk. TC-04 menggunakan kata kunci yang sesuai dengan produk yang tersedia sehingga produk tersebut diharapkan tampil pada hasil pencarian. TC-05 menggunakan kata kunci yang tidak memiliki kecocokan dengan data produk sehingga sistem diharapkan menampilkan jumlah hasil nol dan informasi bahwa produk tidak ditemukan. TC-06 dan TC-07 digunakan untuk memeriksa fungsi keranjang belanja. TC-06 memastikan produk yang tersedia dan memiliki stok dapat ditambahkan serta ditampilkan pada halaman keranjang. TC-07 dilakukan dengan mengubah jumlah produk dari satu menjadi dua. Setelah jumlah diperbarui, aplikasi diharapkan menyesuaikan jumlah produk, total harga produk, dan subtotal keranjang.

TC-08 digunakan untuk memeriksa proses *checkout* dengan data lengkap. Akun pengujian memiliki alamat pengiriman, sedangkan produk, layanan pengiriman, dan metode pembayaran tersedia untuk digunakan. Setelah proses pembuatan pesanan dijalankan, sistem diharapkan membuat pesanan dan menampilkan halaman konfirmasi. TC-09 digunakan untuk memeriksa proses *checkout* ketika akun pengujian belum memiliki alamat pengiriman. Sistem diharapkan menampilkan informasi bahwa alamat pengiriman belum tersedia dan mencegah

pengguna melanjutkan proses pembuatan pesanan sehingga pesanan tidak dibuat. TC-10 digunakan untuk memeriksa proses *checkout* ketika keranjang belanja dalam kondisi kosong. Sistem diharapkan mencegah pengguna melanjutkan proses *checkout* dan menampilkan informasi bahwa keranjang belanja masih kosong.

Implementasi Pengujian Menggunakan Playwright

Pengujian fungsional otomatis diimplementasikan menggunakan Playwright dengan bahasa pemrograman TypeScript. Sepuluh *test case* diimplementasikan dalam bentuk skrip pengujian dan ditempatkan pada direktori *tests*. Setiap skrip digunakan untuk menjalankan skenario pengujian terhadap fitur *login*, pencarian produk, keranjang belanja, dan *checkout*.

Playwright menjalankan pengujian menggunakan peramban Chromium untuk mensimulasikan aktivitas pengguna pada aplikasi. Aktivitas tersebut meliputi membuka halaman, mengisi kolom masukan, menekan tombol, memilih menu atau pilihan yang tersedia, berpindah halaman, serta memeriksa informasi yang ditampilkan oleh aplikasi. Pemeriksaan hasil dilakukan menggunakan *assertion* untuk menentukan apakah respons aktual aplikasi sesuai dengan kondisi yang diharapkan pada setiap *test case*. Implementasi pengujian menggunakan Playwright meliputi konfigurasi pengujian, struktur skrip, implementasi skrip pada masing-masing fitur, serta penentuan hasil pengujian berdasarkan *assertion*.

a. Konfigurasi Playwright

Konfigurasi Playwright digunakan untuk mengatur lokasi berkas pengujian, jumlah *worker*, batas waktu pelaksanaan, pengulangan pengujian pada lingkungan *Continuous Integration*, peramban yang digunakan, serta laporan dan bukti pendukung yang dihasilkan. Konfigurasi tersebut diterapkan melalui berkas *playwright.config.ts*. Rincian konfigurasi pengujian Playwright ditunjukkan pada Tabel 6.

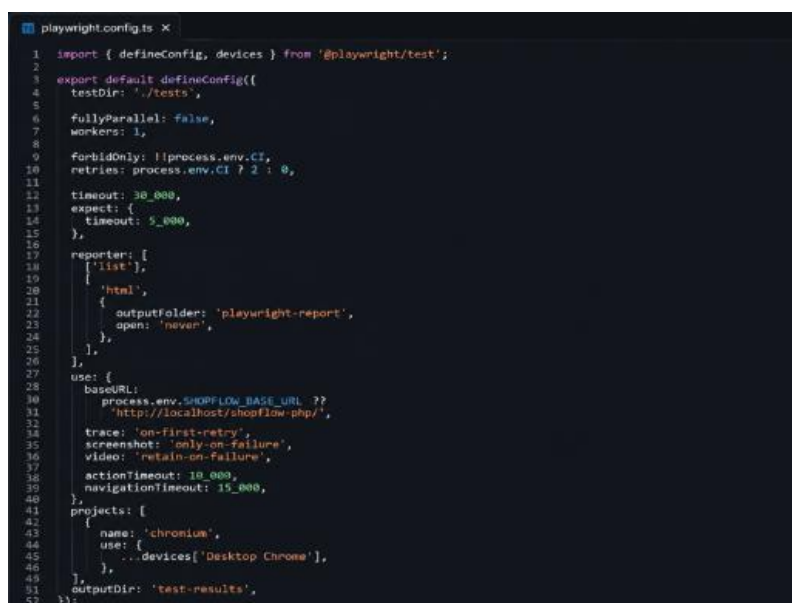
Tabel 6. Rincian konfigurasi pengujian Playwright

No.	Komponen Konfigurasi	Nilai	Fungsi
1	Direktori pengujian	./tests	Menentukan lokasi seluruh berkas test case
2	Pelaksanaan paralel	false	Menonaktifkan pelaksanaan pengujian secara paralel
3	Jumlah worker	1	Menjalankan test case secara berurutan
4	Pengulangan pada CI	2 kali	Mengulangi pengujian yang gagal pada lingkungan CI
5	Batas waktu test case	30 detik	Membatasi waktu pelaksanaan setiap test case
6	Batas waktu assertion	5 detik	Menunggu terpenuhinya kondisi pemeriksaan
7	Batas waktu aksi	10 detik	Membatasi waktu setiap interaksi terhadap elemen halaman
8	Batas waktu navigasi	15 detik	Membatasi waktu perpindahan halaman
9	Reporter	list dan html	Menampilkan hasil pada terminal dan menghasilkan laporan HTML
10	Peramban	Chromium	Menjalankan pengujian menggunakan proyek Chromium
11	Screenshot	Saat terjadi kegagalan	Menyimpan tampilan halaman apabila pengujian gagal
12	Video	Disimpan saat gagal	Menyimpan rekaman pengujian apabila terjadi kegagalan
13	Trace	Pada pengulangan pertama	Menyimpan jejak pengujian pada pengulangan pertama setelah kegagalan
14	Direktori hasil	test-results	Menyimpan hasil dan bukti pendukung pengujian

Berdasarkan Tabel 6, pengujian dijalankan secara berurutan menggunakan satu *worker* karena seluruh *test case* menggunakan basis data pengujian yang sama. Pengaturan tersebut digunakan untuk mengurangi kemungkinan perubahan data secara bersamaan yang dapat memengaruhi hasil pengujian. Pada lingkungan *Continuous Integration*, pengujian yang mengalami kegagalan dapat diulang sebanyak dua kali. Hasil pengujian ditampilkan melalui

terminal dan laporan HTML, sedangkan *screenshot*, video, dan *trace* digunakan sebagai bukti pendukung apabila terjadi kegagalan.

Alamat dasar aplikasi ditentukan melalui bagian `baseUrl`. Pada lingkungan lokal, aplikasi diakses melalui alamat: <http://localhost/shopflow-php/>, sedangkan pada lingkungan GitHub Actions alamat dasar aplikasi diperoleh dari variabel lingkungan `SHOPFLOW_BASE_URL`. Penggunaan `baseUrl` memungkinkan skrip pengujian mengakses halaman menggunakan alamat relatif seperti `login.php`, `indek.php`, `cart.php`, dan `checkout.php`. Konfigurasi Playwright yang digunakan dalam pengujian fungsional ditunjukkan pada Gambar 5.



```

1 import { defineConfig, devices } from '@playwright/test';
2
3 export default defineConfig({
4   testDir: './tests',
5
6   fullyParallel: false,
7   workers: 1,
8
9   forbidOnly: !!process.env.CI,
10  retries: process.env.CI ? 2 : 0,
11
12  timeout: 30_000,
13  expect: {
14    timeout: 5_000,
15  },
16
17  reporter: [
18    ['list'],
19    ['html', {
20      outputFolder: 'playwright-report',
21      open: 'never',
22    }],
23  ],
24
25  use: {
26    baseUrl:
27      process.env.SHOPFLOW_BASE_URL ??
28      'http://localhost/shopflow-php/',
29    trace: 'on-first-retry',
30    screenshot: 'only-on-failure',
31    video: 'retain-on-failure',
32    actionTimeout: 10_000,
33    navigationTimeout: 15_000,
34  },
35  projects: [
36    {
37      name: 'chromium',
38      use: {
39        ...devices['Desktop Chrome'],
40      },
41    },
42  ],
43  outputDir: 'test-results',
44 });
  
```

Gambar 5. Konfigurasi Pengujian Playwright

Gambar 5 menunjukkan konfigurasi Playwright yang digunakan dalam pengujian fungsional. Konfigurasi tersebut mengatur direktori pengujian, jumlah *worker*, batas waktu, pengulangan pada lingkungan CI, bentuk laporan, alamat dasar aplikasi, peramban Chromium, serta penyimpanan bukti pendukung ketika terjadi kegagalan pengujian.

b. Struktur Skrip Pengujian

Setiap skrip pengujian menggunakan fungsi `test()` untuk mendefinisikan skenario pengujian dan fungsi `expect()` untuk melakukan pemeriksaan terhadap hasil aktual aplikasi. Elemen pada halaman ditentukan menggunakan *locator* agar Playwright dapat berinteraksi dengan kolom masukan, tombol, tautan, pilihan, dan informasi yang ditampilkan oleh aplikasi.

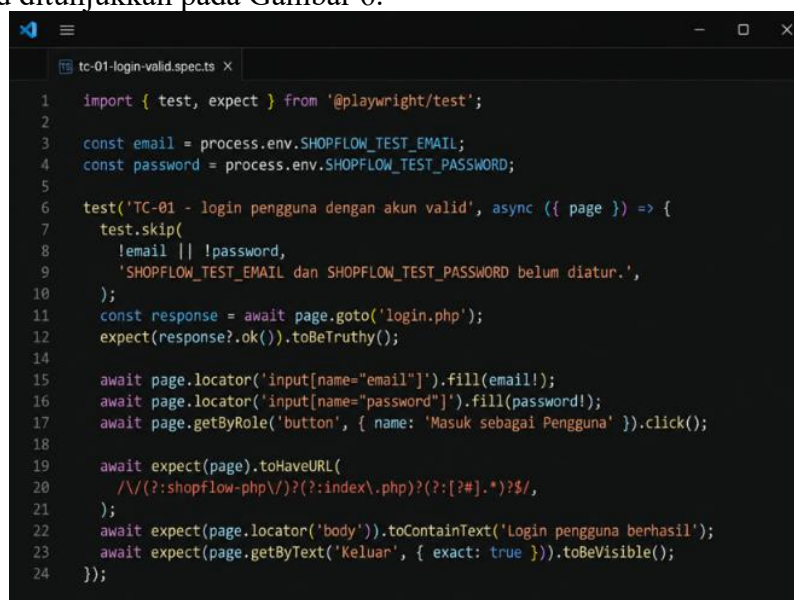
Data pengujian tertentu, seperti akun pengguna dan nama produk, diperoleh melalui variabel lingkungan. Penggunaan variabel lingkungan memungkinkan data pengujian digunakan secara konsisten pada lingkungan lokal maupun GitHub Actions tanpa menuliskan nilai yang sama secara berulang pada setiap skrip.

Secara umum, struktur pelaksanaan skrip pengujian terdiri atas tahapan berikut: 1) menyiapkan data pengujian; 2) membuka halaman aplikasi; 3) menentukan elemen halaman menggunakan *locator*; 4) menjalankan aktivitas pengguna sesuai skenario; 5) memeriksa respons aplikasi menggunakan *assertion*; dan 6) menentukan status *pass* atau *fail* berdasarkan hasil pemeriksaan.

Struktur skrip pengujian juga diterapkan pada skenario *checkout* dengan keranjang kosong. Pada skenario tersebut, kondisi awal pengujian memastikan bahwa keranjang belanja tidak memiliki produk. Skrip kemudian mencoba mengakses proses *checkout* dan melakukan pemeriksaan menggunakan *assertion* untuk memastikan bahwa sistem mencegah proses *checkout* serta menampilkan informasi bahwa keranjang belanja masih kosong. Skenario ini digunakan sebagai TC-10 dalam pengujian fungsional.

c. Implementasi Pengujian Skrip Fitur Login

Pengujian fitur *login* terdiri atas TC-01, TC-02 dan TC-03. TC-01 digunakan untuk menguji proses *login* menggunakan akun yang valid. Skrip membuka halaman *login*, mengisi alamat email dan kata sandi akun pengujian yang valid, kemudian menekan tombol masuk. Setelah proses autentikasi dilakukan, skrip memeriksa perpindahan halaman, pesan keberhasilan, dan keberadaan menu keluar. Apabila seluruh kondisi pemeriksaan terpenuhi, TC-01 dinyatakan *pass*. Implementasi skrip pengujian TC-01 untuk proses *login* menggunakan akun yang valid ditunjukkan pada Gambar 6.



```

1  import { test, expect } from '@playwright/test';
2
3  const email = process.env.SHOPFLOW_TEST_EMAIL;
4  const password = process.env.SHOPFLOW_TEST_PASSWORD;
5
6  test('TC-01 - login pengguna dengan akun valid', async ({ page }) => {
7    test.skip(
8      !email || !password,
9      'SHOPFLOW_TEST_EMAIL dan SHOPFLOW_TEST_PASSWORD belum diatur.',
10    );
11    const response = await page.goto('login.php');
12    expect(response?.ok()).toBeTruthy();
13
14
15    await page.locator('input[name="email"]').fill(email!);
16    await page.locator('input[name="password"]').fill(password!);
17    await page.getByRole('button', { name: 'Masuk sebagai Pengguna' }).click();
18
19    await expect(page).toHaveURL(
20      /\/(?!shopflow-php\/)?(?:index\.php)?(?:[#].*)?$/,
21    );
22    await expect(page.locator('body')).toContainText('Login pengguna berhasil');
23    await expect(page.getByText('Keluar', { exact: true })).toBeVisible();
24  });

```

Gambar 6. Implementasi Skrip Pengujian TC-01

Gambar 6 menunjukkan implementasi skrip TC-01 untuk menguji proses *login* menggunakan akun yang valid. Skrip mengisi alamat email dan kata sandi, menjalankan tombol masuk, kemudian melakukan pemeriksaan terhadap perpindahan halaman, pesan keberhasilan, dan keberadaan menu keluar menggunakan *assertion*.

TC-02 digunakan untuk menguji proses *login* menggunakan email akun pengujian yang valid dengan kata sandi yang tidak valid. Skrip mengisi alamat email dan kata sandi yang salah, kemudian menjalankan tombol masuk. Pemeriksaan dilakukan untuk memastikan bahwa pengguna tetap berada pada halaman *login*, pesan kesalahan ditampilkan, dan menu keluar tidak tersedia. Apabila seluruh kondisi terpenuhi, TC-02 dinyatakan *pass* karena aplikasi berhasil menolak proses autentikasi dengan kata sandi yang tidak sesuai.

TC-03 digunakan untuk menguji proses *login* menggunakan format email yang tidak valid. Skrip membuka halaman *login*, mengisi kolom email menggunakan format email yang tidak memenuhi ketentuan, kemudian mengisi kata sandi pengujian dan menjalankan tombol masuk. Pemeriksaan dilakukan untuk memastikan bahwa proses *login* ditolak, pengguna tetap berada pada halaman *login*, serta pesan validasi format email ditampilkan. Apabila seluruh kondisi tersebut terpenuhi, TC-03 dinyatakan *pass* karena aplikasi berhasil mencegah proses autentikasi menggunakan format email yang tidak valid.

d. Implementasi Skrip Pengujian Fitur Pencarian Produk

Pengujian fitur pencarian produk terdiri atas TC-04 dan TC-05. TC-04 digunakan untuk menguji pencarian terhadap produk yang tersedia pada aplikasi. Skrip membuka halaman aplikasi, mengisi kolom pencarian menggunakan nama produk yang tersedia, kemudian menjalankan proses pencarian. Setelah proses tersebut dilakukan, skrip memeriksa bahwa produk yang sesuai dengan kata kunci ditampilkan pada halaman hasil pencarian. Apabila kondisi tersebut terpenuhi, TC-04 dinyatakan *pass*.

Gambar 7 menunjukkan implementasi skrip TC-09 untuk menguji proses *checkout* tanpa alamat pengiriman. Skrip memeriksa informasi bahwa alamat pengiriman belum tersedia, tidak adanya alamat yang dipilih, kondisi metode pembayaran dan layanan pengiriman yang tidak aktif, serta tombol pembuatan pesanan yang tidak dapat digunakan. Pemeriksaan tersebut memastikan bahwa sistem mencegah proses pembuatan pesanan ketika alamat pengiriman belum tersedia.

TC-10 digunakan untuk menguji proses *checkout* ketika keranjang belanja dalam kondisi kosong. Skrip memastikan kondisi awal keranjang tidak memiliki produk, kemudian mencoba mengakses proses *checkout*. Pemeriksaan dilakukan untuk memastikan bahwa sistem mencegah pengguna melanjutkan proses *checkout* dan menampilkan informasi bahwa keranjang masih kosong. Apabila sistem menampilkan pesan "Keranjang Anda masih kosong." dan proses *checkout* tidak dapat dilanjutkan, TC-10 dinyatakan *pass*.

g. Penentu hasil Pengujian

Penentuan hasil pengujian dilakukan berdasarkan *assertion* yang terdapat pada setiap skrip Playwright. *Assertion* digunakan untuk memeriksa kesesuaian antara respons aktual aplikasi dan kondisi yang diharapkan pada masing-masing *test case*. Pemeriksaan dapat berupa kesesuaian alamat halaman, keberadaan teks atau elemen, kondisi elemen, maupun perubahan nilai yang terjadi setelah aktivitas pengguna dijalankan.

Apabila seluruh *assertion* pada suatu *test case* terpenuhi dalam batas waktu yang telah ditentukan, Playwright memberikan status *pass*. Sebaliknya, apabila terdapat *assertion* yang tidak terpenuhi, Playwright memberikan status *fail* dan menampilkan informasi mengenai bagian pemeriksaan yang mengalami kegagalan.

Ketika terjadi kegagalan, konfigurasi Playwright memungkinkan penyimpanan bukti pendukung berupa *screenshot*, video, dan *trace*, sesuai kondisi konfigurasi yang telah ditetapkan. Selain itu, laporan HTML dan *log* pengujian dapat digunakan untuk membantu mengetahui tahapan serta informasi kegagalan yang terjadi selama pelaksanaan pengujian.

3. Implementasi Workflow GitHub Actions

Otomatisasi pengujian fungsional diintegrasikan dengan GitHub Actions melalui berkas konfigurasi *playwright.yml* yang disimpan pada direktori *.github/workflows*. *Workflow* tersebut digunakan untuk menyiapkan lingkungan aplikasi dan pengujian, menjalankan aplikasi, mengeksekusi seluruh *test case* Playwright menggunakan peramban Chromium, serta menyimpan laporan dan data pendukung hasil pengujian.

Workflow dikonfigurasi agar berjalan secara otomatis ketika terjadi aktivitas *push* ke *branch* main atau *pull request* yang ditujukan ke *branch* main. Setiap aktivitas tersebut memicu GitHub Actions untuk menjalankan satu *job* pengujian pada lingkungan *ubuntu-latest*. Konfigurasi tersebut memungkinkan perubahan kode diperiksa melalui pengujian otomatis sebelum digabungkan melalui *pull request* maupun setelah perubahan dikirimkan ke repositori utama.

Proses implementasi *workflow* meliputi penyiapan lingkungan eksekusi, konfigurasi variabel lingkungan, pemasangan dependensi aplikasi dan pengujian, penyiapan basis data MySQL, menjalankan server aplikasi, pelaksanaan pengujian Playwright, serta penyimpanan laporan hasil pengujian sebagai *artifact*.

a. Pemicu dan Lingkungan Workflow

Bagian awal berkas *playwright.yml* digunakan untuk menentukan nama *workflow*, aktivitas pemicu, hak akses repositori, konfigurasi *job*, lingkungan *runner*, batas waktu pelaksanaan, serta layanan basis data yang digunakan selama pengujian. *Workflow* diberi nama Pengujian Fungsional ShopFlow, sedangkan *job* pengujian diberi nama Playwright Chromium.

Workflow dikonfigurasi untuk berjalan ketika terjadi aktivitas *push* ke *branch* main atau *pull request* yang ditujukan ke *branch* main. *Job* dijalankan menggunakan *ubuntu-latest*

sebagai *runner* dengan batas waktu maksimal 20 menit. Hak akses repositori ditetapkan sebagai `contents: read` agar *workflow* memperoleh izin untuk membaca kode sumber yang diperlukan selama proses pengujian. Selain itu, layanan MySQL 8.4 digunakan sebagai basis data pada lingkungan GitHub Actions. Rincian konfigurasi pemicu dan lingkungan GitHub Actions yang digunakan dalam penelitian ditunjukkan pada Tabel 7.

Tabel 7. Rincian konfigurasi pemicu dan lingkungan GitHub Actions yang digunakan dalam penelitian

No.	Komponen	Nilai	Fungsi
1	Nama workflow	Pengujian Fungsional ShopFlow	Memberikan identitas pada proses otomatisasi pengujian
2	Pemicu pertama	push ke branch main	Menjalankan workflow ketika perubahan kode dikirimkan ke branch utama
3	Pemicu kedua	pull_request ke branch main	Menjalankan workflow ketika terdapat pull request yang ditujukan ke branch utama
4	Hak akses	<code>contents: read</code>	Memberikan izin kepada workflow untuk membaca isi repositori
5	Nama job	Playwright Chromium	Memberikan identitas pada job pengujian
6	Runner	ubuntu-latest	Menyediakan lingkungan Linux untuk menjalankan proses pengujian
7	Batas waktu	20 menit	Menghentikan job apabila pelaksanaan melebihi batas waktu yang ditentukan
8	Layanan basis data	MySQL 8.4	Menyediakan basis data selama proses pengujian berlangsung

Berdasarkan Tabel 7, GitHub Actions menggunakan ubuntu-latest sebagai lingkungan eksekusi dan MySQL 8.4 sebagai layanan basis data selama proses pengujian. Layanan MySQL dikonfigurasi agar dapat digunakan oleh aplikasi selama *workflow* berlangsung. Pemeriksaan kondisi (*health check*) diterapkan untuk memastikan bahwa layanan MySQL telah siap menerima koneksi sebelum tahapan pengujian berikutnya dijalankan.

Penggunaan aktivitas push dan pull_request pada *branch* main memungkinkan pengujian fungsional dijalankan secara otomatis ketika perubahan kode dikirimkan ke repositori utama maupun ketika perubahan akan digabungkan melalui *pull request*. Dengan demikian, proses pengujian dapat menjadi bagian dari pelaksanaan *Continuous Integration* pada repositori ShopFlow. Konfigurasi pemicu dan lingkungan *workflow* GitHub Actions yang digunakan dalam penelitian ditunjukkan pada Gambar 8.

```

1 name: Pengujian Fungsional ShopFlow
2
3 on:
4   push:
5     branches:
6       - main
7   pull_request:
8     branches:
9       - main
10
11 permissions:
12   contents: read
13
14 jobs:
15   playwright:
16     name: Playwright Chromium
17     runs-on: ubuntu-latest
18     timeout-minutes: 20
19
20     services:
21       mysql:
22         image: mysql:8.4
23         env:
24           MYSQL_ROOT_PASSWORD: root
25           MYSQL_DATABASE: shopflow
26         ports:
27           - 3306:3306
28         options: --health-cmd="mysqladmin ping -h localhost -proot"
29               --health-interval=10s
30               --health-timeout=5s
31               --health-retries=3

```

Gambar 8. Konfigurasi Pemicu dan Lingkungan GitHub Actions

Gambar 8 menunjukkan konfigurasi pemicu dan lingkungan GitHub Actions yang digunakan dalam penelitian. Konfigurasi tersebut mengatur aktivitas *push* atau *pull request* pada *branch* main, penggunaan ubuntu-latest sebagai *runner*, batas waktu pelaksanaan *job*, hak akses repositori, serta layanan MySQL 8.4 yang digunakan selama proses pengujian.

b. Konfigurasi Variabel Lingkungan

Variabel lingkungan digunakan untuk menyimpan konfigurasi yang diperlukan selama pelaksanaan *workflow* GitHub Actions, meliputi konfigurasi basis data, alamat dasar aplikasi, serta data yang digunakan dalam pengujian. Konfigurasi basis data terdiri atas alamat *host*, nomor *port*, nama basis data, nama pengguna, dan kata sandi. Sementara itu, SHOPFLOW_BASE_URL digunakan untuk menentukan alamat dasar aplikasi yang dijalankan pada *runner* GitHub Actions.

Data pengujian juga disediakan melalui variabel lingkungan agar skrip Playwright dapat menggunakan data yang konsisten selama proses pengujian. Data tersebut meliputi akun pelanggan utama, akun pelanggan tanpa alamat pengiriman, kata sandi akun, produk yang digunakan pada pengujian keranjang dan *checkout*, produk yang tersedia, serta kata kunci produk yang tidak tersedia. Rincian variabel lingkungan yang digunakan pada GitHub Actions ditunjukkan pada Tabel 8.

Tabel 8. variabel lingkungan yang digunakan pada GitHub Actions

No.	Variabel	Fungsi
1	CI	Menandai bahwa pengujian dijalankan pada lingkungan Continuous Integration
2	DB_HOST	Menentukan alamat layanan MySQL
3	DB_PORT	Menentukan nomor port layanan MySQL
4	DB_NAME	Menentukan nama basis data pengujian
5	DB_USER	Menentukan nama pengguna basis data
6	DB_PASS	Menentukan kata sandi basis data
7	SHOPFLOW_BASE_URL	Menentukan alamat dasar aplikasi pada runnerGitHub Actions
8	SHOPFLOW_TEST_EMAIL	Menentukan email akun pelanggan utama yang digunakan dalam pengujian
9	SHOPFLOW_TEST_PASSWORD	Menentukan kata sandi akun pelanggan utama
10	SHOPFLOW_TEST_SIMPLE_PRODUCT	Menentukan produk yang digunakan pada pengujiankeranjang dan checkout
11	SHOPFLOW_TEST_PRODUCT_AVAILABLE	Menentukan produk yang tersedia untuk pengujian TC-04
12	SHOPFLOW_TEST_PRODUCT_UNAVAILABLE	Menentukan kata kunci produk yang tidak tersedia untuk pengujian TC-05
13	SHOPFLOW_TEST_NO_ADDRESS_EMAIL	Menentukan email akun pelanggan tanpa alamat pengiriman untuk TC-09
14	SHOPFLOW_TEST_NO_ADDRESS_PASSWORD	Menentukan kata sandi akun pelanggan tanpa alamat pengiriman

Berdasarkan Tabel 8, variabel CI digunakan untuk menandai bahwa proses pengujian dijalankan pada lingkungan *Continuous Integration*. Variabel DB_HOST, DB_PORT, DB_NAME, DB_USER, dan DB_PASS digunakan untuk menghubungkan aplikasi dengan layanan MySQL yang berjalan pada *runner* GitHub Actions.

Variabel SHOPFLOW_BASE_URL digunakan untuk menentukan alamat dasar aplikasi yang dijalankan pada lingkungan GitHub Actions. Dengan menggunakan variabel tersebut, skrip Playwright dapat mengakses aplikasi melalui alamat yang sama selama proses *workflow* tanpa menuliskan alamat lengkap secara berulang pada setiap *test case*.

Variabel SHOPFLOW_TEST_EMAIL dan SHOPFLOW_TEST_PASSWORD digunakan sebagai data autentikasi akun pelanggan utama pada pengujian fitur *login*, keranjang belanja, dan *checkout* dengan data lengkap. Sementara itu, SHOPFLOW_TEST_NO_ADDRESS_EMAIL dan

SHOPFLOW_TEST_NO_ADDRESS_PASSWORD digunakan sebagai data autentikasi akun khusus untuk TC-09. Akun tersebut tidak memiliki alamat pengiriman sehingga dapat digunakan untuk memeriksa respons aplikasi ketika proses *checkout* dilakukan tanpa alamat pengiriman.

Variabel produk digunakan untuk menjaga konsistensi data uji pada setiap pelaksanaan *workflow*. SHOPFLOW_TEST_SIMPLE_PRODUCT digunakan pada pengujian keranjang belanja dan *checkout*, SHOPFLOW_TEST_PRODUCT_AVAILABLE digunakan pada pengujian pencarian produk yang tersedia, sedangkan SHOPFLOW_TEST_PRODUCT_UNAVAILABLE digunakan pada pengujian pencarian produk yang tidak tersedia.

Kata sandi akun pelanggan tanpa alamat pengiriman tidak dituliskan secara langsung pada berkas *workflow*, tetapi disimpan menggunakan fasilitas *repository secret* GitHub. Nilai tersebut kemudian digunakan melalui variabel SHOPFLOW_TEST_NO_ADDRESS_PASSWORD agar data autentikasi tidak ditampilkan secara terbuka pada repositori.

c. Penyiapan Lingkungan Pengujian

Setelah kode sumber tersedia pada *runner*, *workflow* menyiapkan lingkungan yang diperlukan untuk menjalankan aplikasi dan pengujian otomatis. Penyiapan tersebut meliputi PHP versi 8.4 beserta ekstensi yang dibutuhkan, Composer, Node.js versi 22, dependensi PHP dan Node.js, Chromium beserta dependensi sistemnya, klien MySQL, serta basis data khusus pengujian.

Selain tahap penyiapan lingkungan, *workflow* juga mencakup proses menjalankan server aplikasi, mengeksekusi pengujian menggunakan Playwright, dan menyimpan laporan hasil pengujian sebagai *artifact*. Tahapan implementasi *workflow* GitHub Actions secara keseluruhan ditunjukkan pada Tabel 9.

Tabel 9. Tahapan implementasi *workflow* GitHub Actions secara keseluruhan

No.	Tahapan	Implementasi	Fungsi
1	Mengambil kode sumber	actions/checkout@v6	Mengambil aplikasi, konfigurasi, dan skrip pengujian dari repositori
2	Menyiapkan PHP	shivammathur/setup-php@v2	Menyediakan PHP versi 8.4, ekstensi PHP, dan Composer
3	Menyiapkan Node.js	actions/setup-node@v6	Menyediakan Node.js versi 22 untuk menjalankan Playwright
4	Memasang dependensi PHP	composer install	Memasang dependensi PHP yang diperlukan aplikasi
5	Memasang dependensi Node.js	npm ci	Memasang Playwright dan dependensi pengujian sesuai package-lock.json
6	Memasang Chromium	npx playwright install --with-deps chromium	Menyediakan Chromium dan dependensi sistem yang diperlukan
7	Memasang klien MySQL	apt-get install mysql-client	Menyediakan perintah untuk mengakses layanan MySQL
8	Mengimpor basis data	database/shopflow_ci.sql	Menyiapkan struktur dan data sintetis untuk pengujian
9	Menjalankan server aplikasi	PHP built-in server	Menyediakan aplikasi pada alamat 127.0.0.1:8000
10	Menjalankan pengujian	npx playwright test	Mengeksekusi seluruh test case menggunakan Playwright
11	Mengunggah laporan	actions/upload-artifact@v7	Menyimpan laporan Playwright dan log server sebagai artifact

Berdasarkan Tabel 9, kode sumber aplikasi, konfigurasi Playwright, dan skrip pengujian diambil dari repositori menggunakan actions/checkout@v6. Selanjutnya, PHP versi 8.4 disiapkan menggunakan shivammathur/setup-php@v2 bersama ekstensi pdo_mysql, mbstring, gd, dan zip, serta Composer yang digunakan untuk mengelola dependensi PHP.

Dependensi PHP dipasang menggunakan perintah `composer install --no-interaction --prefer-dist --no-progress`. Node.js versi 22 kemudian disiapkan menggunakan `actions/setup-node@v6`, sedangkan dependensi Node.js dipasang menggunakan perintah `npm ci`. Penggunaan `npm ci` memungkinkan dependensi dipasang berdasarkan versi yang tercatat pada berkas `package-lock.json` sehingga lingkungan pengujian dapat disiapkan secara konsisten.

Chromium beserta dependensi sistemnya dipasang menggunakan perintah `npx playwright install --with-deps chromium`. Klien MySQL juga dipasang agar *runner* dapat berinteraksi dengan layanan basis data MySQL yang digunakan selama pengujian.

Basis data pengujian diimpor dari berkas `database/shopflow_ci.sql`. Berkas tersebut memuat struktur tabel dan data sintetis yang diperlukan selama pengujian, seperti akun pelanggan utama, akun pelanggan tanpa alamat pengiriman, alamat, kategori, produk, dan data pendukung transaksi. Penggunaan basis data khusus pengujian dilakukan agar pelaksanaan *workflow* tidak bergantung pada basis data lokal maupun data pelanggan sebenarnya.

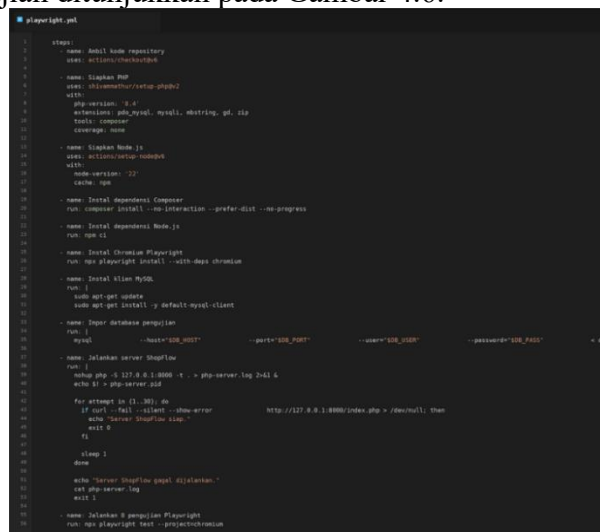
Setelah lingkungan dan basis data siap, aplikasi dijalankan menggunakan PHP *built-in server* dan seluruh *test case* dieksekusi menggunakan Playwright. Setelah pengujian selesai, laporan dan data pendukung diunggah sebagai *artifact* menggunakan `actions/upload-artifact@v7`.

d. Menjalankan Server dan Pengujian

Setelah seluruh dependensi dan basis data selesai disiapkan, aplikasi dijalankan menggunakan PHP *built-in server* pada alamat <http://127.0.0.1:8000/>. Proses server dijalankan pada latar belakang, sedangkan informasi pelaksanaannya disimpan pada berkas `php-server.log`.

Sebelum Playwright menjalankan pengujian, *workflow* melakukan pemeriksaan kesiapan aplikasi menggunakan perintah `curl`. Pemeriksaan dilakukan maksimal 30 kali dengan jeda satu detik pada setiap percobaan untuk mengakses halaman `index.php`. Apabila aplikasi berhasil diakses, *workflow* melanjutkan proses pengujian. Sebaliknya, apabila aplikasi belum dapat diakses setelah seluruh percobaan selesai, isi berkas `php-server.log` ditampilkan untuk membantu mengetahui penyebab kegagalan dan *workflow* dihentikan dengan status gagal.

Setelah aplikasi dinyatakan siap, seluruh *test case* dijalankan menggunakan perintah: `npx playwright test --project=chromium`. Perintah tersebut mengeksekusi sepuluh *test case* menggunakan proyek Chromium yang telah dikonfigurasi pada Playwright. Hasil pelaksanaan pengujian ditampilkan pada *log* GitHub Actions dalam bentuk status *passed*, *failed*, atau *skipped*, serta informasi kesalahan apabila terdapat pengujian yang mengalami kegagalan. Tahapan pelaksanaan *workflow* GitHub Actions mulai dari penyiapan lingkungan hingga eksekusi pengujian ditunjukkan pada Gambar 4.6.



```

name: playwright.yml

on:
  push:
    branches:
      - main

jobs:
  setup:
    name: Setup environment
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Setup PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.1'
          extensions: pdo_mysql, mbstring, gd, zip
          tools: composer
          coverage: none

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '22'
          cache: npm

      - name: Install dependencies Composer
        run: composer install --no-interaction --prefer-dist --no-progress

      - name: Install dependencies Node.js
        run: npm ci

      - name: Install Chromium Playwright
        run: npx playwright install --with-deps chromium

      - name: Install client MySQL
        run: |
          sudo apt-get update
          sudo apt-get install -y default-mysql-client

      - name: Import database pengujian
        run: |
          mysql -h127.0.0.1 -uroot --password=12345678 -e "CREATE DATABASE IF NOT EXISTS shopflow_ci;"

      - name: Jalankan server Shopflow
        run: |
          php -S 127.0.0.1:8000 & > php-server.log 2>&1 &
          echo $! > php-server.pid

      - name: Cek status server
        run: |
          for attempt in {1..30}; do
            if curl -s -o /dev/null -w "%{http_code}" http://127.0.0.1:8000/index.php > /dev/null; then
              echo "Server Shopflow siap"
              exit 0
            fi
            sleep 1
          done

      - name: Jalankan 5 pengujian Playwright
        run: npx playwright test --project=chromium
  
```

Gambar 9. Tahapan Pelaksanaan Workflow GitHub Actions

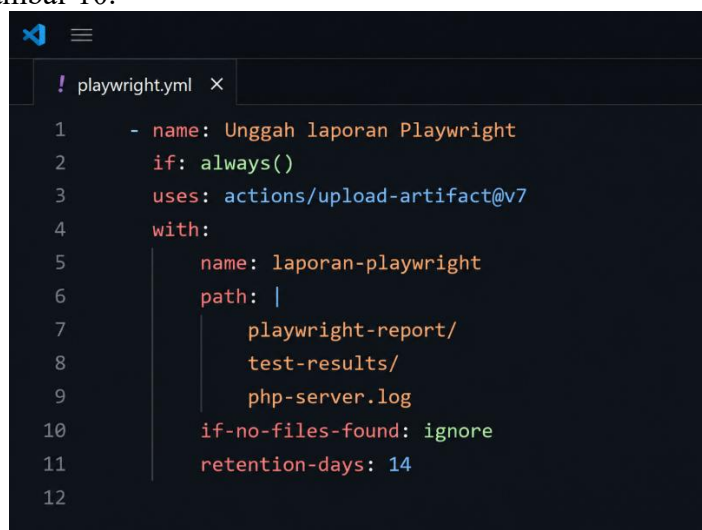
Gambar 9 menunjukkan tahapan pelaksanaan *workflow* GitHub Actions yang digunakan dalam otomatisasi pengujian fungsional. Proses dimulai dari pengambilan kode sumber, penyiapan PHP dan Node.js, pemasangan dependensi, pemasangan Chromium, penyiapan dan impor basis data pengujian, menjalankan server aplikasi, pemeriksaan kesiapan aplikasi, hingga pelaksanaan seluruh *test case* menggunakan Playwright.

e. Penyimpanan Laporan Pengujian

Setelah seluruh *test case* selesai dijalankan, laporan dan data pendukung hasil pengujian disimpan sebagai *artifact* menggunakan `actions/upload-artifact@v7`. Proses pengunggahan dikonfigurasi menggunakan kondisi `always()` agar laporan tetap diproses baik ketika pengujian berhasil maupun ketika terdapat kegagalan.

Data yang disimpan meliputi direktori `playwright-report`, direktori `test-results`, dan berkas `php-server.log`. Direktori `playwright-report` berisi laporan HTML hasil pengujian Playwright, sedangkan direktori `test-results` digunakan untuk menyimpan bukti pendukung pengujian seperti *screenshot*, video, dan *trace* sesuai kondisi yang dihasilkan selama pengujian. Berkas `php-server.log` digunakan untuk menyimpan informasi pelaksanaan server aplikasi pada lingkungan GitHub Actions.

Artifact diberi nama `laporan-playwright` dan disimpan selama 14 hari. Apabila salah satu berkas atau direktori yang ditentukan tidak tersedia, konfigurasi `if-no-files-found: ignore` digunakan agar proses pengunggahan tidak menyebabkan *workflow* gagal hanya karena berkas tersebut tidak dihasilkan. Konfigurasi penyimpanan laporan hasil pengujian sebagai *artifact* ditunjukkan pada Gambar 10.



```

1  - name: Unggah laporan Playwright
2    if: always()
3    uses: actions/upload-artifact@v7
4    with:
5      name: laporan-playwright
6      path: |
7        playwright-report/
8        test-results/
9        php-server.log
10     if-no-files-found: ignore
11     retention-days: 14
12

```

Gambar 10. Konfigurasi Penyimpanan Laporan sebagai Artifact

Gambar 10 menunjukkan konfigurasi penyimpanan laporan hasil pengujian menggunakan `actions/upload-artifact@v7`. Konfigurasi tersebut menyimpan `playwright-report`, `test-results`, dan `php-server.log` dalam *artifact* bernama `laporan-playwright` dengan masa penyimpanan selama 14 hari. Penggunaan kondisi `always()` memungkinkan proses penyimpanan laporan tetap dijalankan setelah tahap pengujian selesai sehingga hasil dan bukti pendukung pengujian dapat diperiksa kembali melalui GitHub Actions.

f. Hasil Pengujian Fungsional Manual

Pengujian fungsional secara manual dilakukan terhadap sepuluh *test case* pada fitur *login*, pencarian produk, keranjang belanja, dan *checkout*. Pengujian dilakukan secara langsung melalui antarmuka aplikasi berdasarkan skenario dan data uji yang telah ditentukan. Hasil aktual dari setiap *test case* dibandingkan dengan hasil yang diharapkan untuk menentukan status keberhasilan pengujian.

Waktu pelaksanaan dicatat untuk setiap *test case* sejak langkah pengujian dimulai hingga hasil akhir diperoleh. *Test case* dinyatakan *pass* apabila hasil aktual sesuai dengan hasil

yang diharapkan dan dinyatakan *fail* apabila terdapat kondisi yang tidak sesuai. Hasil pengujian fungsional secara manual ditunjukkan pada Tabel 10.

Tabel 10. Hasil pengujian fungsional secara manual

No.	ID	Skenario Pengujian	Hasil Yang Diharapkan	Hasil Aktual	Status	Waktu
1	TC-01	Login menggunakan akun valid	Pengguna berhasil masuk dan diarahkan ke halaman utama	Pengguna berhasil masuk, diarahkan ke halaman utama, pesan keberhasilan ditampilkan, dan menu keluar tersedia	Pass	16 detik
2	TC-02	Login menggunakan kata sandi tidak valid	Sistem menolak login dan menampilkan pesan kesalahan	Sistem menolak proses login, pengguna tetap berada pada halaman login, dan pesan kesalahan ditampilkan	Pass	15 detik
3	TC-03	Login menggunakan format email tidak valid	Sistem menolak login dan menampilkan pesan validasi format email	Sistem menolak proses login, dan pesan "Masukkan alamat email yang valid." ditampilkan	Pass	18 detik
4	TC-04	Mencari produk yang tersedia	Produk yang sesuai ditampilkan	Produk yang sesuai dengan kata kunci berhasil ditampilkan pada halaman hasil pencarian	Pass	6 detik
5	TC-05	Mencari produk yang tidak tersedia	Sistem menampilkan informasi bahwa produk tidak ditemukan	Sistem menampilkan jumlah hasil nol dan informasi bahwa produk tidak ditemukan	Pass	5 detik
6	TC-06	Menambahkan produk ke keranjang	Produk ditampilkan pada halaman keranjang	Produk berhasil ditambahkan dan ditampilkan pada halaman keranjang belanja	Pass	4 detik
7	TC-07	Mengubah jumlah produk dari satu menjadi dua	Jumlah produk, total harga, dan subtotal diperbarui	Jumlah produk berhasil diubah menjadi dua serta total harga dan subtotal keranjang diperbarui	Pass	3 detik
8	TC-08	Checkout dengan data lengkap	Pesanan berhasil dibuat dan halaman konfirmasi ditampilkan	Proses checkout berhasil, pesanan dibuat, dan halaman konfirmasi pesanan ditampilkan	Pass	45 detik
9	TC-09	Checkout tanpa alamat pengiriman	Pesanan tidak dapat dibuat dan informasi alamat pengiriman tidak ditampilkan	Sistem menampilkan informasi bahwa alamat pengiriman belum tersedia, tidak terdapat alamat yang dipilih, tombol pembuatan pesanan tidak dapat digunakan, dan pesanan tidak dibuat	Pass	24 detik
10	TC-10	Checkout dengan keranjang kosong	Sistem mencegah checkout dan menampilkan informasi bahwa keranjang masih kosong	Sistem mencegah proses checkout dan menampilkan pesan "Keranjang Anda masih kosong."	Pass	21 detik

Berdasarkan Tabel 10, seluruh sepuluh *test case* pengujian fungsional manual memperoleh status *pass*. Hasil tersebut menunjukkan bahwa respons aplikasi pada fitur *login*, pencarian produk, keranjang belanja, dan *checkout* sesuai dengan hasil yang diharapkan pada skenario yang diuji. Total waktu yang diperlukan untuk menjalankan seluruh pengujian manual adalah 157 detik atau 2 menit 37 detik.

TC-08 memiliki waktu pelaksanaan paling lama, yaitu 45 detik, karena mencakup rangkaian proses *login*, pencarian produk, penambahan produk ke keranjang, pemilihan alamat pengiriman, metode pembayaran, layanan pengiriman, hingga pembuatan pesanan. Sementara itu, TC-07 memiliki waktu pelaksanaan paling singkat, yaitu 3 detik, karena pengujian berfokus pada perubahan jumlah produk serta pemeriksaan perubahan total harga dan subtotal pada halaman keranjang.

g. Hasil Pengujian Menggunakan Playwright

Pengujian fungsional otomatis dilakukan menggunakan Playwright terhadap sepuluh *test case* yang sama dengan pengujian manual. Seluruh pengujian dijalankan menggunakan

proyek Chromium berdasarkan konfigurasi pada berkas `playwright.config.ts`. Playwright menjalankan setiap langkah pengujian secara otomatis dan menentukan hasil berdasarkan *assertion* yang terdapat pada masing-masing skrip.

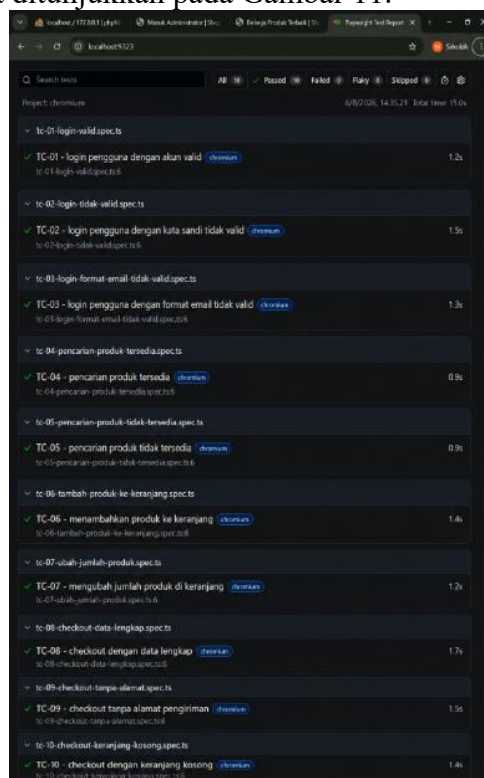
Hasil pengujian otomatis digunakan untuk mengetahui status keberhasilan setiap *test case* pada fitur *login*, pencarian produk, keranjang belanja, dan *checkout*. Hasil pengujian otomatis menggunakan Playwright ditunjukkan pada Tabel 11.

Tabel 11. Hasil pengujian otomatis menggunakan Playwright

No.	ID	Skenario Pengujian	Status
1	TC-01	Login menggunakan akun valid	Pass
2	TC-02	Login menggunakan kata sandi tidak valid	Pass
3	TC-03	Login menggunakan format email tidak valid	Pass
4	TC-04	Mencari produk yang tersedia	Pass
5	TC-05	Mencari produk yang tidak tersedia	Pass
6	TC-06	Menambahkan produk ke keranjang	Pass
7	TC-07	Mengubah jumlah produk dari satu menjadi dua	Pass
8	TC-08	Checkout dengan data lengkap	Pass
9	TC-09	Checkout tanpa alamat pengiriman	Pass
10	TC-10	Checkout dengan keranjang kosong	Pass

Berdasarkan Tabel 11, seluruh sepuluh *test case* berhasil dijalankan menggunakan Playwright dengan status *pass*. Hasil tersebut menunjukkan bahwa seluruh kondisi pemeriksaan yang didefinisikan melalui *assertion* dapat dipenuhi oleh respons aplikasi pada skenario yang diuji.

Laporan HTML Playwright memberikan informasi tambahan mengenai jumlah pengujian yang berhasil, gagal, *flaky*, maupun *skipped*, serta total waktu pelaksanaan seluruh pengujian. Berdasarkan laporan tersebut, sepuluh *test case* memperoleh status *passed*, sedangkan jumlah *failed*, *flaky*, dan *skipped* masing-masing sebanyak nol. Total waktu yang diperlukan Playwright untuk menjalankan seluruh pengujian adalah 15,0 detik. Laporan hasil pengujian otomatis tersebut ditunjukkan pada Gambar 11.



Gambar 11. Laporan Hasil Pengujian Otomatis Menggunakan Playwright

Gambar 11 menunjukkan laporan hasil pengujian otomatis menggunakan Playwright pada peramban Chromium. Laporan tersebut memperlihatkan bahwa seluruh sepuluh *test case* berhasil memperoleh status *passed*, dengan jumlah *failed*, *flaky*, dan *skipped* masing-masing sebanyak nol. Total waktu yang diperlukan untuk menjalankan seluruh pengujian otomatis adalah 15,0 detik.

h. Hasil Eksekusi GitHub Actions

Pengujian fungsional otomatis selanjutnya dijalankan melalui GitHub Actions sebagai bagian dari proses *Continuous Integration*. *Workflow* dikonfigurasi untuk berjalan secara otomatis ketika terjadi aktivitas *push* ke *branch* main atau *pull request* yang ditujukan ke *branch* main. Pada proses tersebut, GitHub Actions menyiapkan lingkungan aplikasi dan pengujian, menjalankan aplikasi serta basis data, kemudian mengeksekusi seluruh *test case* Playwright menggunakan peramban Chromium.

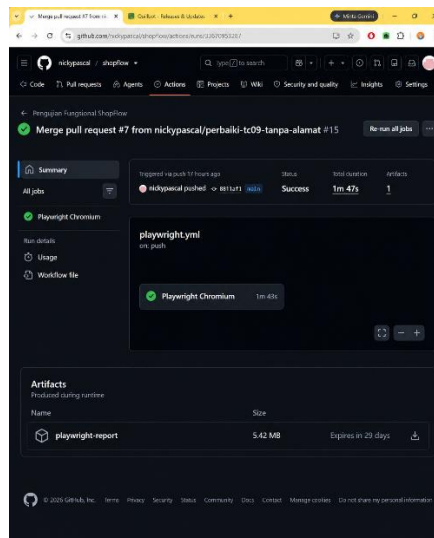
Hasil eksekusi menunjukkan bahwa *workflow* Pengujian Fungsional ShopFlow berhasil dijalankan pada *runner* GitHub Actions. *Job* Playwright Chromium memperoleh status berhasil (*success*) dan seluruh sepuluh *test case* dapat diselesaikan tanpa kegagalan. Hasil tersebut menunjukkan bahwa pengujian otomatis dapat dijalankan pada lingkungan *Continuous Integration* sesuai dengan konfigurasi *workflow* yang telah diterapkan. Ringkasan hasil eksekusi GitHub Actions ditunjukkan pada Tabel 12.

Tabel 12. Ringkasan hasil eksekusi GitHub Actions

No.	Komponen	Hasil
1	Workflow	Pengujian Fungsional ShopFlow
2	Job	Playwright Chromium
3	Runner	ubuntu-latest
4	Peramban pengujian	Chromium
5	Jumlah test case	10
6	Passed	10
7	Failed	0
8	Skipped	0
9	Status workflow	Success
10	Laporan hasil	laporan-playwright

Berdasarkan Tabel 12, seluruh sepuluh *test case* berhasil dijalankan pada lingkungan GitHub Actions dengan jumlah *passed* sebanyak 10, *failed* sebanyak 0, dan *skipped* sebanyak 0. Status *Success* pada *workflow* menunjukkan bahwa rangkaian proses yang diperlukan untuk menjalankan pengujian otomatis dapat diselesaikan dengan baik pada lingkungan *Continuous Integration*.

Setelah proses pengujian selesai, GitHub Actions menghasilkan *artifact* bernama laporan-playwright. *Artifact* tersebut digunakan untuk menyimpan laporan Playwright dan data pendukung hasil pengujian sehingga hasil eksekusi dapat diperiksa kembali setelah *workflow* selesai dijalankan. Hasil eksekusi *workflow* dan *artifact* yang dihasilkan ditunjukkan pada Gambar 4.9.



Gambar 12. Hasil Eksekusi Workflow GitHub Actions

Gambar 12 menunjukkan hasil eksekusi *workflow* GitHub Actions pada *branch* main. *Workflow* *playwright.yml* berhasil dijalankan dengan status *Success* dan *job* Playwright Chromium memperoleh status berhasil. Seluruh sepuluh *test case* berhasil dijalankan dengan jumlah *passed* sebanyak 10, *failed* sebanyak 0, dan *skipped* sebanyak 0. Setelah proses pengujian selesai, GitHub Actions juga menghasilkan *artifact* bernama laporan-playwright yang digunakan untuk menyimpan laporan dan data pendukung hasil pengujian. Hasil tersebut menunjukkan bahwa otomatisasi pengujian fungsional menggunakan Playwright telah berhasil diintegrasikan dan dijalankan pada proses *Continuous Integration*. Kode program dan konfigurasi otomatisasi pengujian yang digunakan dalam penelitian ini dapat dilihat melalui <https://github.com/nickypascal/shopflow>.

4. Perbandingan Pengujian Manual dan Otomatis

Setelah pengujian fungsional secara manual dan otomatis selesai dilakukan, hasil kedua metode dibandingkan berdasarkan jumlah *test case*, hasil pengujian, waktu pelaksanaan, mekanisme eksekusi, pengulangan pengujian, penentuan hasil, dokumentasi, serta integrasi dengan *Continuous Integration*. Pengujian manual dan otomatis menggunakan sepuluh skenario yang sama sehingga hasil keduanya dapat dibandingkan berdasarkan kondisi pengujian yang setara.

Pengujian manual membutuhkan waktu keseluruhan 157 detik atau 2 menit 37 detik, sedangkan pengujian otomatis menggunakan Playwright membutuhkan waktu 15,0 detik untuk menjalankan seluruh sepuluh *test case*. Seluruh *test case* pada kedua metode memperoleh status *pass*. Perbandingan hasil pengujian manual dan otomatis ditunjukkan pada Tabel 13.

Tabel 13. Perbandingan hasil pengujian manual dan otomatis

No.	Aspek Perbandingan	Pengujian Manual	Pengujian Otomatis
1	Jumlah test case	10 test case	10 test case
2	Hasil pengujian	10 pass, 0 fail	10 passed, 0 failed
3	Waktu pelaksanaan	157 detik	15,0 detik
4	Pelaksanaan pengujian	Dilakukan langsung melalui antarmuka aplikasi oleh penguji	Dijalankan secara otomatis melalui skrip Playwright
5	Interaksi	Setiap langkah dilakukan secara manual oleh penguji	Interaksi dilakukan secara otomatis oleh Playwright
6	Pengulangan pengujian	Setiap langkah harus dilakukan kembali secara manual	Skrip yang sama dapat dijalankan kembali
7	Penentuan hasil	Berdasarkan pengamatan terhadap respons aplikasi	Berdasarkan assertion pada skrip pengujian

8	Dokumentasi hasil	Dicatat berdasarkan hasil pelaksanaan pengujian	Menghasilkan laporan HTML, test-results, dan log pengujian
9	Integrasi dengan CI	Tidak terintegrasi secara otomatis	Terintegrasi dengan GitHub Actions
10	Pemicu pengujian	Dijalankan ketika penguji melakukan pengujian	Dapat dijalankan otomatis ketika terjadi push ke branch main atau pull request yang ditujukan ke branch main

Berdasarkan Tabel 13, pengujian manual dan pengujian otomatis menghasilkan status yang sama pada seluruh sepuluh *test case*, yaitu berhasil atau *pass*. Kesamaan hasil tersebut menunjukkan bahwa skrip Playwright mampu menjalankan skenario pengujian fungsional yang sama dengan skenario yang dilakukan secara manual.

Perbedaan utama terdapat pada waktu dan mekanisme pelaksanaan pengujian. Pengujian manual membutuhkan waktu 157 detik untuk menyelesaikan seluruh sepuluh *test case*, sedangkan pengujian otomatis menggunakan Playwright membutuhkan waktu 15,0 detik. Dengan demikian, terdapat selisih waktu sebesar 142 detik, dengan waktu pelaksanaan pengujian otomatis sekitar 90,4% lebih singkat dibandingkan pengujian manual pada skenario yang digunakan dalam penelitian.

Pada pengujian manual, penguji menjalankan setiap langkah secara langsung melalui antarmuka aplikasi dan menentukan hasil berdasarkan respons yang diamati. Sementara itu, Playwright menjalankan aktivitas pengguna dan pemeriksaan hasil secara otomatis berdasarkan skrip serta *assertion* yang telah ditentukan.

Dari sisi pengulangan, pengujian manual mengharuskan setiap langkah kembali ketika pengujian dilakukan ulang, sedangkan pengujian otomatis dapat mengeksekusi kembali skrip yang sama. Selain itu, Playwright menghasilkan laporan pengujian secara terstruktur dan telah diintegrasikan dengan GitHub Actions sehingga pengujian dapat dijalankan sebagai bagian dari *Continuous Integration* ketika terjadi perubahan kode pada repositori.

Hasil pengujian menunjukkan bahwa seluruh sepuluh *test case* pada fitur *login*, pencarian produk, keranjang belanja, dan *checkout* memperoleh status *pass* pada pengujian manual maupun pengujian otomatis menggunakan Playwright. Kesamaan hasil tersebut menunjukkan bahwa skrip pengujian otomatis mampu menjalankan skenario dan pemeriksaan fungsi aplikasi sesuai dengan kondisi yang ditetapkan pada pengujian manual.

Pada fitur *login*, TC-01 menunjukkan bahwa aplikasi dapat menerima data autentikasi yang valid sehingga pengguna berhasil masuk dan diarahkan ke halaman utama. TC-02 menunjukkan bahwa aplikasi dapat menolak proses *login* ketika email akun yang valid digunakan dengan kata sandi yang tidak sesuai. Pengguna tetap berada pada halaman *login* dan pesan kesalahan ditampilkan. Sementara itu, TC-03 menunjukkan bahwa aplikasi dapat menolak proses *login* ketika pengguna memasukkan format email yang tidak valid. Sistem tetap menampilkan halaman *login* dan memberikan pesan validasi bahwa alamat email yang dimasukkan tidak valid. Hasil tersebut menunjukkan bahwa fungsi autentikasi memberikan respons yang sesuai terhadap kondisi *login* valid, kata sandi tidak valid, dan format email tidak valid.

Pada fitur pencarian produk, TC-04 menunjukkan bahwa aplikasi dapat menampilkan produk yang sesuai dengan kata kunci ketika produk tersedia. TC-05 menunjukkan bahwa aplikasi menampilkan jumlah hasil nol dan informasi bahwa produk tidak ditemukan ketika kata kunci tidak memiliki kecocokan dengan data produk. Hasil tersebut menunjukkan bahwa fungsi pencarian memberikan respons yang sesuai pada kondisi produk tersedia maupun tidak tersedia.

Pada fitur keranjang belanja, TC-06 menunjukkan bahwa produk yang tersedia dapat ditambahkan dan ditampilkan pada halaman keranjang. TC-07 menunjukkan bahwa perubahan jumlah produk dari satu menjadi dua menyebabkan jumlah produk, total harga, dan subtotal

keranjang diperbarui sesuai dengan jumlah terbaru. Hasil tersebut menunjukkan bahwa fungsi keranjang belanja mampu menjalankan proses penambahan dan perubahan jumlah produk sesuai dengan skenario yang diuji.

Pada fitur *checkout*, TC-08 menunjukkan bahwa proses pembuatan pesanan dapat diselesaikan ketika data yang diperlukan tersedia, termasuk alamat pengiriman, metode pembayaran, dan layanan pengiriman. Sistem membuat pesanan dan menampilkan halaman konfirmasi. Sebaliknya, TC-09 menunjukkan bahwa sistem mencegah proses pembuatan pesanan ketika akun belum memiliki alamat pengiriman. Informasi bahwa alamat pengiriman belum tersedia ditampilkan, sedangkan metode pembayaran, layanan pengiriman, dan tombol pembuatan pesanan tidak dapat digunakan. Sementara itu, TC-10 menunjukkan bahwa sistem mencegah proses *checkout* ketika keranjang belanja dalam kondisi kosong dan menampilkan informasi bahwa keranjang masih kosong.

Berdasarkan keseluruhan hasil tersebut, empat fitur yang menjadi ruang lingkup penelitian memberikan respons sesuai dengan hasil yang diharapkan pada sepuluh skenario yang diuji. Hasil pengujian otomatis juga konsisten dengan pengujian manual karena seluruh *test case* menghasilkan status *pass* pada kedua metode. Namun, kesimpulan tersebut terbatas pada fitur dan skenario yang diuji dalam penelitian sehingga tidak digunakan untuk menyatakan bahwa seluruh fungsi aplikasi telah bebas dari kesalahan.

Hasil perbandingan menunjukkan bahwa pengujian manual dan pengujian otomatis menggunakan Playwright menghasilkan status yang sama pada seluruh sepuluh *test case*, yaitu *pass*. Kesamaan hasil tersebut menunjukkan bahwa skrip pengujian otomatis mampu menjalankan skenario dan pemeriksaan fungsi aplikasi yang setara dengan pengujian manual pada fitur *login*, pencarian produk, keranjang belanja, dan *checkout*.

Perbedaan utama antara kedua metode terdapat pada waktu dan mekanisme pelaksanaan pengujian. Pengujian manual membutuhkan waktu keseluruhan 157 detik atau 2 menit 37 detik, sedangkan pengujian otomatis menggunakan Playwright membutuhkan waktu 15,0 detik untuk menjalankan sepuluh *test case*. Dengan demikian, terdapat selisih waktu sebesar 142 detik, sehingga waktu pelaksanaan pengujian otomatis sekitar 90,4% lebih singkat dibandingkan pengujian manual pada skenario yang digunakan dalam penelitian ini.

Pada pengujian manual, setiap langkah dilakukan secara langsung oleh penguji melalui antarmuka aplikasi, mulai dari memasukkan data, menekan tombol, berpindah halaman, hingga mengamati respons aplikasi. Kondisi tersebut menyebabkan waktu pelaksanaan dipengaruhi oleh kecepatan dan ketelitian penguji. Sebaliknya, pada pengujian otomatis, seluruh langkah telah didefinisikan dalam skrip Playwright dan dijalankan secara otomatis melalui peramban Chromium. Pemeriksaan hasil dilakukan menggunakan *assertion* sehingga status setiap *test case* dapat ditentukan berdasarkan kondisi yang telah ditetapkan.

Perbedaan juga terlihat pada proses pengulangan pengujian. Pengujian manual mengharuskan penguji menjalankan kembali setiap langkah ketika pengujian perlu dilakukan ulang. Pada pengujian otomatis, skrip yang sama dapat dijalankan kembali tanpa mengulangi seluruh interaksi secara manual. Hal tersebut membuat urutan langkah dan kondisi pemeriksaan lebih konsisten pada setiap pelaksanaan pengujian.

Meskipun pengujian otomatis menunjukkan waktu pelaksanaan yang lebih singkat pada penelitian ini, pengujian manual tetap memiliki peran dalam memeriksa alur penggunaan aplikasi secara langsung serta membantu memastikan skenario dan hasil yang diharapkan sebelum skrip otomatis diterapkan. Dengan demikian, otomatisasi pengujian tidak menggantikan seluruh fungsi pengujian manual, tetapi dapat digunakan untuk mempercepat dan mempermudah pelaksanaan pengujian berulang terhadap skenario yang telah ditentukan.

Implementasi Playwright pada proses *Continuous Integration* dilakukan melalui GitHub Actions dengan menggunakan *workflow* playwright.yml. Integrasi tersebut memungkinkan pengujian fungsional dijalankan secara otomatis ketika terjadi aktivitas *push*

ke *branch* main atau *pull request* yang ditujukan ke *branch* main. Dengan mekanisme tersebut, pengujian tidak hanya dapat dijalankan secara lokal, tetapi juga dapat dieksekusi pada lingkungan *runner* GitHub Actions.

Proses *workflow* mencakup pengambilan kode sumber, penyiapan PHP dan Node.js, pemasangan dependensi, penyiapan Chromium, penyediaan layanan MySQL, impor basis data pengujian, menjalankan server aplikasi, pemeriksaan kesiapan aplikasi, serta pelaksanaan sepuluh *test case* menggunakan Playwright. Hasil eksekusi menunjukkan bahwa seluruh tahapan tersebut dapat diselesaikan dan *job* Playwright Chromium memperoleh status *Success*.

Keberhasilan *workflow* menunjukkan bahwa skrip Playwright dapat dijalankan pada lingkungan *Continuous Integration* dengan konfigurasi yang telah diterapkan. Seluruh sepuluh *test case* memperoleh status *passed* pada lingkungan GitHub Actions. Hasil tersebut juga menunjukkan bahwa basis data pengujian, aplikasi, peramban Chromium, dan skrip pengujian dapat digunakan secara terintegrasi dalam satu rangkaian proses otomatis.

Integrasi dengan GitHub Actions juga memberikan mekanisme pelaksanaan pengujian yang konsisten karena tahapan pengujian didefinisikan dalam berkas *workflow*. Ketika kondisi pemicu terpenuhi, GitHub Actions menjalankan rangkaian proses sesuai konfigurasi yang sama. Dengan demikian, proses pengujian tidak bergantung sepenuhnya pada pelaksanaan manual melalui lingkungan lokal.

Selain status keberhasilan pengujian, implementasi tersebut menghasilkan *log* eksekusi dan *artifact* laporan-playwright yang berisi laporan Playwright serta data pendukung pengujian. Keberadaan *log* dan *artifact* memudahkan pemeriksaan kembali terhadap hasil pengujian dan membantu proses identifikasi ketika terdapat tahap *workflow* atau *test case* yang mengalami kegagalan.

Berdasarkan hasil tersebut, Playwright berhasil diintegrasikan dengan GitHub Actions untuk menjalankan otomatisasi pengujian fungsional pada proses *Continuous Integration*. Implementasi ini memungkinkan sepuluh skenario pengujian yang telah ditentukan dijalankan secara otomatis pada lingkungan CI serta menghasilkan informasi status, *log*, dan laporan pengujian sebagai bukti hasil eksekusi.

Berdasarkan hasil implementasi dan pengujian, penerapan otomatisasi pengujian fungsional menggunakan Playwright yang terintegrasi dengan GitHub Actions telah mencapai tujuan penelitian yang ditetapkan. Ketercapaian tersebut ditunjukkan melalui keberhasilan pelaksanaan sepuluh *test case*, kesesuaian hasil pengujian otomatis dengan pengujian manual, perbedaan waktu pelaksanaan pengujian, serta keberhasilan *workflow* GitHub Actions menjalankan proses pengujian pada lingkungan *Continuous Integration*.

Dari sisi implementasi, Playwright berhasil digunakan untuk mengotomatisasi pengujian pada fitur *login*, pencarian produk, keranjang belanja, dan *checkout*. Fitur *login* dan *checkout* masing-masing memiliki tiga skenario pengujian, sedangkan fitur pencarian produk dan keranjang belanja masing-masing memiliki dua skenario pengujian sehingga terdapat sepuluh *test case* yang diperiksa menggunakan *assertion*. Seluruh *test case* memperoleh status *passed*, yang menunjukkan bahwa respons aplikasi memenuhi kondisi yang ditetapkan pada seluruh skenario yang diuji.

Dari sisi hasil pengujian, pengujian manual dan otomatis memberikan hasil yang konsisten karena seluruh sepuluh *test case* memperoleh status berhasil pada kedua metode. Pengujian manual membutuhkan waktu keseluruhan 157 detik atau 2 menit 37 detik, sedangkan Playwright membutuhkan waktu 15,0 detik untuk menjalankan seluruh skenario. Dengan demikian, pengujian otomatis memiliki selisih waktu 142 detik atau sekitar 90,4% lebih singkat dibandingkan pengujian manual pada skenario dan kondisi pengujian dalam penelitian ini.

Dari sisi integrasi *Continuous Integration*, GitHub Actions berhasil menjalankan rangkaian *workflow* mulai dari penyiapan lingkungan, penyediaan basis data, menjalankan aplikasi, hingga mengeksekusi seluruh *test case* Playwright menggunakan Chromium. Hasil

eksekusi menunjukkan *workflow* memperoleh status *Success*. Selain status pengujian, GitHub Actions menyediakan *log* setiap tahapan eksekusi dan menghasilkan *artifact* laporan-playwright yang menyimpan laporan serta data pendukung hasil pengujian. Dengan demikian, hasil pengujian dapat diperiksa kembali setelah proses *workflow* selesai.

Keberhasilan tersebut menunjukkan bahwa penerapan Playwright dan GitHub Actions dapat membentuk proses pengujian fungsional otomatis yang dapat dijalankan kembali dengan skenario dan pemeriksaan yang telah ditentukan. Integrasi tersebut juga memungkinkan pengujian dijalankan sebagai bagian dari proses *Continuous Integration* ketika terjadi aktivitas *push* ke *branch* main atau *pull request* yang ditujukan ke *branch* main. Namun, hasil ketercapaian penelitian ini terbatas pada sepuluh *test case*, empat fitur aplikasi, peramban Chromium, dan lingkungan pengujian yang digunakan dalam penelitian.

E. KESIMPULAN

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, kesimpulan penelitian ini adalah sebagai berikut: Otomatisasi pengujian fungsional berhasil diimplementasikan menggunakan Playwright yang terintegrasi dengan GitHub Actions pada proses *Continuous Integration* aplikasi *e-commerce*. Pengujian diterapkan terhadap sepuluh *test case* yang mencakup fitur *login*, pencarian produk, keranjang belanja, dan *checkout*. Fitur *login* dan *checkout* masing-masing diuji menggunakan tiga skenario, sedangkan fitur pencarian produk dan keranjang belanja masing-masing diuji menggunakan dua skenario. Playwright digunakan untuk menjalankan aktivitas pengujian dan melakukan pemeriksaan hasil menggunakan *assertion*, sedangkan GitHub Actions menjalankan *workflow* pengujian secara otomatis ketika terjadi aktivitas *push* ke *branch* main atau *pull request* yang ditujukan ke *branch* main. *Workflow* tersebut mampu menyiapkan lingkungan pengujian, menjalankan aplikasi, mengeksekusi pengujian menggunakan Chromium, serta menyimpan laporan hasil pengujian dalam bentuk *artifact*. Dengan demikian, otomatisasi pengujian fungsional berhasil diterapkan sebagai bagian dari proses *Continuous Integration*.

Hasil penerapan otomatisasi pengujian menunjukkan bahwa seluruh sepuluh *test case* berhasil dijalankan dengan status *passed*, sedangkan jumlah *failed* dan *skipped* masing-masing sebanyak 0. Hasil tersebut konsisten dengan pengujian manual yang seluruhnya memperoleh status *pass*. Pengujian manual membutuhkan waktu keseluruhan 157 detik atau 2 menit 37 detik, sedangkan Playwright membutuhkan waktu 15,0 detik untuk menjalankan seluruh sepuluh *test case*. Terdapat selisih waktu sebesar 142 detik, sehingga waktu pelaksanaan pengujian otomatis sekitar 90,4% lebih singkat dibandingkan pengujian manual pada skenario yang digunakan dalam penelitian. Selain itu, GitHub Actions memperoleh status *Success* serta menghasilkan *log* dan *artifact* laporan-playwright yang dapat digunakan untuk memeriksa kembali hasil pengujian. Berdasarkan hasil tersebut, penerapan otomatisasi mampu menjalankan skenario pengujian secara lebih cepat, konsisten, dapat diulang, serta terintegrasi dengan proses *Continuous Integration*.

DAFTAR PUSTAKA

- Abrar, M. F., Alharbi, Y., Alsaffar, M., Hussain, S., Saqib, M., Khan, J., & Lee, Y. (2025). Developing SPIM-TA: a maturity-level framework for systematic process improvement in software testing automation. *Ain Shams Engineering Journal*, 16(8). <https://doi.org/10.1016/j.asej.2025.103472>
- Almabruk, S., Abdalhamid, S., Almabruk, T., & Academy, T. L. (2026). *Investigating Automated Software Testing in Terms of Efficiency : A Comparative Study of Selenium and Playwright*. 9(4), 1053–1066.
- Amanda, R. (2024). Plant Based Watering System Internet of Things Arduino and Monitoring

- with Telegram. *Bit-Tech*, 6(3), 295–302. <https://doi.org/10.32877/bt.v6i3.1094>
- Arujisaputra, E. T., Heriansyah, M., Zamani, F. E., & Husain, A. (2025). UI/UX Design Of Website-Based E-Commerce Application Using Lean User Experience Method. *CoreID Journal*, 3(1), 23–36. <https://doi.org/10.60005/coreid.v3i1.92>
- Balsam, S., & Mishra, D. (2025). Web application testing—Challenges and opportunities. *Journal of Systems and Software*, 219. <https://doi.org/10.1016/j.jss.2024.112186>
- Bauer, A., Frattini, J., & Alégroth, E. (2024). Augmented testing to support manual GUI-based regression testing: An empirical study. *Empirical Software Engineering*, 29(6). <https://doi.org/10.1007/s10664-024-10522-z>
- Delicheh, H. O., Cardoen, G., & Mens, T. O. M. (2026). *Automation and Reuse Practices in GitHub Actions Workflows : A Practitioner ' s Perspective*.
- Faqih, A. R., Taufiqurrahman, A., Husen, J. H., & Sabariah, M. K. (2024). Empirical Analysis of CI/CD Tools Usage in GitHub Actions Workflows. *Journal of Informatics and Web Engineering*, 3(2), 251. <https://doi.org/10.33093/jiwe.2024.3.2.18>
- Gosik, P., & Miłosz, M. (2025). Comparative analysis of Cypress and Playwright frameworks in end-to-end testing for applications based on Angular. *Journal of Computer Sciences Institute*, 36, 320–327. <https://doi.org/10.35784/jcsi.7662>
- Hutahaeen, H. A., Ruwaida, A., Adnyani, N. L. S. S., Govindaraju, R., & Sudirman, I. (2024). Incorporating Functional Quality into Usability Model of E-Commerce Application. *TEM Journal*, 13(3), 1889–1904. <https://doi.org/10.18421/TEM133-17>
- Leotta, M., Ricca, F., Marchetto, A., & Olinas, D. (2024). An empirical study to compare three web test automation approaches: NLP-based, programmable, and capture&replay. *Journal of Software: Evolution and Process*, 36(5). <https://doi.org/10.1002/smr.2606>
- Melyawati, N. L. P., Asana, I. M. D. P., Putri, N. N. S., Atmaja, K. J., & Sudipa, I. G. I. (2024). Comparison of Automation Testing On Card Printer Project Using Playwright And Selenium Tools. *Journal of Computer Networks, Architecture and High Performance Computing*, 6(3), 1309–1320. <https://doi.org/10.47709/cnahpc.v6i3.4362>
- Moń, M., & Pańczyk, B. (2025). A comparative analysis of web application test automation tools. *Journal of Computer Sciences Institute*, 35, 159–165. <https://doi.org/10.35784/jcsi.7119>
- Pasławski, P., & Pańczyk, M. (2024). Comparison of selected tools for automation testing of Web applications. *Journal of Computer Sciences Institute*, 31, 145–150. <https://doi.org/10.35784/jcsi.6238>
- , G. A., & Setiani, N. (2025). *Perbandingan kinerja framework automation ui testing menggunakan metode the distance to the ideal alternative 1*. 10(1), 224–237.
- Raghithwetsagul, C., Krinke, J., Choetkiertikul, M., Sunetnanta, T., & Sarro, F. (2024). Adoption of automated software engineering tools and techniques in Thailand. *Empirical Software Engineering*, 29(4). <https://doi.org/10.1007/s10664-024-10472-6>
- Rezeki, A., Saputro, S. W., & Saragih, T. H. (2026). *Empirical Performance of E2E Frameworks in React-Vue SPAs Using DIA*. 7(1), 317–333. <https://doi.org/10.59395/ijadis.v7i1.1528>
- Ricca, F., Marchetto, A., & Stocco, A. (2025). A multi-year grey literature review on AI-assisted test automation. *Information and Software Technology*, 186. <https://doi.org/10.1016/j.infsof.2025.107799>
- Riza, F., Berliyanto, B., Nurrohmah, A., & Setiabudi, R. (2024). Comparative analysis of automation functional testing tools performance for Playstore apps with DIA method. *Jurnal Techno Nusa Mandiri*, 21(1), 9-14.
- Rostami, P., Decan, A., Mens, T., & Wessel, M. (2026). *An Empirical Study of the Evolution*. 1–40.
- Santos, J., da Costa, D. A., McIntosh, S., & Kulesza, U. (2025). On the need to monitor

- continuous integration practices. *Empirical Software Engineering*, 30(5). <https://doi.org/10.1007/s10664-025-10682-6>
- Soares, E., Sizilio, G., Santos, J., da Costa, D. A., & Kulesza, U. (2022). The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering*, 27(3). <https://doi.org/10.1007/s10664-021-10114-1>
- Tymoshchuk, A. (2025). The Evolution of Test Automation: From Selenium to Playwright. A Comparison of Automation Tools: Selenium vs. Playwright vs. Cypress. *International Journal of Computer (IJC) International Journal of Computer*, 54(1), 37–45. <https://ijcjournal.org/index.php/InternationalJournalOfComputer/index>
- Wang, Y., Mäntylä, M. V, Liu, Z., & Markkula, J. (2022). Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration. *Journal of Systems and Software*, 188. <https://doi.org/10.1016/j.jss.2022.111259>
- Wessel, M., Vargovich, J., Gerosa, M. A., & Treude, C. (2023). GitHub Actions: The Impact on the Pull Request Process. *Empirical Software Engineering*, 28(6). <https://doi.org/10.1007/s10664-023-10369-w>
- Zheng, L., Li, S., Huang, X., Huang, J., Lin, B., Chen, J., & Xuan, J. (2026). Why Do GitHub Actions Workflows Fail? An Empirical Study. *ACM Transactions on Software Engineering and Methodology*, 35(5). <https://doi.org/10.1145/3749371>