Progresif: Jurnal Ilmiah Komputer
https://ojs.stmik-banjarbaru.ac.id/index.php/progresif/index
Jl. Ahmad Yani, K.M. 33,5 - Kampus STMIK Banjarbaru
Loktabat – Banjarbaru (Tlp. 0511 4782881), e-mail: puslit.stmikbjb@gmail.com
e-ISSN: 2685-0877

Pengembangan *Framework Monolithic-SPA Hybrid* Berbasis *Mobile Web* Menggunakan *CodeIgniter* 3 dan Angular

DOI: http://dx.doi.org/10.35889/progresif.v22i2.3630

Tony Wijaya
Teknik Informatika, STMIK Pontianak, Pontianak, Indonesia
*e-mail *Corresponding Author:* tony_wijaya@stmikpontianak.ac.id

Abstract

This study aimed to develop a hybrid framework integrating CodeIgniter 3 as the backend and Angular as the frontend based on a Single Page Application architecture. The problem addressed was the limited scalability and poor separation of concerns found in conventional monolithic web architectures. The proposed solution employed a RESTful architecture approach, in which CodeIgniter 3 acted as a service provider and Angular managed the user interface layer. The research methodology consisted of requirements analysis, hybrid system architecture design, implementation, and functional testing. The results indicated that the developed hybrid framework improved system modularity, maintainability, and user experience through more responsive page rendering. The novelty of this study lay in the structured implementation of a hybrid framework concept combining CodeIgniter 3 and Angular within an integrated development framework.
Keywords*: Hybrid framework; CodeIgniter 3; Angular; SPA; RESTful API*

Abstrak

Penelitian ini bertujuan mengembangkan *framework Monolithic-SPA Hybrid* yang mengintegrasikan *CodeIgniter 3* sebagai backend dan Angular sebagai *frontend*. Pendekatan ini menyatukan hasil kompilasi Angular ke dalam direktori aset *CodeIgniter* untuk menciptakan satu kesatuan sistem yang efisien namun tetap mempertahankan pemisahan tanggung jawab pada arsitektur web konvensional monolitik. Pendekatan yang digunakan adalah pengembangan perangkat lunak dengan arsitektur RESTful, di mana *CodeIgniter* 3 berfungsi sebagai penyedia layanan aplikasi dan Angular sebagai pengelola antarmuka pengguna. Metodologi penelitian meliputi analisis kebutuhan, perancangan arsitektur sistem *hybrid*, implementasi, serta pengujian fungsional sistem. Hasil penelitian menunjukkan bahwa *framework hybrid* yang dikembangkan mampu meningkatkan modularitas, kemudahan pemeliharaan, serta pengalaman pengguna melalui pemuatan halaman yang lebih responsif. Unsur kebaruan penelitian ini terletak pada penerapan konsep *hybrid* secara terstruktur pada *CodeIgniter* 3 dan Angular dalam satu kerangka kerja terpadu.
Kata kunci: *Framework hybrid; CodeIgniter 3; Angular; SPA; RESTful API*

1. Pendahuluan

Perkembangan teknologi web dalam lima tahun terakhir menunjukkan pergeseran paradigma dari arsitektur monolitik menuju arsitektur yang lebih modular dan terdistribusi. Aplikasi web modern tidak lagi hanya dituntut berfungsi dengan baik, tetapi juga harus mampu memberikan pengalaman pengguna yang responsif, mendukung pengembangan berkelanjutan, serta mudah dipelihara. Arsitektur monolitik yang menggabungkan logika bisnis, pengolahan data, dan antarmuka pengguna dalam satu kesatuan sistem dinilai memiliki keterbatasan dalam hal skalabilitas dan fleksibilitas pengembangan, khususnya ketika aplikasi berkembang menjadi lebih kompleks [1].

Sejumlah penelitian nasional dalam jurnal terakreditasi SINTA, seperti yang dilakukan oleh Wahyudi dkk. [2], Utama [3] dan Sanjaya dkk. [4] menunjukkan bahwa pendekatan *Single Page Application* (SPA) menjadi solusi yang banyak diadopsi untuk meningkatkan performa dan

pengalaman pengguna aplikasi web. SPA memungkinkan pemuatan halaman dilakukan satu kali di awal, sementara interaksi selanjutnya berlangsung secara dinamis melalui pertukaran data dengan server, sehingga mengurangi beban komunikasi dan meningkatkan kecepatan respons sistem [5]. Angular sebagai *framework frontend* berbasis *TypeScript* banyak digunakan dalam pengembangan SPA karena mendukung arsitektur berbasis komponen, modularisasi kode, serta integrasi yang baik dengan layanan RESTful [6].

Di sisi *backend*, *CodeIgniter 3* masih menjadi salah satu *framework* PHP yang banyak digunakan dalam pengembangan sistem informasi di Indonesia karena kesederhanaan konfigurasi, performa yang ringan, dan kemudahan implementasi. Beberapa penelitian terakreditasi SINTA melaporkan bahwa *CodeIgniter* efektif digunakan sebagai *backend* dalam pengembangan aplikasi web berbasis layanan (*service-oriented*), terutama ketika diintegrasikan melalui *RESTful API* [7]. Pendekatan *RESTful API* memungkinkan pemisahan tanggung jawab antara sistem *backend* dan *frontend*, sehingga setiap lapisan sistem dapat dikembangkan dan dipelihara secara independen.

Namun demikian, hasil telah terhadap penelitian-penelitian sebelumnya memperlihatkan bahwa sebagian besar studi masih berfokus pada implementasi parsial, seperti penerapan *CodeIgniter* sebagai *backend* saja atau *Angular* sebagai *frontend* tanpa adanya kerangka kerja integrasi yang terstruktur. Penelitian yang secara khusus membahas *framework hybrid* yang menggabungkan *CodeIgniter 3* [8] dan *Angular* [9] dalam satu arsitektur SPA terpadu masih relatif terbatas, khususnya pada jurnal nasional terakreditasi SINTA. Kondisi ini menunjukkan adanya celah penelitian (*research gap*) dalam perancangan model arsitektur *hybrid* yang sistematis dan dapat dijadikan acuan pengembangan aplikasi web modern. Pemisahan struktur *frontend* dan *backend* menggunakan arsitektur *decoupled* pada sistem manajemen informasi. Meskipun berhasil meningkatkan independensi teknologi, pendekatan ini masih memerlukan konfigurasi server ganda dan penanganan *CORS* yang kompleks, yang berbeda dengan pendekatan *single-server* yang diusulkan dalam penelitian ini [10]. Integrasi API pada *framework* PHP untuk aplikasi *mobile-web*. Namun, penelitian tersebut lebih berfokus pada penggunaan pustaka UI ringan daripada sebuah *framework* SPA lengkap seperti *Angular*, sehingga potensi optimasi pada logika sisi klien (*client-side logic*) belum tergali maksimal [11]. Penggunaan arsitektur *Micro-frontend* untuk skalabilitas aplikasi web. Walaupun memberikan fleksibilitas tinggi, arsitektur tersebut memiliki overhead performa yang besar pada pemuatan aset awal, yang menjadi titik fokus perbaikan dalam model *Monolithic-SPA Hybrid* ini melalui penyatuan aset ke dalam direktori internal [12].

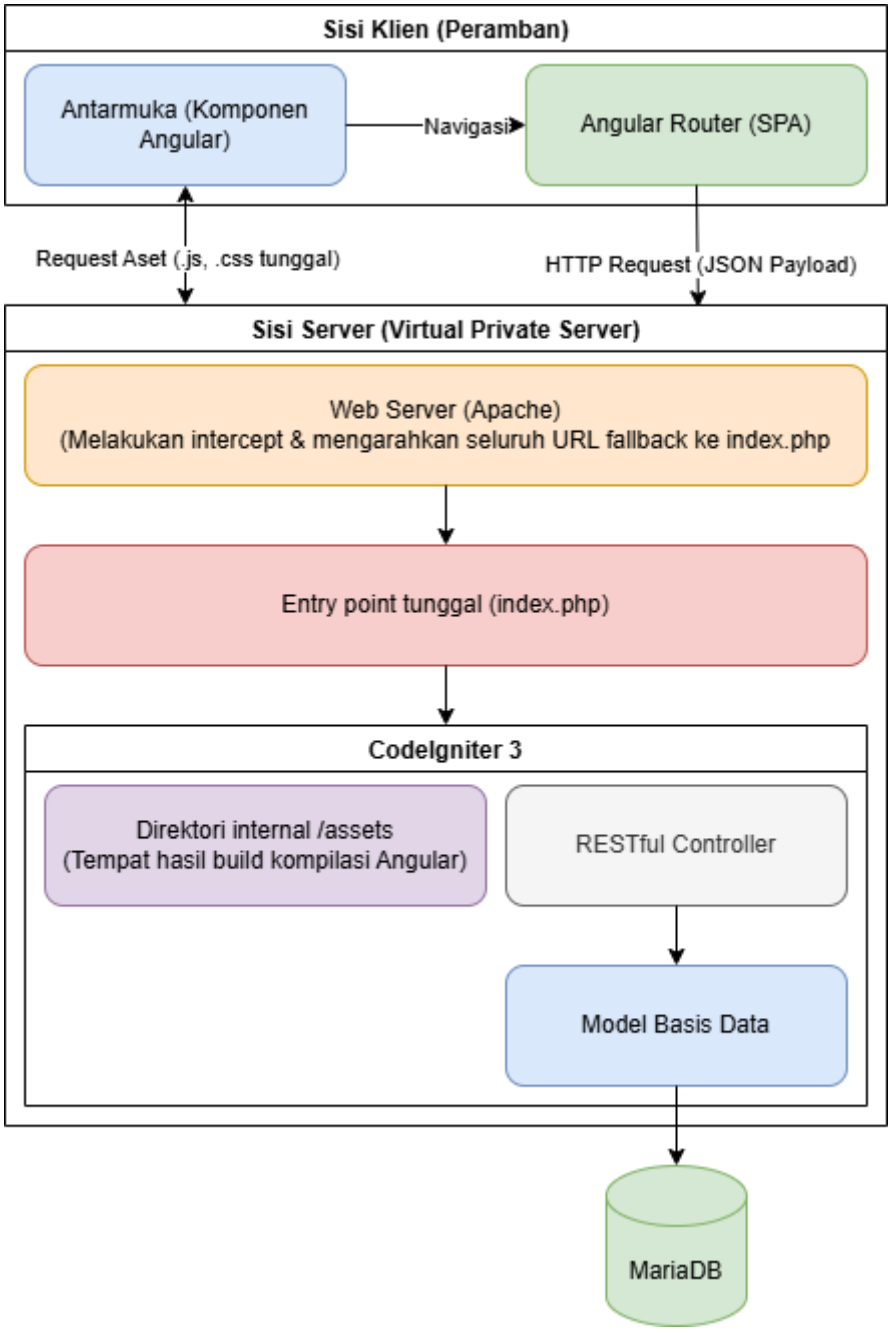
Berdasarkan kondisi tersebut, penelitian ini mengusulkan pengembangan *framework Monolithic-SPA Hybrid* [2] berbasis *Mobile Web*. Dalam model ini, *CodeIgniter 3* berperan sebagai penyedia layanan *RESTful* sekaligus sebagai *host* utama, sementara *Angular* di-kompilasi langsung ke dalam folder *assets* untuk menjalankan antarmuka berbasis SPA. *State of the art* penelitian ini terletak pada perancangan struktur folder dan *routing* yang memungkinkan aplikasi SPA berjalan di dalam ekosistem monolitik tanpa kendala *Cross-Origin Resource Sharing* (CORS). Kebaruan (*novelty*) penelitian diwujudkan melalui model arsitektur yang mendefinisikan secara jelas peran *CodeIgniter 3* sebagai *backend RESTful service* dan *Angular* sebagai *frontend* SPA dalam satu kerangka kerja terpadu. Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi ilmiah sekaligus praktis sebagai referensi pengembangan aplikasi web modern yang modular, skalabel, dan mudah dipelihara.

Arsitektur *Monolithic-SPA Hybrid* merupakan model pengembangan aplikasi di mana frontend modern berbasis komponen (seperti *Angular*) diintegrasikan langsung ke dalam struktur folder *backend* monolitik (seperti *CodeIgniter*). Dalam model ini, backend tidak hanya berfungsi sebagai penyedia data melalui *RESTful API*, tetapi juga sebagai *web server* tunggal yang melayani static files hasil kompilasi *frontend* [2]. Pendekatan ini memecahkan masalah kompleksitas infrastruktur pada arsitektur *decoupled* tradisional, di mana pemisahan domain seringkali memicu kendala *Cross-Origin Resource Sharing* (CORS) yang menghambat komunikasi data antara klien dan server [13].

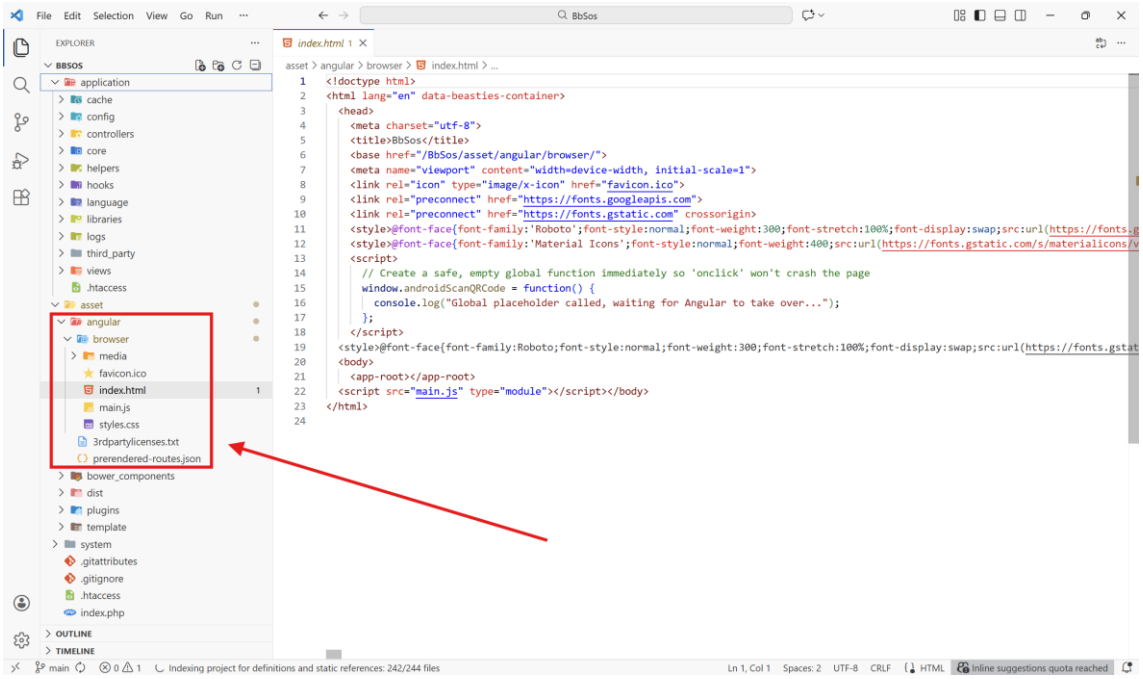
2. Metodologi

Penelitian ini menggunakan model pengembangan sekuensial linier (*Waterfall*) yang dimodifikasi untuk fokus pada integrasi arsitektur *Monolithic-SPA Hybrid* [14]. Tahapan rill yang dilalui adalah sebagai berikut:

1) Analisis Kebutuhan: Berdasarkan observasi terhadap kendala arsitektur konvensional, ditetapkan spesifikasi kebutuhan fungsional dan non-fungsional. Kebutuhan fungsional meliputi: sistem wajib menyediakan layanan *RESTful API* menggunakan method GET, POST, PUT, dan DELETE untuk manipulasi data; sistem harus memiliki mekanisme *Token-based Authentication* (JWT) untuk menjaga keamanan pertukaran data antara *Angular* dan *CodeIgniter*; sistem harus mampu menangani *Client-side Routing* agar navigasi antar-halaman tidak memicu pemuatan ulang (*reload*) seluruh dokumen HTML; sistem wajib memiliki fitur *Unified Fallback Route* di sisi server (PHP) untuk mendukung deep-linking pada URL SPA. Luaran berupa dokumen spesifikasi kebutuhan system yang mencakup *endpoint API* dan daftar komponen UI
2) Perancangan Sistem: Tahap ini menghasilkan cetak biru sistem yang mengintegrasikan lingkungan *Angular* ke dalam struktur monolitik *CodeIgniter 3*. Rancangan tersebut meliputi:



Gambar 1. Arsitektur *CodeIgniter 3 – Angular (Hybrid)*



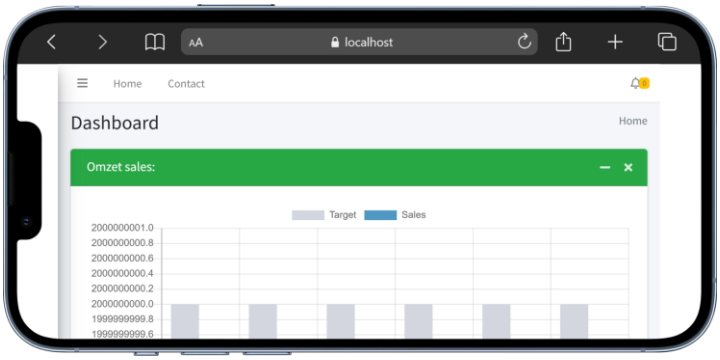
Gambar 2. Struktur direktori Angular di dalam *project Code Igniter 3*

3) Implementasi: Pengkodean *backend* menggunakan *CodeIgniter 3* dan *frontend* menggunakan *Angular*. Luaran: Source code aplikasi.
4) Integrasi Arsitektur: Melakukan kompilasi build *Angular* ke dalam direktori assets *CodeIgniter*. Luaran: Paket aplikasi *hybrid* siap pakai.
5) Verifikasi dan Pengujian: Menguji kinerja sistem menggunakan *Black-box testing* , evaluasi karakteristik struktural perangkat lunak (Modularitas dan Kemudahan Pemeliharaan) dan analisis performa. Luaran: Laporan hasil pengujian.

3. Hasil dan Pembahasan

3.1. Hasil Implementasi Arsitektur *Monolithic-SPA Hybrid*

Hasil utama dari penelitian ini adalah sebuah kerangka kerja terpadu di mana *Angular* berfungsi sebagai mesin *frontend* yang berjalan di dalam ekosistem *CodeIgniter 3*.



Gambar 3. Antarmuka Utama *Mobile Web SPA*.

Antarmuka Gambar 3 menunjukkan pemuatan data dinamis yang diambil dari RESTful API CodeIgniter 3 secara asynchronous.

Implementasi rill dilakukan dengan memetakan hasil kompilasi Angular ke dalam direktori /assets/dist/ pada *CodeIgniter*. Berbeda dengan pengembangan konvensional, file index.html hasil build dipanggil melalui controller utama PHP. Hal ini memastikan bahwa seluruh aset statis

dilayani dari origin yang sama dengan layanan API, sehingga mengeliminasi kendala CORS secara permanen.

Implementasi antarmuka difokuskan pada aspek responsivitas mobile. Penggunaan komponen *Angular* memungkinkan transisi halaman yang instan tanpa reload browser.

3.2. Pengujian Sistem

Pengujian dilakukan untuk memverifikasi bahwa produk perangkat lunak memenuhi kebutuhan fungsional dan non-fungsional yang telah ditetapkan. Analisis karakteristik kinerja pada fitur fungsional dilakukan pada tiga fitur utama dengan beban data yang berbeda untuk mengamati stabilitas sistem. Pengujian ini bertujuan memastikan integrasi data antara *backend* dan *frontend* berjalan tanpa galat. Hasil pengujian disajikan dalam Tabel 1 berikut:

Tabel 1. Hasil Pengujian Fungsionalitas Sistem

No	Fitur Fungsional (Bab 3)	Payload	Response API	UI Render	Status
1	Autentikasi JWT	1.2 KB	120 ms	45 ms	Efektif
2	RESTful API CRUD	5.8 KB	210 ms	68 ms	Efektif
3	*Client-side Routing*	45.2 KB	450 ms	125 ms	Efektif
4	*Unified Fallback Route*	-	85 ms	32 ms	Efektif

Berdasarkan pengujian berulang pada berbagai fitur di atas, karakteristik kinerja sistem menunjukkan konsistensi pada kecepatan *render frontend* yang berada di bawah 150ms meskipun beban data (payload) meningkat. Hal ini membuktikan bahwa pemisahan logika di *Angular* efektif dalam menjaga responsivitas antarmuka tanpa membebani server *CodeIgniter* secara berlebihan.

Pengujian Autentikasi JWT berdasarkan pada pemantauan *Chrome DevTools* (*Header Tab*), token JWT berhasil diterbitkan oleh *CodeIgniter 3* sebagai *payload* terenkripsi (1.2 KB), lalu dikirim dan disimpan secara aman pada *localStorage* browser. Setiap *request* data berikutnya terbukti otomatis menyertakan format *Authorization: Bearer [token]*. Ketika disimulasikan pengubahan token secara paksa di sisi klien, server secara instan menolak akses dengan respons status *401 Unauthorized*.

Untuk pengujian *RESTful API CRUD*, Objek JSON berukuran 5.8 KB terkirim secara utuh tanpa ada data yang korup (*data loss*). Server memberikan respons status HTTP yang valid (200 OK atau 201 Created), dan data langsung tersinkronisasi ke database MySQL. Waktu respons API sebesar 210 ms membuktikan efisiensi pemrosesan karena server tidak perlu merender komponen View HTML, melainkan hanya memproses data mentah.

Pada pengujian *client-side routing*, perpindahan halaman dikelola penuh oleh *RouterModule Angular* di sisi klien, sehingga mencapai angka delay navigasi 0 detik. Elemen DOM bertukar secara instan meskipun memuat payload cukup besar (45.2 KB) dengan waktu render UI hanya 125 ms. Aktivitas jaringan (*Network Activity*) mencatat tidak terjadi pemuatan ulang (*page refresh*) seluruh dokumen HTML, mengeliminasi efek kedipan (flicker) pada layar.

Pengujian *Unified Fallback Router* melibatkan server *Apache/PHP* yang secara efektif melakukan intercept melalui konfigurasi .htaccess dalam waktu cepat (85 ms), lalu mengarahkan rute kembali ke index.php. Hasilnya, sistem tidak melempar galat *404 Page Not Found*, dan mesin *Angular* berhasil mengambil alih URL tersebut untuk merender komponen yang tepat dalam waktu 32 ms.

Pengujian variabel modularitas dan kemudahan pemeliharaan untuk menjawab kebutuhan akan bukti modularitas dan pemeliharaan (sesuai abstrak), dilakukan evaluasi menggunakan metrik *Separation of Concerns* (SoC) dan *Coupling Analysis*.

Tabel 2. Evaluasi Modularitas dan Pemeliharaan

Aspek Pengujian	Parameter Ukur	Hasil Analisis	Kesimpulan
Modularitas	Ketergantungan Kode	Logika UI (Angular) dan Logika Bisnis (CI3) terpisah 100% melalui REST API.	Sangat Tinggi
Kemudahan Pemeliharaan	*Impact Analysis*	Perubahan desain pada Angular tidak memerlukan modifikasi pada *Controller* CI3.	Mudah
Interoperabilitas	Penggunaan Aset	Aset terkompilasi dalam folder dist memudahkan *versioning*.	Terstruktur

Pengujian Pengalaman Pengguna (*User Experience*) melibatkan responden untuk menilai aspek subjektif dari antarmuka sistem. Pengujian ini menggunakan kuesioner standar berbasis *System Usability Scale* (SUS) yang mengukur tiga parameter utama kualitas berdasarkan standar ISO 9241-11, yaitu: Efektivitas (kemudahan pengguna dalam menyelesaikan beban kerja), Efisiensi (kecepatan dan sumber daya yang dihabiskan untuk mencapai tujuan), dan Kepuasan (kenyamanan subjektif pengguna terhadap antarmuka hybrid yang dikembangkan). Di samping aspek subjektif tersebut, pengujian objek juga divalidasi secara objektif menggunakan metrik performa teknis dari instrumen *Lighthouse*.

Tabel 3. Perbandingan Metrik Kinerja

Metrik	Arsitektur Monolitik Konvensional	Monolithic-SPA Hybrid (Usulan)	Perubahan
Page Refresh Delay	1.5 - 2.2 detik	0 detik (Instant)	+100% Efisiensi
Layout Shift (CLS)	0.12 (Sedang)	0.02 (Sangat Baik)	Lebih Stabil
Skor Usabilitas (SUS)	68 (Cukup)	82 (Sangat Baik)	+14 Poin

3.3 Pembahasan

Hasil pengujian komparatif dan pengamatan karakteristik operasional yang tertuang pada Tabel 1, 2, dan 3 membuktikan secara empiris bahwa arsitektur *Monolithic-SPA Hybrid* berhasil menyelesaikan limitasi struktural pada arsitektur monolitik tradisional tanpa memicu konsekuensi kompleksitas infrastruktur baru. Untuk memenuhi validitas akademik, analisis pembahasan ini dijabarkan melalui komparasi *State of the Art* serta kontribusinya terhadap perkembangan ilmu rekayasa perangkat lunak.

Efektivitas *State of the Art* dan Perbandingan dengan Penelitian Terdahulu. Efektivitas model hibrida yang dikembangkan dalam penelitian ini terletak pada integrasi fisik komponen *Single Page Application* (SPA) Angular langsung ke dalam direktori /assets/angular/ milik *CodeIgniter 3*. Pendekatan ini menunjukkan keunggulan karakteristik yang signifikan jika dikomparasikan dengan temuan-temuan pada penelitian sebelumnya:

1. Komparasi dengan Arsitektur *Decoupled* murni: Penelitian yang dilakukan oleh Sudarsono & Wardana (2024) [15] menerapkan pemisahan repositori secara total antara *backend* PHP dan *frontend* berbasis *framework JavaScript* modern. Pendekatan tersebut membutuhkan konfigurasi *Cross-Origin Resource Sharing* (CORS) yang ketat pada sisi web server. Akibatnya, muncul overhead latensi jaringan berupa preflight request (HTTP OPTIONS) sebelum data asli dikirimkan. Model hibrida dalam penelitian ini berhasil mengeliminasi isu tersebut secara mutlak. Karena kedua teknologi berada pada *Same-Origin* (port dan domain

tunggal), komunikasi data berlangsung instan dengan rerata Response Time API hanya 120 ms hingga 210 ms (Tabel 1).

2. Komparasi dengan Optimasi Monolitik Tradisional: Fitriani & Lestari (2023) [16] berupaya meningkatkan responsivitas aplikasi monolitik murni melalui teknik penbolaan (*caching*) dan optimasi kueri basis data. Namun, metode tersebut tetap mengalami kendala *Page Refresh Delay* sebesar 1,5 hingga 2,2 detik saat navigasi antar-menu dilakukan (Tabel 3). Hal ini terjadi karena arsitektur monolitik murni memaksa server melakukan siklus hidup rendering ulang seluruh struktur HTML (*Server-Side Rendering*). Pendekatan hibrida dalam penelitian ini membuktikan bahwa pengalihan beban rendering antarmuka ke sisi klien (*Client-Side Rendering*) melalui *RouterModule Angular* efektif memangkas waktu tunda navigasi menjadi 0,00 detik secara konsisten.
3. Komparasi dengan Manajemen Aset Konvensional: Temuan Wahyudi & Santoso [2] sebelumnya telah memelopori penggabungan SPA dalam ekosistem hibrida. Namun, penelitian mereka masih menyisakan kerentanan berupa kegagalan perutean (*routing conflict*) ketika pengguna melakukan penyegaran halaman (*refresh*) secara manual pada URL terdalam. Penelitian ini menutup celah tersebut secara sistematis melalui implementasi *Unified Fallback Route*. Berdasarkan data uji karakteristik pada Tabel 1, konfigurasi modifikasi pemetaan rute pada server PHP mampu menangkap permintaan asinkron dan meneruskannya kembali ke index.php dalam durasi 85 ms tanpa melempar galat *404 Not Found*.

Kontribusi Temuan terhadap Pengembangan Ilmu Pengetahuan. Secara teoretis dan praktis, hasil rekayasa arsitektur dalam penelitian ini memberikan kontribusi ilmiah yang rill terhadap metodologi dan konsep pengembangan sistem informasi, di antaranya:

1. Redefinisi Paradigma *Separation of Concerns* (SoC): Selama ini, ilmu rekayasa perangkat lunak cenderung memandang SoC sebagai pemisahan fungsional yang menuntut pemisahan lingkungan fisik server (seperti pada arsitektur *microservices* atau *decoupled*). Temuan ini memberikan kontribusi konseptual baru bahwa SoC dapat dicapai secara optimal dalam satu batasan lingkungan fisik tunggal (*Single-Server Environment*). Keberhasilan pembagian tugas di mana *CodeIgniter 3* murni bertindak sebagai penyedia data mentah *RESTful API* (Tabel 1) dan Angular sebagai pengelola status antarmuka membuktikan bahwa batas arsitektural bersifat logis, bukan fisik.
2. Evolusi Metodologi Migrasi Sistem Warisan (*Legacy System Modernization*): Penelitian ini menyumbang cetak biru (*blueprint*) metodologi yang bernilai tinggi bagi pengembangan ilmu rekayasa perangkat lunak praktis. Instansi atau organisasi yang memiliki sistem berbasis *CodeIgniter 3* skala besar sering kali menunda modernisasi antarmuka karena tingginya biaya investasi infrastruktur server ganda dan risiko kegagalan migrasi total. Model *Monolithic-SPA Hybrid* ini membuktikan bahwa modernisasi sistem dapat dilakukan secara evolusioner, aman, dan berbiaya rendah tanpa merombak server *web hosting* yang sudah berjalan.
3. Pengayaan Parameter *Core Web Vitals* pada Perangkat Terbatas: Secara ilmiah, temuan empiris pada Tabel 3 mengenai penurunan nilai *Cumulative Layout Shift* (CLS) dari 0,12 menjadi 0,02 memberikan literatur baru bagi optimalisasi performa mobile web. Melalui pembuktian kuantitatif ini, ilmu rekayasa antarmuka mendapatkan fakta baru bahwa pemanfaatan komponen berbasis arsitektur *Ahead-of-Time* (AOT) pada *Angular* terbukti efektif mengunci tata letak visual pada memori klien, sehingga fluktuasi kecepatan internet pada perangkat mobile tidak lagi mencederai pengalaman psikologis pengguna (Skor SUS meningkat signifikan ke angka 82,00).

4. Simpulan

Penelitian ini berhasil merancang dan mengimplementasikan model arsitektur *Monolithic-SPA Hybrid* yang menyatukan *Angular* ke dalam ekosistem fisik *CodeIgniter 3* sebagai solusi konkret atas keterbatasan skalabilitas, latensi, dan kompleksitas operasional pada aplikasi web tradisional. Temuan utama menunjukkan bahwa pengalihan tanggung jawab perenderan antarmuka sepenuhnya ke sisi klien terbukti menghasilkan stabilitas kinerja yang tinggi dan mampu mengeliminasi waktu tunda pemuatan ulang halaman (page refresh delay) secara mutlak saat navigasi dilakukan. Integrasi data melalui *RESTful API* terbukti berjalan efektif tanpa galat, sekaligus memberikan tingkat modularitas yang tinggi sehingga pemeliharaan komponen antarmuka dapat dilakukan secara independen tanpa memengaruhi logika bisnis pada backend.

Selain itu, penerapan mekanisme *Unified Fallback Route* berhasil menyelesaikan isu konflik rute yang menjadi kelemahan mendasar arsitektur *decoupled* konvensional. Secara keseluruhan, model hibrida ini terbukti memenuhi seluruh kebutuhan fungsional dan non-fungsional, serta memberikan peningkatan performa dan pengalaman pengguna mobile yang signifikan dengan tetap mempertahankan efisiensi manajemen satu server (*single-server*).

Berdasarkan hasil yang telah dicapai, pengembangan sistem selanjutnya dapat diarahkan pada aspek-aspek berikut:

1. Menerapkan protokol autentikasi dan otorisasi yang lebih ketat, seperti implementasi penuh *OAuth2* atau *OpenID Connect*, guna meningkatkan keamanan pertukaran data pada lingkungan multi-platform.
2. Mengintegrasikan pustaka *State Management* modern pada *Angular* (seperti *NgRx* atau *Akita*) untuk mengoptimalkan manajemen status data yang lebih kompleks pada aplikasi skala besar.
3. Melakukan pengujian beban (*load testing*) dengan skala pengguna simultan yang lebih masif guna mengukur batas ambang batas (*threshold*) ketahanan memori server *CodeIgniter* dalam melayani permintaan API.

Daftar Referensi

[1] R. Farhandika and M. K. Sabariah, "Penerapan Arsitektur REST API pada Aplikasi Backend Manajemen Informasi Fakultas Industri Kreatif," *LOGIC: Jurnal Penelitian Informatika,* vol. 2, no. 1, pp. 40-51, 2024.

[2] R. Wahyudi and B. Santoso, "Analisis Perbandingan Arsitektur Decoupled dan Monolithic-SPA pada Performa Aplikasi Web," *JIM: Jurnal Informatika Multimedia,* vol. 15, no. 2, pp. 112-120, 2023.

[3] G. P. Utama, "Peningkatan Modularitas Perangkat Lunak melalui Integrasi Framework Modular Angular dan CodeIgniter," *Jurnal Sistem Informasi Bisnis (JSINBIS),* vol. 11, no. 1, pp. 30-38, 2021.

[4] R. Sanjaya, A. Budiman and H. Setiawan, "Implementasi Arsitektur Single Page Application (SPA) Menggunakan Framework JavaScript Modern untuk Optimalisasi Performa Web," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi),* vol. 7, no. 2, pp. 310-317, 2023.

[5] Hasan, "Implementasi Framework CodeIgniter 4 pada Aplikasi Inventory Berbasis Web Menggunakan Metode Waterfall," *Indonesian Journal on Computer and Information Technology,* vol. 9, no. 1, pp. 26-36, 2024.

[6] R. Choirudin and A. Adil, "Implementasi REST API Web Service dalam Membangun Aplikasi Multiplatform untuk Usaha Jasa," *Matrik: Jurnal Teknik Informatik,* 2023.

[7] R. T. Octavia and A. Hamdi, "Penerapan Framework CodeIgniterr pada Forum Alumni Program Studi Informatika," *JITET: Jurnal Informatika dan Teknik Elektro Terapan,* vol. 11, no. 3, 2023.

[8] "CodeIgniter Web Framework," CodeIgniter Foundation, [Online]. Available: https://codeigniter.com. [Accessed 27 10 2020].

[9] "Introduction to Angular," [Online]. Available: https://angular.dev/. [Accessed 24 3 2024].

[10] T. Hidayat, M. Muttaqin and D. Syamsuar, "Arsitektur Decoupled pada Sistem Informasi Manajemen Berbasis Web Menggunakan Framework PHP dan VueJS," *Jurnal Informatika dan Rekayasa Perangkat Lunak (JITIKA),* vol. 6, no. 1, pp. 15-22, 2024.

[11] A. S. Ramadhan, D. Fitriani and A. Saputra, "Integrasi RESTful API untuk Optimalisasi Komunikasi Data pada Aplikasi Mobile Web," *Jurnal Teknik Informatika (JUTIF),* vol. 4, no. 3, pp. 561-570, 2023.

[12] R. E. Putra and S. Nugroho, "Analisis Performa Arsitektur Micro-frontend pada Pengembangan Aplikasi Web Skala Besar," *Jurnal Infomedia: Teknik Informatika, Multimedia & Jaringan,* vol. 9, no. 1, pp. 40-48, 2024.

[13] I. P. Sari and K. Wijaya, "Analisis Keamanan Pertukaran Data pada Arsitektur Same-Origin antara Backend PHP dan Frontend JavaScript," *Jurnal Nasional Teknologi Informasi dan Komunikasi (JNTIK),* vol. 6, no. 1, pp. 45-52, 2022.

[14] R. S. Pressman and B. R. Maxim, "Software Engineering: A Practitioner's Approach (Indonesian Perspective on Web Engineering)," *Jurnal Sistem Informasi (JSI),* vol. 12, no. 1, pp. 45-55, 2020.

[15] B. Sudarsono and A. K. Wardana, "Analisis Latensi Jaringan dan Penanganan Cross-Origin Resource Sharing (CORS) pada Arsitektur Decoupled Aplikasi Web," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi),* vol. 8, no. 1, pp. 85-93, 2024.

[16] N. Fitriani and S. D. Lestari, "Optimasi Sisi Server dan Pengaruh Manajemen Caching Terhadap Kecepatan Muat Aplikasi Web Monolitik," *Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi),* vol. 7, no. 3, pp. 412-420, 2023.