SINTA Jurnal Sistem Informasi dan Teknologi Komputasi
Volume 2 Nomor 2 April 2025 [page: 66-75]
DOI: https://doi.org/10.61124/sinta.v2i2.42

PENGEMBANGAN APPLICATION PROGRAMMING INTERFACE (API) APLIKASI PENDETEKSI PENYAKIT TANAMAN MENGGUNAKAN METODE DEVOPS

Bagus Syafiq Faqihuddin[1], Acep Taryana[1], Mulki Indana Zulfa[1]
[1]Program Studi Teknik Elektro, Universitas Jenderal Soedirman
e-mail: bagus.faqihuddin@mhs.unsoed.ac.id

Abstrak
Pengembangan backend Application Programming Interface (API) menjadi elemen penting dalam sistem pendeteksi penyakit tanaman berbasis kecerdasan buatan. Namun, pengembangan API sering menghadapi kendala dalam integrasi, deployment, dan monitoring yang menghambat kestabilan sistem. Penelitian ini menerapkan metode DevOps secara menyeluruh untuk membangun backend API yang efisien, modular, dan berkelanjutan. Empat pilar utama DevOps yang diterapkan meliputi continuous development, integration, deployment, dan monitoring, menggunakan GitHub Actions sebagai pipeline otomasi dan Docker sebagai sistem containerisasi di server DigitalOcean. Hasil pengujian menunjukkan bahwa dari 23 kali deployment, 22 di antaranya berhasil, dengan waktu eksekusi rata-rata 42–152 detik. Proses monitoring menunjukkan performa API yang stabil, meskipun beberapa fitur seperti manajemen pengguna dan riwayat memerlukan optimasi lebih lanjut. Penelitian ini membuktikan bahwa praktik DevOps dapat diterapkan secara efektif bahkan oleh pengembang individu, dan dapat meningkatkan kualitas serta kecepatan pengembangan API.

Kata kunci: Application Programming Interface ; backend; DevOps; continuous.

Abstract
The development of backend Application Programming Interfaces (APIs) is a crucial component of AI-based plant disease detection systems. However, API development often faces challenges in integration, deployment, and monitoring that hinder system stability. This study implements a comprehensive DevOps method to build an efficient, modular, and sustainable backend API. The DevOps pillars applied include continuous development, integration, deployment, and monitoring, using GitHub Actions for automation and Docker as the containerization platform on DigitalOcean servers. The testing results show that 22 out of 23 deployments were successful, with average execution times ranging from 42 to 152 seconds. Monitoring results indicate that the APIs perform reliably, although some features such as user management and history require further optimization. This research demonstrates that DevOps practices can be effectively applied even by solo developers, enhancing both the quality and speed of backend API development.

Keywords: Application Programming Interface; backend; DevOps; continuous

1. PENDAHULUAN

Indonesia merupakan negara agraris dengan jumlah rumah tangga petani yang besar, namun masih menghadapi risiko gagal panen akibat serangan penyakit tanaman [1]. Salah satu solusi untuk mengatasi permasalahan ini adalah penerapan teknologi berbasis kecerdasan buatan dalam bentuk aplikasi mobile yang mampu mendeteksi penyakit tanaman secara otomatis [2]. Aplikasi seperti ini memerlukan komponen backend yang handal dalam bentuk Application Programming Interface (API), yang bertugas menangani permintaan dari client, mengelola data, dan menyambungkan ke model prediksi [3] [4]. Salah satu arsitektur yang populer digunakan yaitu RESTful API yaitu gaya arsitektur yang menggunakan HTTP method serta mengandalkan prinsip *resource-oriented* yang dapat diakses melalui URL tertentu dengan format pertukaran data berupa JSON [4].

Pengembangan API modern memerlukan stabilitas, efisiensi, dan kemampuan deployment berkelanjutan menjadi sangat krusial. Pendekatan DevOps menjadi solusi karena mengintegrasikan proses pengembangan (*development*) dan operasional (*operations*), serta memungkinkan otomatisasi pengujian, deployment, dan pemantauan sistem [2] [5] [6]. DevOps bekerja dalam empat siklus utama, yaitu *continuous development*, *continuous integration*, *continuous deployment*, dan *continuous monitoring* [7], [8]. Pada DevOps seringkali membutuhkan teknologi *cloud computing* untuk memungkinkan dapat mengakses sumber daya komputasi seperti penyimpanan data,server , dan perangkat lunak melalui internet, *cloud computing* digunakan karena kemudahannya dalam akses serta sumber daya dikelola secara terpusat sehingga memudahkan untuk dilakukan *monitoring* dan mudah dalam melakukan *scalable server* [9][10].

Beberapa penelitian menunjukkan keberhasilan penerapan DevOps dalam berbagai jenis aplikasi. Nurhayati dan Agussalim [5] menerapkan DevOps dalam pembangunan backend API berbasis Node.js

DOI: https://doi.org/10.61124/sinta.v2i2.42

Express untuk aplikasi AyamHub, menghasilkan sistem yang lebih modular dan cepat dideploy. Studi oleh Paramudita et al. [8] menunjukkan bahwa penerapan pipeline CI/CD dalam pengembangan aplikasi pendeteksi kualitas beras berbasis machine learning dapat mengurangi waktu integrasi dan risiko error. Zulfikar [11] bahkan menyoroti pentingnya penggunaan Docker dan Kubernetes pipeline dalam pengembangan sistem berbasis MLOps, yang relevan untuk integrasi API dan model prediksi.

Studi lain oleh Falah dan Komarudin [12] menggunakan kombinasi Node.js dan Python untuk mengembangkan REST API berbasis microservices di Google Cloud Platform, mencakup hasil performa deployment yang stabil. Penelitian Taryana et al. [13] juga membuktikan efektivitas DevOps dalam meningkatkan jaminan mutu internal sistem pendidikan tinggi melalui otomasi QA system. Selain itu, Kurniawan et al. [14] membangun aplikasi mobile berbasis Android menggunakan pendekatan DevOps, menekankan pentingnya otomatisasi pada siklus pengembangan sistem informasi masyarakat.

Meskipun riset-riset tersebut memberikan kontribusi penting dalam pengembangan sistem berbasis DevOps, sebagian besar dilakukan oleh tim pengembang dengan sumber daya besar. Penerapan DevOps oleh pengembang individu dalam pengembangan backend API untuk sistem deteksi penyakit tanaman belum banyak dibahas. Celah ini membuka peluang untuk penelitian, khususnya dalam penerapan DevOps pada skala kecil hingga menengah. Tantangan utama yang dihadapi mencakup pengolahan data, pengujian kode, deployment cepat, dan monitoring sistem. Penelitian ini fokus pada pengembangan backend API yang efisien dan berkelanjutan menggunakan penerapan DevOps secara menyeluruh untuk mengisi gap tersebut. Selain itu, meskipun banyak penelitian yang berfokus pada aplikasi mobile, belum ada yang secara khusus membahas penerapan DevOps untuk pengembangan backend API dalam sistem deteksi penyakit tanaman yang berbasis AI. Kurniawan et al. [14] dan Paramudita et al.[15] lebih fokus pada aplikasi mobile, sedangkan penelitian ini berfokus pada backend API yang menjadi tulang punggung aplikasi berbasis mobile, yang sangat penting untuk memastikan kestabilan dan akurasi prediksi berbasis machine learning.

Pengembangan RESTful API penelitian ini menggunakan 2 bahasa pemrograman berbeda yaitu Node JS dan Flask. Node JS merupakan runtime enviroment berbasis *even-driven* dan *non-blocking* I/O yang ideal untuk aplikasi sederhana serta populer sehingga memudahkan melakukan pencarian dokumentasi [16]. Sementara itu Flask merupakan *framework* berbasis Python yang ideal untuk membangun API untuk model machine learning yang membutuhkan *server-side* secara efisien [17]. Dalam pengembangan RESTful API juga membutuhkan manajemen database untuk melakukan penyimpanan data salah satu database yang populer digunakan yaitu MySQL karena kestabilannya, skalabilitasnya serta dukungan terhadap *query* SQL [18] [19].

Dokumentasi dan kolaborasi menggunakan Git dan GitHub sebagai alat manajemen versi serta workflow GitHub Actions menjadi standar umum dalam implementasi pipeline CI/CD [19] [21]. Penggunaan Docker untuk containerization dan DigitalOcean untuk deployment juga umum digunakan dalam skala pengembangan kecil hingga menengah. Sebagian besar studi terdahulu dilakukan oleh tim pengembang, sementara penerapan DevOps oleh pengembang individu masih jarang dikaji secara mendalam. Celah ini menjadi menarik untuk diteliti, khususnya dalam konteks pengembangan backend API untuk sistem deteksi penyakit tanaman, yang membutuhkan sinkronisasi antara pengolahan data, pengujian kode, deployment cepat, dan monitoring sistem.

Berdasarkan hal tersebut, penelitian ini bertujuan untuk menerapkan metode DevOps secara menyeluruh dalam pengembangan backend API, dengan menggunakan GitHub Actions sebagai pipeline otomasi dan Docker sebagai platform deployment ke server cloud (DigitalOcean). Penelitian ini mencakup empat pilar DevOps dan mengevaluasi efektivitas proses dalam mendukung pengembangan sistem yang efisien, modular, dan dapat dipantau secara berkelanjutan. Meskipun fokus utama penelitian ini adalah pada pengembangan backend API yang mendukung aplikasi pendeteksi penyakit tanaman berbasis kecerdasan buatan, penting untuk menekankan relevansi sistem ini dalam sektor pertanian Indonesia. Penyakit tanaman, seperti hama wereng pada padi yang dapat mengurangi hasil panen hingga 30%, menimbulkan kerugian besar bagi petani dan mengancam ketahanan pangan nasional. Sistem deteksi penyakit yang cepat dan akurat sangat dibutuhkan untuk membantu petani dalam mengidentifikasi penyakit lebih dini dan mengambil tindakan pencegahan yang tepat. Oleh karena itu, pengembangan backend API yang efisien, stabil, dan dapat diintegrasikan dengan model prediksi berbasis AI menjadi krusial untuk mendukung aplikasi ini. Penelitian ini bertujuan untuk mengoptimalkan pengembangan backend API yang mendukung aplikasi pendeteksi penyakit tanaman melalui penerapan DevOps, guna membantu petani mengurangi kerugian akibat penyakit tanaman dan meningkatkan produktivitas pertanian di Indonesia.

DOI: https://doi.org/10.61124/sinta.v2i2.42

2. METODE

Penelitian ini menggunakan metode rekayasa perangkat lunak menggunakan pendekatan DevOps dalam pengembangan *Application Programming Interface* (API) untuk sistem pendeteksi penyakit tanaman. Proses dilakukan melalui tahapan yaitu studi literatur, perancangan pipeline DevOps, implementasi berbasis CI/CD, evaluasi, dan pelaporan.

1. Studi Literatur

Tahap awal dilakukan dengan mengkaji referensi seperti jurnal ilmiah, buku, artikel, dan sumber daring terkait DevOps, CI/CD, pengembangan API, dan sistem monitoring. Tujuan utamanya adalah memperoleh pemahaman komprehensif sebagai dasar perancangan metode DevOps yang sesuai untuk proyek ini.

2 Perancangan DevOps

Pipeline CI/CD dirancang menggunakan GitHub Actions untuk otomatisasi proses mulai dari commit, pengujian otomatis, *build image*, hingga *deployment* berbasis tag. Pada tahap pengujian otomatis, Jest digunakan untuk melakukan unit testing pada aplikasi Node.js, sedangkan pytest digunakan untuk pengujian pada aplikasi berbasis Python. ESLint diterapkan untuk melakukan linting pada kode JavaScript guna memastikan kualitas dan konsistensi kode sesuai dengan standar penulisan yang ditetapkan. Deployment dilakukan di server DigitalOcean menggunakan Docker container, sedangkan kontrol layanan melalui workflow YAML. Server yang digunakan memiliki spesifikasi CPU 2 vCPU, RAM 4 GB, storage 80 GB SSD, dan OS Ubuntu 20.04 LTS. Docker versi terbaru digunakan untuk menjalankan container, dan server ini diatur dengan static IP untuk akses publik.

3 Penerapan DevOps

Penerapan dilakukan berdasarkan empat pilar utama DevOps, yaitu:

- **Continuous Development**
  Pengembangan fitur dilakukan secara modular dan bertahap menggunakan strategi feature branching. Setiap fitur dirancang dan diuji secara terpisah untuk menjaga kualitas sebelum digabung ke sistem utama.
- **Continuous Integration**
  Setelah kode ditulis, pipeline secara otomatis menjalankan pengujian unit dan integrasi setiap kali terjadi perubahan kode, yang memudahkan deteksi kesalahan sejak dini.
- **Continuous Deployment**
  Aplikasi yang telah diuji secara otomatis dideploy ke server produksi. Pada tahap ini, GitHub Actions digunakan untuk mengakses server dan menjalankan perintah Docker Compose. Proses ini berlangsung secara berkelanjutan untuk setiap versi yang stabil.
- **Continuous Monitoring**
  Sistem dimonitor secara real-time untuk memantau performa, kestabilan API, dan respon server. Monitoring membantu tim segera mengetahui dan memperbaiki gangguan yang terjadi setelah aplikasi dirilis.

3. HASIL PENELITIAN

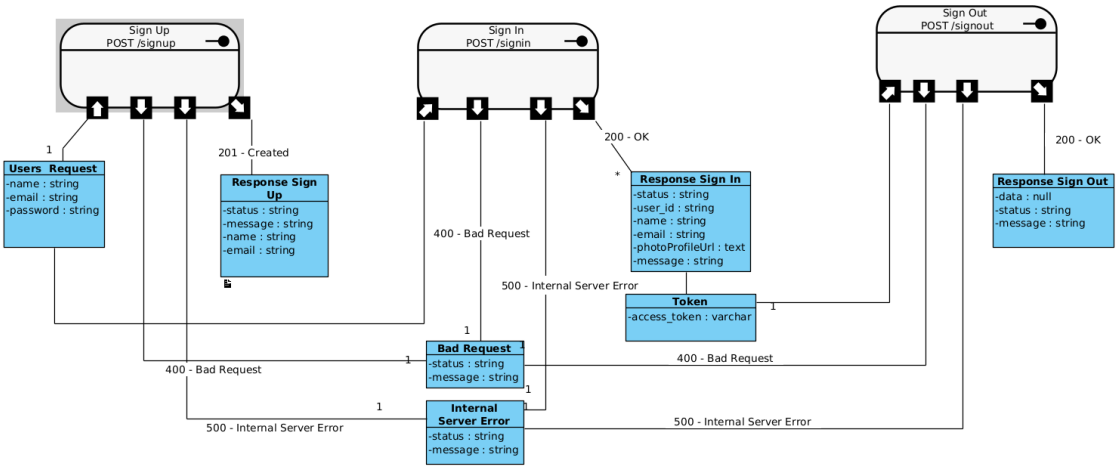
A. Continuous Development

Implementasi metode DevOps dalam pengembangan API pendeteksi penyakit tanaman dilakukan melalui pendekatan *continuous development* yang sistematis dan terukur. Tahap ini menjadi fondasi dalam siklus DevOps karena memastikan bahwa sistem dikembangkan secara modular, terdokumentasi, dan siap diintegrasikan secara otomatis. Tahapan ini memastikan setiap fitur dikembangkan dalam *branch* terpisah, diuji secara terisolasi, dan dapat diintegrasikan ke sistem utama tanpa mengganggu stabilitas aplikasi. Strategi ini diterapkan melalui feature branching di GitHub, disertai dengan proses commit dan dokumentasi yang rapi. Sebagai contoh, pada pengembangan fitur autentikasi, REST Diagram disusun terlebih dahulu untuk merancang alur komunikasi antara klien dan server. Diagram ini menjadi acuan dalam membangun tiga endpoint utama: /signup, /signin, dan /signout. Setiap endpoint diimplementasikan menggunakan arsitektur MVC, yang memisahkan logika di antara controller, model, dan middleware.

Autentikasi menggunakan JWT (JSON Web Token), yang dihasilkan saat pengguna login dan dilampirkan dalam header setiap permintaan berikutnya. *Middleware* khusus memvalidasi token tersebut sebelum permintaan diproses, memastikan hanya pengguna terautentikasi yang dapat mengakses fitur tertentu. Alur ini mencerminkan pendekatan umum yang diterapkan dalam pengembangan fitur lainnya,

DOI: https://doi.org/10.61124/sinta.v2i2.42

dimulai dengan desain REST Diagram, pengembangan pada branch terpisah, serta penerapan struktur MVC



Gambar 1 REST Diagram Fitur Autentikasi

dan middleware untuk keamanan dan pengolahan data..
Contoh hasil dari fitur autentikasi mencakup:

- /signup: Pendaftaran pengguna baru. Jika berhasil, akan mengembalikan pesan sukses dan data pengguna.
- /signin: Autentikasi pengguna. Jika berhasil, akan mengembalikan access token untuk digunakan pada permintaan selanjutnya.
- /signout: Menghapus sesi autentikasi pengguna. Token disimpan dalam daftar blacklist.

Pengembangan selanjutnya dilakukan untuk fitur-fitur berikut:

- Manajemen Pengguna : Fitur manajemen pengguna dirancang untuk mengelola data pengguna dengan aman dan efisien, termasuk mengambil, memperbarui informasi, dan mengganti kata sandi. Pengguna dapat mengubah nama, email, foto profil, dan kata sandi mereka, hanya pengguna terautentikasi yang diperbolehkan untuk melakukan perubahan. Foto profil diunggah ke DigitalOcean Spaces untuk mempermudah pengelolaan gambar.
- Prediksi Penyakit : Fitur prediksi penyakit tanaman dikembangkan menggunakan Model API berbasis Flask, yang memuat dan menjalankan model penyakit tanaman dalam format .h5. Pengguna mengunggah gambar tanaman yang diproses oleh Model API, yang kemudian mengembalikan hasil prediksi dalam format JSON, termasuk informasi penyakit, tingkat kepercayaan model, dan solusi yang disarankan. Hasil prediksi disimpan dalam database untuk fitur riwayat. Pengguna harus terautentikasi melalui Backend API untuk mendapatkan JWT token, yang digunakan untuk mengakses Model API. Autentikasi berbasis JWT memastikan hanya pengguna terautentikasi yang dapat menggunakan fitur prediksi, menjaga keamanan dan kontrol akses.
- Riwayat : Fitur Riwayat menampilkan data yang mencakup informasi seperti jenis tanaman, hasil prediksi, tingkat akurasi, waktu prediksi, serta gambar yang digunakan (jika ada). Fitur ini mengambil data hasil prediksi langsung dari Model API, yang menyimpan hasil prediksi ke dalam database. Pengguna dapat melihat riwayat prediksi sebelumnya, termasuk detail seperti nama tanaman dan solusi yang disarankan, serta memantau akurasi model dari waktu ke waktu.
- Sharing : Fitur Sharing memungkinkan pengguna untuk berbagi informasi tentang penyakit tanaman dalam forum sederhana. Fitur ini tidak mendukung komentar atau fungsi blog penuh. Pengguna dapat mengirimkan informasi atau solusi yang belum tersedia di aplikasi. Gambar yang diunggah disimpan di DigitalOcean Spaces, sementara hanya URL gambar yang disimpan dalam database. Untuk efisiensi tampilan, fitur ini menggunakan pagination untuk membatasi jumlah postingan per halaman, menjaga kinerja aplikasi.
- Setiap fitur dikembangkan pada branch GitHub masing-masing (*auth, users, predict, riwayat, sharing*) untuk menjaga isolasi dan mendukung pengujian terpisah. Proses *commit* dilakukan secara bertahap dengan mencatat perubahan signifikan, seperti penambahan *endpoint*, optimasi *query*, maupun perbaikan validasi. Struktur pengembangan menggunakan pola MVC dan RESTful API, serta basis data MySQL yang telah dirancang menggunakan ERD sebelum pengembangan dimulai. Pendekatan ini memungkinkan pengembangan sistem dilakukan secara cepat, aman, dan fleksibel.

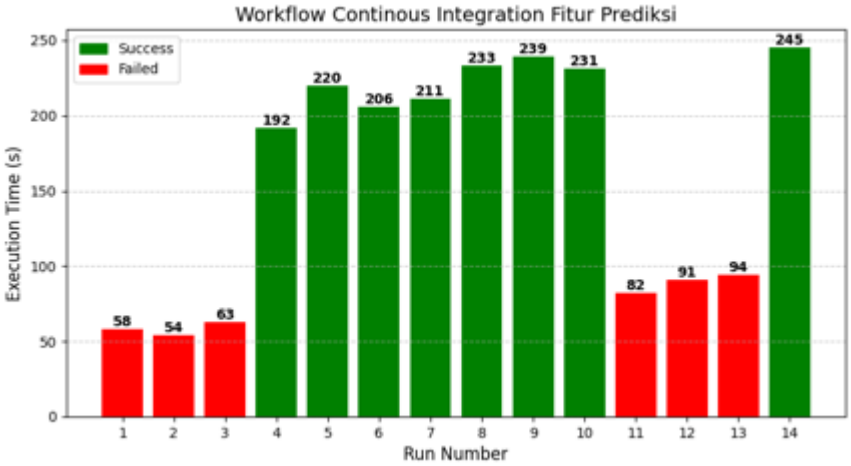
DOI: https://doi.org/10.61124/sinta.v2i2.42

Semua ini menjadi landasan untuk tahap Continuous Integration dan Continuous Deployment yang akan dibahas pada bagian selanjutnya. Implementasi lengkap pengembangan kode pada Continuous Development dapat ditemukan di Repository GitHub melalui tautan berikut:

- **Backend API** : https://bit.ly/BackendAPI
- **Model API** : https://bit.ly/ModelAPI

B. Continuous Integration

Pada tahap *Continuous Integration* (CI), setiap perubahan kode diuji secara otomatis setiap kali ada *pull request* yang diajukan ke branch utama. Tujuan utama dari CI adalah untuk memastikan bahwa setiap perubahan yang diterapkan pada kode tidak merusak kestabilan aplikasi dan bahwa aplikasi tetap dapat berfungsi sebagaimana mestinya. Github Actions digunakan sebagai alat bantu otomatisasi dalam menjalankan proses CI. CI memungkinkan pipeline pengujian otomatis dijalankan untuk setiap fitur utama, yang melibatkan beberapa tahapan penting seperti *linting, unit testing, integration testing,* dan *build*. Hasil pengujian dan integrasi dicatat dan dianalisis untuk melihat seberapa efektif fitur berfungsi setelah setiap perubahan. Sebagai contoh, dapat dilihat pengujian CI pada fitur Prediksi pada Gambar 2, yang memiliki tingkat kompleksitas lebih tinggi dibandingkan dengan fitur lainnya. Fitur ini melibatkan pemrosesan model prediksi berbasis TensorFlow, yang memerlukan waktu komputasi yang lebih lama, sehingga mempengaruhi durasi pengujian dan keberhasilan integrasi fitur tersebut. Fitur Prediksi melakukan deteksi penyakit pada tanaman dengan menggunakan model berbasis TensorFlow. Karena kompleksitasnya yang tinggi, fitur ini memerlukan waktu eksekusi yang lebih lama dan lebih sering mengalami kegagalan pada iterasi awal. Dalam pengujian CI untuk fitur ini, terdapat 14 run yang dilakukan, di mana 6 di antaranya gagal, sementara 8 run lainnya berhasil. Berdasarkan Gambar 2 di atas, dapat dilihat bahwa fitur Prediksi mengalami 6 kegagalan pada tahap pengujian. Setelah dilakukan perbaikan pada konfigurasi dan pengelolaan dependensi, pipeline berhasil menguji dan mengintegrasikan fitur ini dengan baik pada iterasi berikutnya. Waktu eksekusi yang lebih lama pada run yang berhasil (rata-rata 226.50 detik) menggambarkan bahwa pemrosesan model berbasis TensorFlow membutuhkan komputasi intensif. Berikut ini adalah ringkasan hasil pengujian Continuous Integration untuk seluruh fitur utama yang diuji, yang mencakup Autentikasi, Manajemen Pengguna, Prediksi, Riwayat, dan Sharing:



Gambar 2 Hasil Eksekusi Workflow CI Fitur Prediksi

Tabel 1 Ringkasan Hasil Workflow CI.

Fitur	Total Run	Jumlah Gagal	Jumlah Berhasl	Rata-rata Watu Gagal (s)	Rata-rata Watu Berhasl (s)
Auth	8	4	4	40.25	91.25
Users	9	5	4	38.40	73.00
Prediksi	14	6	8	73.67	226.50
Riwayat	4	0	4	-	79.00
Sharing	6	0	6	-	79.50

Tabel 1 menyajikan hasil eksekusi dari berbagai fitur yang diuji dalam sistem. Fitur Riwayat dan Sharing menunjukkan stabilitas yang sangat baik, dengan semua *run* berhasil tanpa kegagalan. Hal ini mencerminkan bahwa kedua fitur tersebut telah melalui pengujian menyeluruh dan tidak mengalami kendala

DOI: https://doi.org/10.61124/sinta.v2i2.42

teknis, sehingga dapat diandalkan untuk digunakan dalam lingkungan produksi tanpa risiko kesalahan. Waktu eksekusi rata-rata sekitar 79 detik menunjukkan efisiensi operasional meskipun tidak ditemukan kegagalan. Sementara itu, fitur Autentikasi, Users, dan Prediksi menunjukkan kinerja yang lebih dinamis, mencerminkan karakteristik fitur yang lebih kompleks atau lebih sering digunakan dalam pengujian.

Pada fitur Autentikasi (8 total run), meskipun 4 run mengalami kegagalan di awal, perbaikan yang dilakukan memungkinkan mayoritas run berikutnya berhasil dijalankan. Waktu eksekusi rata-rata sebesar 91,25 detik untuk run yang berhasil menunjukkan bahwa fitur ini dapat beroperasi secara efisien. Hal serupa terjadi pada fitur Users, yang mengalami 5 kegagalan dari total 9 run. Namun, waktu eksekusi rata-rata sebesar 73 detik pada run yang berhasil mencerminkan bahwa kendala pada pengelolaan pengguna dapat diatasi seiring waktu. Perbaikan yang dilakukan menunjukkan peningkatan stabilitas, menandakan bahwa fitur ini semakin siap untuk digunakan dalam lingkungan produksi. Sementara itu, fitur Prediksi memiliki kompleksitas lebih tinggi karena melibatkan proses komputasi intensif menggunakan model berbasis machine learning. Tercatat 6 kegagalan dari total 14 run, sementara run yang berhasil menunjukkan waktu eksekusi rata-rata sebesar 226,50 detik. Meskipun waktu eksekusinya lebih lama dibandingkan fitur lainnya, hal ini masih dapat diterima mengingat beban proses yang lebih besar dan sifat komputasi yang kompleks. Hal ini menunjukkan bahwa pengujian pada fitur ini memerlukan waktu lebih lama karena beban komputasi yang tinggi, tetapi dengan perbaikan pada konfigurasi dan optimasi kode, hasil yang lebih stabil dapat dicapai. Secara keseluruhan, hasil ini menegaskan bahwa pipeline CI/CD sangat efektif dalam mendeteksi dan mengatasi kesalahan sejak dini. Hal ini memungkinkan pengembang untuk memastikan bahwa kode yang akhirnya dideploy ke DigitalOcean telah melalui serangkaian pengujian yang ketat, sehingga dapat dipastikan siap digunakan di lingkungan produksi. Angka-angka yang tercatat menunjukkan efektivitas pipeline dalam mengidentifikasi masalah, serta pentingnya perbaikan dan optimasi untuk memastikan kualitas kode yang tinggi..

C. Continuous Deployment

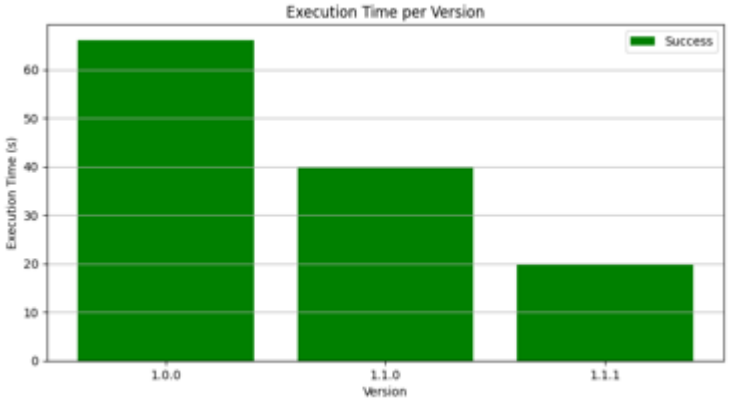
Setelah tahap Continuous Integration (CI) selesai, proses dilanjutkan ke tahap Continuous Deployment (CD). Pada tahap ini, kode yang telah berhasil melewati pengujian otomatis dan pembangunan image akan langsung dideploy ke platform agar dapat digunakan oleh pengguna. Otomatisasi penuh dicapai melalui integrasi GitHub Actions dan Docker Compose, sehingga setiap perubahan yang telah lolos pengujian dapat diterapkan ke sistem utama secara langsung tanpa memerlukan intervensi manual.

Deployment ini dipicu oleh Git tag yang menandai versi tertentu (misalnya 1.0.0). *Workflow* yang diterapkan pada CD memastikan bahwa image Docker yang dihasilkan sesuai dengan kode yang telah diuji dan divalidasi, serta siap untuk di-deploy ke server atau *cloud computing* platform seperti DigitalOcean. Semua langkah ini bertujuan untuk memastikan bahwa aplikasi yang dideploy selalu stabil dan dapat berfungsi dengan baik di lingkungan produksi. Salah satu fitur yang diuji dan dideploy dalam tahap *Continuous Deployment* adalah Fitur Autentikasi. Fitur ini mengalami beberapa iterasi deployment melalui versioning (1.0.0, 1.1.0, 1.1.1). Deployment fitur *Autentikasi* dilakukan dalam tiga tahap versioning. Versi terakhir merupakan versi final yang sepenuhnya lengkap dan siap menjalankan seluruh fungsionalitas API. Setiap versi sudah dilakukan test dan build secara otomatis, yang memastikan bahwa image yang di-build dapat langsung digunakan untuk deployment. Hal ini sangat penting karena dalam *workflow* CI, jika terdapat run yang gagal, maka proses build image akan dihentikan, sehingga tidak dapat melanjutkan ke tahap deployment. Proses otomatisasi ini memastikan kualitas dan konsistensi aplikasi yang dideploy, mengurangi kemungkinan adanya bug atau error yang lolos ke production. Dengan adanya tahapan testing yang dilakukan pada setiap versi, pengembang dapat memastikan bahwa setiap iterasi dari setiap fitur sudah siap digunakan dalam lingkungan produksi. Gambar 3.3 menunjukan eksekusi waktu setiap *deployment,* memberikan gambaran tentang waktu yang dibutuhkan untuk setiap tahapan *deployment*.

Berdasarkan Gambar 3.3 diatas versi 1.0.0, waktu eksekusi deployment tercatat cukup lama, yaitu 66 detik. Hal ini disebabkan oleh ukuran image Docker yang besar akibat penggunaan Node.js versi yang kurang optimal serta manajemen dependensi yang tidak efisien. Setelah dilakukan perbaikan, pada versi 1.1.0, waktu eksekusi berhasil dipangkas menjadi 40 detik berkat penggantian versi Node.js dan pembaruan beberapa dependensi. Pada versi 1.1.1, optimasi lebih lanjut dilakukan, dan waktu eksekusi yang paling efisien tercatat 20 detik, menunjukkan bahwa perbaikan bertahap dapat meningkatkan performa pipeline secara signifikan. Secara total, terdapat 23 eksekusi deployment, 22 berhasil dan hanya 1 kegagalan yang terjadi pada fitur prediksi versi 1.3.2. Kegagalan tersebut disebabkan oleh *broken pipe* yang terjadi akibat beban server yang tinggi saat proses pengambilan image. Hal ini terjadi ketika server tidak dapat menangani permintaan karena kapasitas yang terbatas, sehingga menyebabkan kesalahan pada proses *deployment*. Masalah tersebut berhasil diatasi setelah dilakukan optimasi infrastruktur, termasuk peningkatan kapasitas server dan

DOI: https://doi.org/10.61124/sinta.v2i2.42

perbaikan konfigurasi untuk mengelola beban yang lebih tinggi. Setelah itu, proses *retry deployment* dilakukan dan berhasil dengan lancar, memastikan bahwa *deployment* berjalan dengan baik.



Gambar 3 Histogram Exec Time CD Fitur Autentikasi

Proses ini juga menyoroti pentingnya siklus *Continuous Integration* (CI) dan *Continuous Deployment* (CD) yang sebelumnya dilakukan. Siklus ini meminimalkan potensi error dengan memastikan setiap perubahan atau update pada fitur diuji secara otomatis dan terus-menerus. Tahapan *testing* dan *build* otomatis memungkinkan kesalahan terdeteksi lebih awal, memudahkan tim pengembang untuk segera mengidentifikasi dan memperbaiki masalah sebelum mencapai tahap deployment ke *production*. Hal ini tidak hanya mengurangi kemungkinan error, tetapi juga meningkatkan efisiensi dan reliabilitas dalam siklus pengembangan perangkat lunak. Tabel 2 menyajikan hasil workflow yang berjalan untuk setiap fitur pada fase *continuous deployment*.

Tabel 2 Ringkasan Tabel Hasil Workflow CD.

Fitur	Jumlah Versi	Jumlah Berhasil	Jumlah Gagal	Rata-rata Waktu Eksekusi	Keterangan
Autent.	3	3	-	42	Semua deployment berhasil tanpa kendala
Users	4	4	-	44.25	Waktu deployment konsisten dan stabil
Prediksi	7	6	1	152.71	Terdapat satu kegagalan pada versi 1.3.2, namun berhasil setelah retry
Riwayat	3	3	-	53.67	Semua versi berhasil ter-deploy dengan lancar
Sharing	6	6	-	42.83	Deployment berjalan sukses dan efisien pada semua versi

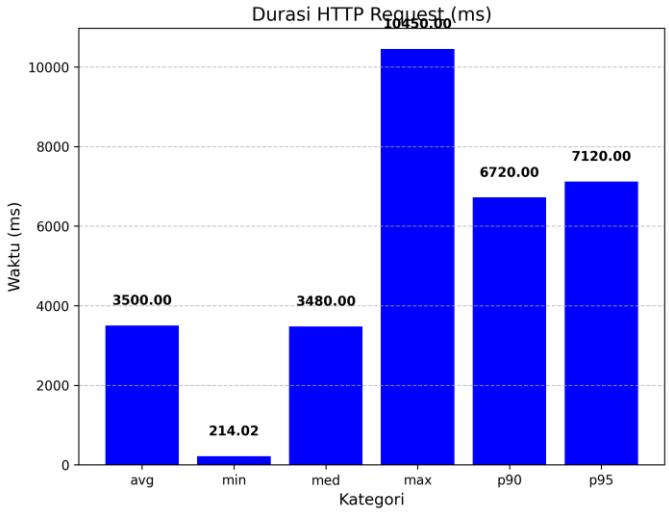
Analisis hasil menunjukkan bahwa sistem CD telah berjalan dengan stabil dan mendukung proses peluncuran fitur baru secara cepat, aman, dan otomatis. Variasi waktu eksekusi *deployment* disebabkan oleh ukuran image, kompleksitas pipeline, serta beban server pada saat *runtime*. Pipeline ini terbukti mampu meminimalkan intervensi manual, mempercepat waktu rilis fitur, dan menjaga konsistensi antar versi. Efektivitas deployment ini menunjukkan bahwa implementasi DevOps berhasil mendukung proses pengembangan yang berkelanjutan dan siap dioperasikan secara nyata dengan pengujian dan montoring pada siklus berikutnya.

D. Continuous Monitoring

Continuous Monitoring adalah tahap penting dalam siklus DevOps yang dilakukan setelah proses *Continuous Deployment* (CD). Monitoring bertujuan untuk mengevaluasi performa sistem secara berkelanjutan, memastikan bahwa API yang telah dideploy dapat berjalan stabil di lingkungan produksi, serta memberikan visibilitas kepada tim pengembang mengenai performa aplikasi secara *real-time*. Pada tahap ini, load testing menggunakan tool seperti K6 digunakan untuk mensimulasikan trafik pengguna dengan berbagai jumlah *virtual users* (VUs). Pengujian dilakukan pada seluruh fitur API, mencakup pemantauan metrik seperti waktu respons, tingkat kegagalan, beban CPU, penggunaan memori, dan konsumsi bandwidth. Pengujian dilakukan dengan meningkatkan jumlah virtual users untuk menilai bagaimana API berperforma di bawah beban trafik yang meningkat, serta untuk mengidentifikasi *bottleneck* potensial yang dapat mempengaruhi

DOI: https://doi.org/10.61124/sinta.v2i2.42

kestabilan sistem. Fitur Autentikasi adalah salah satu fitur utama dalam API yang bertanggung jawab untuk menangani proses login dan registrasi pengguna. Pengujian *load testing* dilakukan untuk mengukur performa API dalam menangani banyak permintaan autentikasi secara bersamaan. Hasil pengujian menunjukkan bahwa rata-rata durasi HTTP request untuk permintaan autentikasi adalah 812,71 ms. Waktu respons maksimum tercatat 3,54 detik, mengindikasikan adanya lonjakan waktu respons yang signifikan. Gambar 4 merupakan grafik Durasi HTTP Request pada fitur prediksi.



Gambar 4 Durasi HTTP request fitur prediksi

Pada pengujian ini, API dapat menangani rata-rata 8.516 request per detik. Meski angka ini menunjukkan performa yang stabil, terdapat indikasi bahwa jika beban pengguna meningkat, kapasitas server mungkin perlu ditingkatkan, baik melalui horizontal scaling atau meningkatkan spesifikasi server. Meskipun demikian, tidak ada request yang gagal (0% *failure rate*), yang menunjukkan bahwa API berhasil menangani semua permintaan meskipun ada lonjakan waktu respons. Hal ini menunjukkan ketahanan dan keandalan sistem, meskipun ada peningkatan beban yang dapat memengaruhi waktu respons. Meskipun terdapat penurunan performa dalam hal kecepatan, API tetap dapat menjaga kestabilan tanpa ada request yang gagal diproses selama pengujian load. Setiap fitur akan menjalani tahap yang sama seperti pada fitur autentikasi, untuk mengevaluasi performa API serta kinerja cloud server. Tabel berikut menyajikan ringkasan hasil monitoring setiap fitur.

Tabel 3 Ringkasan Hasil Monitoring Setiap Fitur.

Fitur	Waktu Respons Rata-rata	Tingkat Kegagalan (%)	Jumlah Request per Detik	Penggunaan CPU	Penggunaan Memori
Autentikasi	812.71 ms	0%	8.516	Tinggi (100%)	40%
Manajemen User	77.46 ms	24.96%	14.29	Sedang (43.81%)	Stabil (40%)
Prediksi	3.5 detik	0%	5.19	Tinggi (100%)	50%
Riwayat	201.61 ms	18.46%	76.44	60%	30%
Sharing	562.05 ms	0%	172.20	50-60%	30%

Berdasarkan tabel diatas dapat disimpulkan sebagai berikut:

1. Fitur Autentikasi : Meskipun beban CPU mencapai 100%, tidak ada request yang gagal. Waktu respons maksimal mencapai 3.54 detik saat terjadi lonjakan trafik, menunjukkan bahwa meskipun performa server stabil, optimasi untuk skala lebih besar sangat diperlukan.
2. Fitur Manajemen User : Memiliki waktu respons rata-rata yang cepat (77.46 ms), namun tingkat kegagalan cukup tinggi (24.96%). Kegagalan ini terjadi karena pengujian yang dilakukan menggunakan K6 tidak menemukan data yang diperlukan selama simulasi, yang mengarah pada tingkat kegagalan yang lebih tinggi. Setelah pengujian dioptimalkan, kesalahan ini berhasil diatasi, dan fitur Sharing yang menggunakan pendekatan serupa menunjukkan performa yang jauh lebih baik tanpa kegagalan, karena data pengujian yang lebih lengkap dan tepat.

DOI: https://doi.org/10.61124/sinta.v2i2.42

3. Fitur Prediksi : Waktu respons sistem relatif lama (rata-rata 3,5 detik) karena melibatkan pemrosesan machine learning. Meskipun penggunaan CPU mencapai 100%, tidak terdeteksi adanya kegagalan. Hal ini menunjukkan efisiensi kode, namun terdapat potensi bottleneck jika trafik meningkat tanpa adanya skalabilitas vertikal.
4. Fitur Riwayat : Beban hingga 300 Virtual Users memungkinkan sistem untuk menangani lebih dari 41 ribu request. Lalu, tingkat kegagalan sebesar 18,46% terjadi karena pengujian tidak menemukan data yang diperlukan. Hal ini mengindikasikan bahwa optimasi database atau query sangat diperlukan, serta penyesuaian metode pengujian untuk menghindari kesalahan pengambilan data.
5. Fitur Sharing : Memiliki performa terbaik dengan lebih dari 124 ribu request diproses tanpa kegagalan. Meskipun waktu respons rata-rata cukup tinggi (562.05 ms), sistem tetap stabil sementara penggunaan CPU dan memori tetap terkendali. Hasil ini menunjukkan bahwa pengujian dengan data yang lebih lengkap dan pengelolaan yang tepat dapat menghasilkan performa yang optimal.
6. *Continuous Monitoring* menunjukkan bahwa sebagian besar fitur API dapat berjalan dengan stabil di lingkungan real-time, meskipun beberapa fitur (seperti Manajemen User dan Riwayat) membutuhkan perhatian lebih lanjut terkait tingkat kegagalan yang tinggi dan optimasi beban server. Kegagalan yang terjadi pada Manajemen User dan Riwayat lebih disebabkan oleh konsep pengujian yang tidak sepenuhnya mencerminkan kondisi nyata, yang dapat diatasi dengan perbaikan pada data pengujian dan pengelolaan server. Sharing menjadi contoh bahwa dengan pengujian yang tepat, API dapat berfungsi secara optimal tanpa kegagalan. Secara keseluruhan, monitoring ini membuktikan bahwa sistem DevOps yang diterapkan berhasil menjaga stabilitas API dan memberikan visibilitas yang penting untuk pengelolaan performa aplikasi jangka panjang. Melalui load testing dan pemantauan waktu respons, penggunaan CPU, serta memori, kita dapat mengidentifikasi bottleneck dan mengambil langkah-langkah optimasi yang diperlukan agar sistem tetap scalable dan reliable dalam menghadapi trafik yang lebih besar di masa mendatang.

4. DISKUSI

Penerapan metode DevOps dalam penelitian ini menunjukkan efektivitas pendekatan *continuous development, continuous integration, continuous deployment,* dan *continuous monitoring* dalam pengembangan backend API. Meskipun demikian, terdapat beberapa hal yang perlu diperhatikan, seperti pengelolaan beban dan efisiensi kode pada beberapa fitur, seperti Manajemen Pengguna dan Riwayat Prediksi. Selain itu, meskipun API berfungsi dengan baik pada sebagian besar fitur, terdapat kebutuhan untuk peningkatan performa, terutama dalam menghadapi trafik yang lebih besar dan pengelolaan kapasitas server. Sebagian besar kegagalan yang terdeteksi pada pengujian menggunakan K6 disebabkan oleh konsep pengujian yang dirancang untuk mensimulasikan beban tinggi pada sistem, yang menyebabkan beberapa data pengujian tidak menemukan data yang diperlukan. Hal ini bukan merupakan kegagalan pada fitur itu sendiri, melainkan akibat pengujian yang dilakukan dalam kondisi simulasi yang tidak sepenuhnya mencerminkan kondisi nyata pada saat aplikasi digunakan. Pengembangan riset selanjutnya, disarankan untuk menguji penerapan skalabilitas *vertikal* dan *horizontal*, optimasi database dan q*uery*, serta integrasi sistem logging dan alerting untuk meningkatkan respons dan stabilitas sistem. Penerapan pendekatan mikroservis juga dapat menjadi alternatif untuk memecah aplikasi menjadi layanan-layanan yang lebih kecil dan lebih mudah dikelola. Pengembangan ini membuat sistem yang dibangun menjadi lebih efisien, scalable, dan siap digunakan dalam produksi skala lebih besar.

5. KESIMPULAN

Penelitian ini menunjukkan bahwa penerapan DevOps dalam pengembangan backend API pendeteksi penyakit tanaman berhasil meningkatkan efisiensi dan stabilitas sistem. Pada Continuous Development, fitur dikembangkan secara modular, memfasilitasi pengujian dan integrasi lebih cepat. Continuous Integration mengotomatiskan pengujian dengan 40+ run, mencapai tingkat keberhasilan 60%-75% pada fitur utama seperti Prediksi dan Autentikasi. Continuous Deployment memastikan tidak ada kegagalan pada tahap deployment, dengan 22 dari 23 versi berhasil dideploy, rata-rata waktu eksekusi 42 detik. Hanya satu kegagalan pada Prediksi terjadi akibat beban server tinggi. Continuous Monitoring menunjukkan API mampu menangani 8.500 request per detik pada Autentikasi dengan waktu respons rata-rata 812.71 ms, meskipun CPU mencapai 100%. Manajemen Pengguna memiliki waktu respons 77.46 ms dengan tingkat kegagalan 24.96% akibat data pengujian yang tidak lengkap. Secara keseluruhan, API tetap stabil meskipun dengan beban tinggi, dan kegagalan lebih sering disebabkan oleh data pengujian yang tidak

DOI: https://doi.org/10.61124/sinta.v2i2.42

lengkap. Penelitian ini menyarankan peningkatan skalabilitas, optimasi database, dan penerapan microservices serta logging untuk mendukung pengelolaan kapasitas server di masa depan.

DAFTAR PUSTAKA

[1] Admin, "Sensus Pertanian 2023 - Badan Pusat Statistik." Accessed: Jul. 05, 2024. [Online]. Available: https://sensus.bps.go.id/main/index/st2023

[2] Y. H. Natbais and A. B. S. Umbu, "Aplikasi Deteksi Penyakit pada Daun Tomat Berbasis Android Menggunakan Model Terlatih Tensorflow Lite," *TEKNOTAN*, vol. 17, no. 2, p. 83, Aug. 2023, doi: 10.24198/jt.vol17n2.1.

[3] N. K. Akmal and M. N. Dasaprawira, "Rancang bangun Application Programming Interface (API) menggunakan gaya arsitektur Graphql untuk pembuatan sistem informasi pendataan anggota Unit Kegiatan Mahasiswa (UKM) studi kasus UKM Starlabs," *J. SITECH Sist. Inf. Dan Teknol.*, vol. 5, no. 1, pp. 37–40, Aug. 2022, doi: 10.24176/sitech.v5i1.7937.

[4] A. Ehsan, M. A. M. E. Abuhaliqa, C. Catal, and D. Mishra, "RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions," *Appl. Sci.*, vol. 12, no. 9, p. 4369, Apr. 2022, doi: 10.3390/app12094369.

[5] Q. Aini, M. Yusup, N. P. L. Santoso, A. R. Ramdani, and U. Rahardja, "Digitalization Online Exam Cards in the Era of Disruption 5.0 using the DevOps Method," *J. Educ. Sci. Technol. EST*, pp. 67–75, Apr. 2021, doi: 10.26858/est.v7i1.18837.

[6] P. Rodríguez *et al.*, "Continuous deployment of software intensive products and services: A systematic mapping study," *J. Syst. Softw.*, vol. 123, pp. 263–291, Jan. 2017, doi: 10.1016/j.jss.2015.12.015.

[7] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *J. Syst. Softw.*, vol. 123, pp. 176–189, Jan. 2017, doi: 10.1016/j.jss.2015.06.063.

[8] Amazon Web Services, "Continuous Monitoring," AWS Documentation. Accessed: Nov. 11, 2024. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/devops-guidance/continuous-monitoring.html

[9] A. Syaikhu, "KOMPUTASI AWAN (CLOUD COMPUTING) PERPUSTAKAAN PERTANIAN," vol. 10, no. 1.

[10] H. Setiawan and A. Ashari, "Cloud Computing : Solusi ICT ?," *J. Sist. Inf. JSI*, vol. 3, no. 2, pp. 336–345, Feb. 2011.

[11] A. Zulfikar, "PENGGUNAAN DOCKER DAN KUBERNETES PIPELINE DALAM PENGEMBANGAN APLIKASI PREDIKSI CACAT PERANGKAT LUNAK MELALUI PENDEKATAN MLOPS." UIN Syarif Hidayatullah Jakarta, Dec. 28, 2022. Accessed: Nov. 05, 2024. [Online]. Available: https://repository.uinjkt.ac.id/dspace/handle/123456789/66765

[12] R. Fajrul Falah and M. Komarudin, "PERANCANGAN MICROSERVICE BERBASIS REST API PADA GOOGLE CLOUD PLATFORM MENGGUNAKAN NODEJS DAN PYTHON," *J. Inform. Dan Tek. Elektro Terap.*, vol. 11, no. 3s1, Sep. 2023, doi: 10.23960/jitet.v11i3s1.3506.

[13] A. Taryana, I. Setiawan, A. Fadli, and E. Murdyantoro, "Pioneering the automation of lnternal quality assurance system of higher education (IQAS-HE) using DevOps approach," in *2017 International Conference on Sustainable Information Engineering and Technology (SIET)*, Malang: IEEE, Nov. 2017, pp. 259–264. doi: 10.1109/SIET.2017.8304146.

[14] B. Adityo Kurniawan, A. Taryana, Y. Ramadhani, and A. Fadli, "Rancang Bangun Aplikasi Quest Board Untuk Masyarakat Menggunakan Metode Devops Berbasis Android," *J. Pendidik. Dan Teknol. Indones.*, vol. 3, no. 4, pp. 151–164, May 2023, doi: 10.52436/1.jpti.285.

[15] F. Paramudita, M. I. Zulfa, and A. Taryana, "IMPLEMENTASI DEVOPS PADA PENGEMBANGAN APLIKASI ANDROID PENDETEKSI KUALITAS BERAS BERBASIS MACHINE LEARNING," *Transm. J. Ilm. Tek. Elektro*, vol. 26, no. 3, pp. 105–113, Jul. 2024, doi: 10.14710/transmisi.26.3.105-113.

[16] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, Nov. 2010, doi: 10.1109/MIC.2010.145.

[17] M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, I. F. Rochimansyah, and A. Rokhim, "Design an MVC Model using Python for Flask Framework Development," in *2019 International Electronics Symposium (IES)*, Surabaya, Indonesia: IEEE, Sep. 2019, pp. 214–219. doi: 10.1109/ELECSYM.2019.8901656.

[18] K. Sidharta and T. Wibowo, "STUDI EFISIENSI SUMBER DAYA TERHADAP EFEKTIVITAS PENGGUNAAN DATABASE : STUDI KASUS SQL SERVER DAN MYSQL," vol. 1, 2020.

[19] C. Coronel and S. Morris, *Database systems: design, implementation, and management*, 11th edition. Stamford, CT: Cengage Learning, 2015.

[20] R. C. Davis, "Git and GitHub for Librarians," *Behav. Soc. Sci. Libr.*, vol. 34, no. 3, pp. 158–164, Jul. 2015, doi: 10.1080/01639269.2015.1062586.

[21] Github, "Understanding GitHub Actions," GitHub Docs. Accessed: Apr. 12, 2025. [Online]. Available: https://docs-internal.github.com/en/actions/about-github-actions/understanding-github-actions